

# **Manuel de référence canonique de Csound**

**Version 6.14.0**

**Csound Community, [csound.com](http://csound.com)**

---

# **Manuel de référence canonique de Csound: Version 6.14.0**

par Csound Community

---

# Table des matières

Préface .....	xxxv
Préface du manuel de Csound .....	xxxv
Histoire du manuel de référence canonique de Csound .....	xxxvi
Mentions de copyright .....	xxxviii
Débuter avec Csound .....	xxxix
Les nouveautés de Csound 6.14.0 .....	xli
I. Vue d'ensemble .....	1
Introduction .....	4
La commande Csound .....	5
Ordre de priorité .....	5
Description de la syntaxe de la commande .....	5
Ligne de commande de Csound .....	7
Options de ligne de commande (par catégorie) .....	19
Variables d'environnement de Csound .....	32
Format de fichier unifié pour les orchestres et les partitions .....	34
Description .....	34
Exemple .....	37
Fichier de paramètres de ligne de commande (.csound6rc) .....	38
Prétraitement du fichier de partition .....	38
La fonction Extract .....	38
Prétraitement indépendant avec Scsort .....	38
Utiliser Csound .....	40
Sortie console de Csound .....	40
Comment Csound fonctionne .....	42
Valeurs d'amplitude dans Csound .....	43
Audio en temps réel .....	44
Entrées/sorties en temps réel sous Linux .....	45
Mac OSX .....	51
Windows .....	52
Entrées/sorties en temps réel avec le kit de connexion de JACK .....	53
Optimisation de la latence audio en E/S .....	54
Configuration .....	56
Syntaxe de l'orchestre .....	57
Instructions de l'en-tête de l'orchestre .....	58
Instructions de bloc d'instrument et d'opcode .....	58
Instructions ordinaires .....	59
Types, constantes et variables .....	59
Initialisation de variable .....	60
Expressions .....	61
Répertoires et fichiers .....	61
Nomenclature .....	62
Macros .....	62
Instruments nommés .....	63
Opcodes définis par l'utilisateur (UDO) .....	65
Vecteurs et tableaux .....	66
Syntaxe fonctionnelle dans Csound6 .....	66
Serveur UDP .....	67
La partition numérique standard .....	70
Prétraitement des partitions standard .....	70
Carry .....	70
Tempo .....	71

Sort .....	71
Instructions de partition .....	72
Symboles next-P et previous-P .....	73
Ramping .....	73
Macros de partition .....	74
Partition dans plusieurs fichiers .....	76
Evaluation des expressions .....	77
Chaînes de caractères dans les p-champs .....	78
Frontaux .....	80
CsoundAC .....	81
Construire Csound .....	84
Liens Csound .....	85
II. Vue d'ensemble des opcodes .....	86
Générateurs de signal .....	90
Synthèse/resynthèse additive .....	90
Oscillateurs élémentaires .....	90
Oscillateurs à spectre dynamique .....	90
Synthèse FM .....	91
Synthèse granulaire .....	91
Synthèse Hyper Vectorielle .....	92
Générateurs linéaires et exponentiels .....	92
Générateurs d'enveloppe .....	93
Modèles et émulations .....	93
Phaseurs .....	94
Générateurs de nombres aléatoires (de bruit) .....	95
Reproduction de sons échantillonnés .....	96
Soundfonts .....	97
Synthèse par balayage .....	98
Accès aux tables .....	99
Synthèse par terrain d'ondes .....	100
Modèles physiques par guide d'onde .....	100
Entrée et sortie de signal .....	101
Entrées et sorties fichier .....	101
Entrée de signal .....	101
Sortie de signal .....	101
Bus logiciel .....	102
Impression et affichage .....	102
Requêtes sur les fichiers sons .....	102
Modificateurs de signal .....	104
Modificateurs d'amplitude et traitement des dynamiques .....	104
Convolution et morphing .....	104
Retard .....	104
Panoramique et spatialisation .....	105
Réverbération .....	106
Opérateurs du niveau échantillon .....	107
Limiteurs de signal .....	107
Effets spéciaux .....	107
Filtres standard .....	108
Filtres spécialisés .....	110
Guides d'onde .....	110
Distorsion non-linéaire et distorsion de phase .....	110
Contrôle d'instrument .....	112
Contrôle d'horloge .....	112
Valeurs conditionnelles .....	112



Instructions de contrôle de durée .....	112
Widgets FLTK et contrôleurs GUI .....	112
Conteneurs FLTK .....	115
Valuateurs FLTK .....	115
Autres widgets FLTK .....	116
Modifier l'apparence des widgets FLTK .....	116
Opcodes généraux relatifs aux widgets FLTK .....	117
Appel d'instrument .....	117
Contrôle séquentiel d'un programme .....	118
Contrôle de l'exécution en temps réel .....	119
Initialisation et réinitialisation .....	119
Détection et contrôle .....	120
Piles .....	121
Contrôle de sous-instrument .....	121
Lecture du temps .....	121
Contrôle des tables de fonction .....	123
Requêtes sur une table .....	123
Opérations de lecture/écriture de table .....	123
Lecture de table avec sélection dynamique .....	124
Opérations mathématiques .....	125
Conversion d'amplitude .....	125
Opérations arithmétiques et logiques .....	125
Compareurs et accumulateurs .....	125
Fonctions mathématiques .....	126
Opcodes équivalents à des fonctions .....	126
Fonctions aléatoires .....	126
Fonctions trigonométriques .....	127
Opcodes d'algèbre linéaire .....	128
Opcodes de tableaux .....	136
Conversion des hauteurs .....	144
Fonctions .....	144
Opcodes de hauteurs .....	144
Support MIDI en temps réel .....	145
Clavier virtuel MIDI .....	146
Entrée MIDI .....	149
Sortie de message MIDI .....	150
Entrée et sortie génériques .....	150
Convertisseurs .....	150
Extension d'évènements .....	150
Sortie de note-on/note-off .....	151
Opcodes pour l'interopérabilité MIDI/partition .....	151
Messages système temps réel .....	152
Banques de réglettes .....	152
Traitement spectral .....	154
Resynthèse par transformée de Fourier à court-terme (STFT) .....	154
Resynthèse par codage prédictif linéaire (LPC) .....	154
Traitement spectral non-standard .....	155
Outils pour le traitement spectral en temps réel (opcodes pvs) .....	155
Traitement spectral avec ATS .....	157
Opcodes Loris .....	157
Opcodes spectraux basés sur des tableaux .....	161
Chaînes de caractères .....	162
Opcodes de manipulation de chaîne .....	163
Opcodes de conversion de chaîne .....	164

Opcodes vectoriels .....	165
Opérateurs de tableaux de vecteurs .....	165
Opérations entre un signal vectoriel et un signal scalaire .....	165
Opérations entre deux signaux vectoriels .....	166
Générateurs vectoriels d'enveloppe .....	166
Limitation et enroulement des signaux vectoriels de contrôle .....	167
Chemins de retard vectoriel au taux de contrôle .....	167
Générateurs de signal aléatoire vectoriel .....	167
Système de patch zak .....	168
Accueil de greffon .....	169
DSSI et LADSPA pour Csound .....	169
OSC et réseau .....	170
Opcodes Ableton Link .....	170
OSC .....	171
Réseau .....	171
Opcodes pour le traitement à distance .....	171
Opcodes Mixer .....	172
Opcodes de graphe de fluence .....	173
Opcodes Jacko .....	176
Opcodes Python .....	179
Introduction .....	179
Syntaxe de l'orchestre .....	179
Opcodes pour le traitement d'image .....	181
Opcodes STK .....	182
Opcodes divers .....	184
III. Référence .....	185
Opcodes et opérateurs de l'orchestre .....	214
!= .....	215
#define .....	217
#include .....	221
#undef .....	224
#ifdef .....	225
#ifndef .....	226
\$NOM .....	227
% .....	230
&& .....	232
> .....	234
>= .....	236
lastcycle .....	238
< .....	240
<= .....	242
* .....	244
+ .....	246
- .....	248
/ .....	250
= .....	252
+= .....	254
== .....	256
^ .....	258
.....	260
! .....	262
0dbfs .....	264
A4 .....	267
<< .....	268

>>	270
&	271
	273
¬	274
#	275
a	276
abs	278
active	280
adsr	284
adsyn	287
adsynt	289
adsynt2	292
aftouch	295
alpass	297
alwayson	299
ampdb	302
ampdbfs	304
ampmidi	306
ampmidicurve	308
ampmidid	310
areson	312
aresonk	315
atone	317
atonek	319
atonex	321
ATSadd	323
ATSaddnz	327
ATSbufread	330
ATScross	332
ATSinfo	335
ATSinterpread	338
ATSread	340
ATSreadnz	344
ATSpartialtap	346
ATSSinnoi	348
babo	352
balance	356
balance2	358
bamboo	360
barmodel	362
bbcutm	364
bbcuts	369
beadsynt	372
beosc	376
betarand	379
bexprnd	382
bformenc1	384
bformdec1	387
binit	390
biquad	392
biquada	397
birnd	400
bpf	402
bpfcos	405

bqrez .....	408
butbp .....	410
butbr .....	411
buthp .....	412
butlp .....	413
butterbp .....	414
butterbr .....	416
butterhp .....	418
butterlp .....	420
button .....	422
buzz .....	424
c2r .....	426
cabasa .....	428
cauchy .....	430
cauchy1 .....	432
ceil .....	434
cell .....	436
cent .....	439
centroid .....	441
ceps .....	443
cepsinv .....	445
cggoto .....	447
chanctrl .....	449
changed .....	451
changed2 .....	453
chani .....	456
chano .....	457
cbrt .....	458
chebyshevpoly .....	460
checkbox .....	463
chn .....	465
chnclear .....	467
chnexport .....	469
chnget .....	471
chnmix .....	474
chnparams .....	476
chnset .....	477
chuap .....	480
cigoto .....	484
ckgoto .....	486
clear .....	488
clfilt .....	490
clip .....	493
clockoff .....	496
clockon .....	498
cmp .....	500
cmplxprod .....	503
cngoto .....	505
comb .....	507
combinv .....	509
compress .....	511
compileorc .....	514
compilestr .....	516
compress2 .....	518

compilecsd .....	521
connect .....	523
control .....	526
convle .....	527
convolve .....	528
copya2ftab .....	532
copyf2array .....	534
cos .....	536
cosseg .....	538
cossegb .....	540
cossegr .....	542
cosh .....	544
cosinv .....	546
cps2pch .....	548
cpsmidi .....	552
cpsmidib .....	554
cpsmidinn .....	556
cpsoct .....	560
cpspch .....	563
cpstmid .....	566
cpstun .....	568
cpstuni .....	571
cpsexpch .....	574
cpumeter .....	578
cpuprc .....	580
cross2 .....	583
crossfm .....	585
crunch .....	588
ctrl14 .....	590
ctrl21 .....	592
ctrl7 .....	594
ctrlinit .....	597
cuserrnd .....	598
dam .....	601
date .....	604
dates .....	606
db .....	608
dbamp .....	610
dbfsamp .....	612
dcblock .....	614
dcblock2 .....	616
dconv .....	618
dct .....	620
dctinv .....	622
deinterleave .....	624
delay .....	626
delay1 .....	628
delayk .....	630
delayr .....	633
delayw .....	635
deltap .....	637
deltap3 .....	640
deltapi .....	643
deltapn .....	646

deltapx .....	648
deltapxw .....	650
denorm .....	652
diff .....	654
diode_ladder .....	656
directory .....	659
diskgrain .....	661
diskin .....	664
diskin2 .....	667
dispfft .....	671
display .....	673
distort .....	675
distort1 .....	677
divz .....	679
doppler .....	681
dot .....	683
downsamp .....	684
dripwater .....	686
dssiactivate .....	688
dssiaudio .....	691
dssictls .....	693
dssiinit .....	695
dssilist .....	697
dumpk .....	699
dumpk2 .....	702
dumpk3 .....	705
dumpk4 .....	708
duserrnd .....	711
dust .....	713
dust2 .....	715
else .....	717
elseif .....	719
endif .....	721
endin .....	723
endop .....	725
envlpx .....	728
envlpxr .....	731
ephasor .....	734
eqfil .....	736
evalstr .....	738
event .....	739
event_i .....	743
exciter .....	745
exitnow .....	747
exp .....	749
expcurve .....	751
expon .....	753
exprand .....	755
exprandi .....	757
expseg .....	759
expsega .....	761
expsegb .....	763
expsegba .....	765
expsegr .....	767

faustaudio .....	769
faustcompile .....	771
faustctl .....	773
faustdsp .....	775
faustgen .....	777
faustplay .....	779
fareylen .....	781
fareyleni .....	783
ficlose .....	786
filebit .....	788
filelen .....	790
filenchnls .....	792
filepeak .....	794
filescale .....	796
filesr .....	798
filevalid .....	800
fillarray .....	802
fft .....	804
filter2 .....	806
fin .....	808
fini .....	810
fink .....	812
fiopen .....	814
flanger .....	816
flashtxt .....	818
FLbox .....	820
FLbutBank .....	825
FLbutton .....	828
FLcloseButton .....	833
FLcolor .....	836
FLcolor2 .....	838
FLcount .....	839
FLexecButton .....	842
FLgetsnap .....	846
FLgroup .....	847
FLgroupEnd .....	849
FLgroup_end .....	850
FLhide .....	851
FLhvsBox .....	852
FLhvsBoxSetValue .....	853
FLjoy .....	854
FLkeyIn .....	857
FLknob .....	859
FLlabel .....	864
FLloadsnap .....	866
FLmouse .....	867
flooper .....	869
flooper2 .....	871
floor .....	873
FLpack .....	875
FLpackEnd .....	878
FLpack_end .....	879
FLpanel .....	880
FLpanelEnd .....	884

FLpanel_end .....	885
FLprintk .....	886
FLprintk2 .....	887
FLroller .....	888
FLrun .....	891
FLsavesnap .....	892
FLscroll .....	898
FLscrollEnd .....	901
FLscroll_end .....	902
FLsetAlign .....	903
FLsetBox .....	904
FLsetColor .....	906
FLsetColor2 .....	908
FLsetFont .....	909
FLsetPosition .....	911
FLsetSize .....	912
FLsetsnap .....	913
FLsetSnapGroup .....	915
FLsetText .....	916
FLsetTextColor .....	918
FLsetTextSize .....	919
FLsetTextType .....	920
FLsetVal_i .....	923
FLsetVal .....	924
FLshow .....	925
FLslidBnk .....	926
FLslidBnk2 .....	930
FLslidBnkGetHandle .....	934
FLslidBnkSet .....	935
FLslidBnkSetk .....	936
FLslidBnk2Set .....	938
FLslidBnk2Setk .....	939
FLslider .....	942
FLtabs .....	948
FLtabsEnd .....	954
FLtabs_end .....	955
FLtext .....	956
FLupdate .....	959
fluidAllOut .....	960
fluidCCi .....	962
fluidCCk .....	964
fluidControl .....	966
fluidEngine .....	969
fluidInfo .....	972
fluidLoad .....	974
fluidNote .....	976
fluidOut .....	978
fluidProgramSelect .....	981
fluidSetInterpMethod .....	984
FLvalue .....	986
FLvkeybd .....	989
FLvslidBnk .....	990
FLvslidBnk2 .....	994
FLxyin .....	997



fmanal .....	1000
fmax .....	1002
fmb3 .....	1004
fmbell .....	1006
fmin .....	1009
fmmetal .....	1011
fmod .....	1014
fmpercfl .....	1016
fmrhode .....	1018
fmvoice .....	1021
fmwurlie .....	1023
fof .....	1026
fof2 .....	1029
fofilter .....	1035
fog .....	1037
fold .....	1040
follow .....	1042
follow2 .....	1044
foscil .....	1046
foscili .....	1048
fout .....	1050
fouti .....	1054
foutir .....	1056
foutk .....	1058
fprintks .....	1060
fprints .....	1066
frac .....	1068
fractalnoise .....	1070
framebuffer .....	1072
freeverb .....	1074
ftaudio .....	1076
ftchnls .....	1079
ftconv .....	1081
ftcps .....	1084
ftexists .....	1086
ftfree .....	1088
ftgen .....	1090
ftgenonce .....	1093
ftgentmp .....	1095
ftlen .....	1097
ftload .....	1099
ftloadk .....	1100
ftlptim .....	1101
ftmorf .....	1103
ftom .....	1105
ftprint .....	1107
ftsamplerbank .....	1109
ftsaver .....	1111
ftsaverk .....	1113
ftslice .....	1114
ftsr .....	1116
gain .....	1118
gainslider .....	1120
gtf .....	1122

gauss .....	1123
gaussi .....	1125
gausstrig .....	1127
gbuzz .....	1130
genarray .....	1132
genarray_i .....	1134
gendy .....	1136
gendyc .....	1140
gendyx .....	1143
getcfig .....	1147
getcol .....	1149
getftargs .....	1151
getrow .....	1153
getrowlin .....	1156
getseed .....	1159
gogobel .....	1160
goto .....	1162
grain .....	1164
grain2 .....	1166
grain3 .....	1171
granule .....	1176
guiro .....	1180
harmon .....	1182
harmon2 .....	1184
hdf5read .....	1187
hdf5write .....	1189
hilbert .....	1191
hilbert2 .....	1196
hrtfearly .....	1198
hrtfmove .....	1202
hrtfmove2 .....	1205
hrtfreverb .....	1208
hrtfstat .....	1211
hsboscil .....	1214
hvs1 .....	1217
hvs2 .....	1221
hvs3 .....	1227
hypot .....	1230
i .....	1232
if .....	1233
fftinv .....	1238
igoto .....	1240
ihold .....	1242
imagecreate .....	1244
imagefree .....	1246
imagegetpixel .....	1248
imageload .....	1251
imagesave .....	1253
imagesetpixel .....	1255
imagesize .....	1257
in .....	1259
in32 .....	1261
inch .....	1262
inh .....	1264

init .....	1265
initc14 .....	1268
initc21 .....	1269
initc7 .....	1270
inleta .....	1272
inletk .....	1275
inletkid .....	1277
inletf .....	1278
inletv .....	1279
ino .....	1280
inq .....	1281
inrg .....	1283
ins .....	1284
insremot .....	1286
insglobal .....	1289
instr .....	1291
int .....	1293
integ .....	1295
interleave .....	1297
interp .....	1299
invalue .....	1302
inx .....	1304
inz .....	1305
JackoAudioIn .....	1306
JackoAudioInConnect .....	1307
JackoAudioOut .....	1308
JackoAudioOutConnect .....	1309
JackoFreewheel .....	1310
JackoInfo .....	1311
JackoInit .....	1313
JackoMidiInConnect .....	1315
JackoMidiOutConnect .....	1316
JackoMidiOut .....	1317
JackoNoteOut .....	1318
JackoOn .....	1319
JackoTransport .....	1320
jacktransport .....	1321
jitter .....	1323
jitter2 .....	1325
joystick .....	1327
jspline .....	1330
k .....	1332
K35_hpf .....	1333
K35_lpf .....	1337
kgoto .....	1341
kr .....	1343
ksmps .....	1344
lenarray .....	1345
lfo .....	1347
limit .....	1349
limit1 .....	1351
line .....	1353
linen .....	1355
linenr .....	1357

lineto .....	1360
linlin .....	1362
lincos .....	1364
link_beat_force .....	1366
link_beat_get .....	1367
link_beat_request .....	1368
link_create .....	1369
link_enable .....	1371
link_is_enabled .....	1373
link_metro .....	1375
link_peers .....	1377
link_tempo_get .....	1379
link_tempo_set .....	1381
linrand .....	1383
linseg .....	1385
linsegb .....	1387
linsegr .....	1389
liveconv .....	1391
locsend .....	1394
locsig .....	1397
log .....	1400
log10 .....	1402
log2 .....	1404
logbtwo .....	1406
logcurve .....	1408
loop_ge .....	1410
loop_gt .....	1411
loop_le .....	1412
loop_lt .....	1415
loopseg .....	1418
loopsegp .....	1421
looptseg .....	1423
loopxseg .....	1425
lorenz .....	1427
lorisread .....	1430
lorismorph .....	1433
lorisplay .....	1436
loscil .....	1439
loscil3 .....	1442
loscilx .....	1445
lowpass2 .....	1448
lowres .....	1450
lowresx .....	1452
lpf18 .....	1454
lpfreson .....	1456
lphasor .....	1458
lpinterp .....	1460
lposcil .....	1461
lposcil3 .....	1463
lposcila .....	1465
lposcilsa .....	1467
lposcilsa2 .....	1469
lpread .....	1471
lpreson .....	1474

lpshold .....	1477
lpsholdp .....	1479
lpslot .....	1480
mac .....	1481
maca .....	1483
madsr .....	1485
mags .....	1489
mandel .....	1492
mandol .....	1495
maparray .....	1497
marimba .....	1500
massign .....	1503
max .....	1506
maxabs .....	1508
maxabsaccum .....	1510
maxaccum .....	1512
maxalloc .....	1514
max_k .....	1516
maxarray .....	1518
mclock .....	1520
mdelay .....	1522
median .....	1524
mediank .....	1526
metro .....	1528
metro2 .....	1530
mfb .....	1533
midglobal .....	1535
midiarp .....	1536
midic14 .....	1538
midic21 .....	1540
midic7 .....	1542
midichannelaftertouch .....	1544
midichn .....	1546
midicontrolchange .....	1549
midictrl .....	1551
mididefault .....	1553
midiin .....	1555
midifilestatus .....	1558
midinoteoff .....	1559
midinoteoncps .....	1561
midinoteonkey .....	1563
midinoteonoct .....	1565
midinoteonpch .....	1567
midion2 .....	1569
midion .....	1571
midiont .....	1574
midiont_i .....	1576
midipitchbend .....	1578
midipolyaftertouch .....	1581
midiprogramchange .....	1584
miditempo .....	1586
midremot .....	1588
min .....	1591
minabs .....	1593

minabsaccum .....	1595
minaccum .....	1597
mincer .....	1599
minarray .....	1601
mirror .....	1603
MixerSetLevel .....	1605
MixerSetLevel_i .....	1608
MixerGetLevel .....	1609
MixerSend .....	1611
MixerReceive .....	1613
MixerClear .....	1615
mode .....	1617
modmatrix .....	1620
monitor .....	1625
moog .....	1627
moogladder .....	1629
moogladder2 .....	1631
moogvcf .....	1633
moogvcf2 .....	1635
moscil .....	1637
mp3in .....	1639
mp3len .....	1641
mp3scal .....	1643
mpulse .....	1645
mrtmsg .....	1647
mtof .....	1648
mton .....	1650
multitap .....	1652
mute .....	1654
mvchpf .....	1656
mvclpf1 .....	1658
mvclpf2 .....	1660
mvclpf3 .....	1662
mvclpf4 .....	1664
mxadsr .....	1666
nchnls .....	1669
nchnls_hw .....	1671
nchnls_i .....	1672
nestedap .....	1674
nlfilt .....	1677
nlfilt2 .....	1680
noise .....	1683
noteoff .....	1686
noteon .....	1687
noteondur2 .....	1688
noteondur .....	1690
notnum .....	1692
nreverb .....	1694
nrpn .....	1697
nsamp .....	1699
nstance .....	1701
ntof .....	1704
ntom .....	1706
nstrnum .....	1708

nstrstr .....	1709
ntrpol .....	1710
octave .....	1712
octcps .....	1714
octmidi .....	1717
octmidib .....	1719
octmidinn .....	1721
octpch .....	1724
olabuffer .....	1727
opcode .....	1729
oscbnk .....	1735
oscil1 .....	1741
oscil1i .....	1743
oscil3 .....	1745
oscil .....	1747
oscili .....	1749
oscilikt .....	1751
osciliktp .....	1753
oscilikts .....	1755
osciln .....	1757
oscils .....	1759
oscilx .....	1761
OSCbundle .....	1762
OSCcount .....	1764
OSCinit .....	1766
OSCinitM .....	1768
OSClisten .....	1770
OSCCraw .....	1774
OSCsend .....	1776
out32 .....	1778
out .....	1779
outc .....	1781
outch .....	1783
outh .....	1786
outiat .....	1787
outic14 .....	1789
outic .....	1791
outipat .....	1793
outipb .....	1794
outipc .....	1796
outkat .....	1798
outkc14 .....	1800
outkc .....	1801
outkpat .....	1803
outkpb .....	1804
outkpc .....	1806
outleta .....	1809
outletf .....	1811
outletk .....	1812
outletkid .....	1814
outletv .....	1815
outo .....	1816
outq1 .....	1817
outq2 .....	1819

outq3 .....	1821
outq4 .....	1823
outq .....	1825
outrg .....	1827
outs1 .....	1829
outs2 .....	1831
outs .....	1833
outvalue .....	1835
outx .....	1837
outz .....	1838
p5gconnect .....	1839
p5gdata .....	1841
p .....	1843
pan2 .....	1845
pan .....	1847
pareq .....	1849
partials .....	1852
partikkel .....	1854
partikkelget .....	1865
partikkelset .....	1868
partikkelsync .....	1871
passign .....	1875
paulstretch .....	1877
pcauchy .....	1879
pchbend .....	1881
pchmidi .....	1883
pchmidib .....	1885
pchmidinn .....	1887
pchoct .....	1890
pchtom .....	1893
pconvolve .....	1895
pcount .....	1898
pdclip .....	1901
pdhalf .....	1904
pdhalfy .....	1907
peak .....	1911
pgmassign .....	1913
phaser1 .....	1917
phaser2 .....	1920
phasor .....	1924
phasorbnk .....	1926
phs .....	1928
pindex .....	1931
pinker .....	1935
pinkish .....	1936
pitch .....	1939
pitchamdf .....	1942
planet .....	1945
platerev .....	1948
plltrack .....	1950
pluck .....	1952
poisson .....	1954
pol2rect .....	1958
polyaft .....	1961



polynomial .....	1963
port .....	1966
portk .....	1968
poscil3 .....	1970
poscil .....	1973
pow .....	1975
powershape .....	1977
powoftwo .....	1979
prealloc .....	1981
prepiano .....	1983
print .....	1986
printf .....	1988
printk2 .....	1990
printk .....	1992
printks .....	1994
printks2 .....	1997
prints .....	2000
printarray .....	2002
product .....	2005
product .....	2007
pset .....	2008
ptable .....	2010
ptablei .....	2012
ptable3 .....	2015
ptablew .....	2016
ptrack .....	2019
puts .....	2021
pvadd .....	2023
pvbufread .....	2027
pvcross .....	2029
pvinterp .....	2032
pvoc .....	2035
pvread .....	2038
pvsadsyn .....	2040
pvsanal .....	2043
pvsarp .....	2046
pvsbandp .....	2049
pvsbandr .....	2051
pvsbin .....	2053
pvsblur .....	2055
pvsbuffer .....	2057
pvsbufread .....	2058
pvsbufread2 .....	2061
pvscale .....	2063
pvscent .....	2065
pvsceps .....	2067
pvsccross .....	2069
pvsdemix .....	2071
pvsdiskin .....	2073
pvsdisp .....	2075
pvsfilter .....	2077
pvsfread .....	2080
pvsfreeze .....	2082
pvsftr .....	2084

pvsftw .....	2086
pvsfwrite .....	2089
pvsgain .....	2091
pvshift .....	2093
pvsifd .....	2095
pvsinfo .....	2097
pvsinit .....	2099
pvsin .....	2100
pvslock .....	2101
pvsmaska .....	2103
pvmix .....	2105
pvmorph .....	2107
pvsMOOTH .....	2110
pvsout .....	2112
pvsosc .....	2113
pvspitch .....	2116
pvstanal .....	2119
pvstencil .....	2121
pvstrace .....	2123
pvsvoc .....	2125
pvsynth .....	2127
pvsWarp .....	2129
pvs2tab .....	2131
pyassign Opcodes .....	2133
pycall Opcodes .....	2134
pyeval Opcodes .....	2138
pyexec Opcodes .....	2139
pyinit Opcodes .....	2142
pyrun Opcodes .....	2143
pwd .....	2145
qinf .....	2147
qnan .....	2149
r2c .....	2151
rand .....	2153
randh .....	2155
randi .....	2158
random .....	2160
randomh .....	2162
randomi .....	2165
rbjeq .....	2168
readclock .....	2171
readf .....	2173
readfi .....	2175
readk .....	2177
readk2 .....	2180
readk3 .....	2183
readk4 .....	2186
readscore .....	2189
readscratch .....	2191
rect2pol .....	2193
reinit .....	2195
release .....	2197
remoteport .....	2198
remove .....	2199

repluck .....	2200
reshapearray .....	2202
reson .....	2204
resonk .....	2206
resonr .....	2208
resonx .....	2212
resonxk .....	2214
resony .....	2216
resonz .....	2218
resyn .....	2222
return .....	2224
reverb .....	2225
reverb2 .....	2227
reverb3 .....	2228
rewindscore .....	2230
rezzy .....	2232
rfft .....	2234
rifft .....	2237
rigoto .....	2239
riturn .....	2240
rms .....	2242
rnd .....	2244
rnd31 .....	2246
round .....	2252
rspline .....	2254
rtclock .....	2256
S .....	2258
s16b14 .....	2259
s32b14 .....	2261
samphold .....	2263
sandpaper .....	2265
scale .....	2267
scalearray .....	2269
scanhammer .....	2271
scans .....	2272
scantable .....	2276
scanu .....	2278
schedkwhen .....	2280
schedkwhennamed .....	2283
schedule .....	2286
schedulek .....	2289
schedwhen .....	2290
scoreline .....	2292
scoreline_i .....	2294
sc_lag .....	2296
sc_lagud .....	2298
sc_phasor .....	2300
sc_trig .....	2302
seed .....	2304
sekere .....	2306
select .....	2308
semitone .....	2310
sense .....	2312
sensekey .....	2313

serialBegin .....	2317
serialEnd .....	2319
serialFlush .....	2320
serialPrint .....	2321
serialRead .....	2322
serialWrite_i .....	2324
serialWrite .....	2325
seqtime2 .....	2327
seqtime .....	2330
setcol .....	2333
setctrl .....	2335
setksmps .....	2338
setrow .....	2340
setscorepos .....	2343
sfilist .....	2345
sfinstr3 .....	2347
sfinstr3m .....	2350
sfinstr .....	2353
sfinstrm .....	2356
sfload .....	2358
sflooper .....	2361
sfpassign .....	2364
sfplay3 .....	2367
sfplay3m .....	2370
sfplay .....	2373
sfplaym .....	2375
sfplist .....	2378
sfpreset .....	2380
shaker .....	2383
shiftin .....	2385
shiftout .....	2387
signum .....	2389
sin .....	2391
sinh .....	2393
sininv .....	2395
sinsyn .....	2397
sleighbells .....	2399
slicearray .....	2401
slider16 .....	2403
slider16f .....	2405
slider16table .....	2407
slider16tablef .....	2409
slider32 .....	2411
slider32f .....	2413
slider32table .....	2415
slider32tablef .....	2417
slider64 .....	2419
slider64f .....	2421
slider64table .....	2423
slider64tablef .....	2425
slider8 .....	2427
slider8f .....	2429
slider8table .....	2431
slider8tablef .....	2433

sliderKawai .....	2435
sndloop .....	2436
sndwarp .....	2438
sndwarpst .....	2442
sockrecv .....	2446
socksend .....	2448
sorta .....	2450
sortd .....	2451
soundin .....	2452
space .....	2455
spat3d .....	2460
spat3di .....	2469
spat3dt .....	2473
spdist .....	2478
specaddm .....	2482
specdiff .....	2483
specdisp .....	2484
specfilt .....	2485
spechist .....	2486
specptrk .....	2487
specscal .....	2489
specsum .....	2490
spectrum .....	2491
splitrig .....	2493
sprintf .....	2495
sprintfk .....	2497
spsend .....	2499
sqrt .....	2502
squnewave .....	2504
sr .....	2509
statevar .....	2511
stix .....	2513
STKBandedWG .....	2515
STKBeeThree .....	2517
STKBlowBotl .....	2519
STKBlowHole .....	2521
STKBowed .....	2523
STKBrass .....	2525
STKClarinet .....	2527
STKDrummer .....	2529
STKFlute .....	2531
STKFMVoices .....	2533
STKHevyMetl .....	2535
STKMandolin .....	2537
STKModalBar .....	2539
STKMoog .....	2541
STKPercFlut .....	2543
STKPlucked .....	2545
STKResonate .....	2547
STKRhodey .....	2549
STKSaxofony .....	2551
STKShakers .....	2553
STKSimple .....	2555
STKSitar .....	2557

STKStifKarp .....	2559
STKTubeBell .....	2561
STKVoicForm .....	2563
STKWhistle .....	2565
STKWurley .....	2567
strchar .....	2569
strchark .....	2571
strcpy .....	2572
strcpyk .....	2573
strcat .....	2575
strcatk .....	2577
strcmp .....	2579
strcmpk .....	2581
streson .....	2582
strfromurl .....	2584
strget .....	2586
strindex .....	2588
strindexk .....	2589
strlen .....	2591
strlenk .....	2592
strlower .....	2593
strlowerk .....	2595
strrindex .....	2596
strrindexk .....	2598
strset .....	2599
strstrip .....	2601
strsub .....	2603
strsubk .....	2605
strtod .....	2606
strtodk .....	2607
strtol .....	2608
strtolk .....	2609
strupper .....	2610
strupperk .....	2611
subinstr .....	2612
subinstrinit .....	2615
sum .....	2616
sumarray .....	2618
svfilter .....	2620
syncgrain .....	2623
syncloop .....	2626
syncphasor .....	2628
system .....	2632
tab .....	2634
tabifd .....	2636
table .....	2638
table3 .....	2640
tablecopy .....	2641
tablefilter .....	2644
tablefilteri .....	2646
tablegpw .....	2648
tablei .....	2649
tableicopy .....	2652
tableigpw .....	2653

tableikt .....	2654
tableimix .....	2657
tablekt .....	2659
tablemix .....	2662
tableng .....	2664
tablera .....	2666
tableseg .....	2669
tableshuffle .....	2671
tablew .....	2674
tablewa .....	2677
tablewkt .....	2680
tablexkt .....	2682
tablexseg .....	2685
tabmorph .....	2687
tabmorpha .....	2690
tabmorphak .....	2693
tabmorphi .....	2696
tabplay .....	2699
tabrec .....	2700
tabrowlin .....	2702
tabsum .....	2705
tab2array .....	2707
tab2pvs .....	2709
tambourine .....	2711
tan .....	2713
tanh .....	2715
taninv .....	2717
taninv2 .....	2719
tbvcf .....	2721
tempest .....	2724
tempo .....	2727
temposcal .....	2729
tempoval .....	2731
tigoto .....	2733
timedseq .....	2735
timeinstk .....	2738
timeinsts .....	2740
timek .....	2742
times .....	2744
timeout .....	2747
tival .....	2749
tlineto .....	2751
tone .....	2753
tonek .....	2755
tonex .....	2757
trandom .....	2759
tradsyn .....	2761
transeg .....	2763
transegb .....	2765
transegr .....	2767
trcross .....	2769
trfilter .....	2771
trhighest .....	2773
trigger .....	2775

trigseq .....	2777
trim .....	2781
trirand .....	2783
trlowest .....	2785
trmix .....	2787
trscale .....	2789
trshift .....	2791
trsplt .....	2793
turnoff .....	2795
turnoff2 .....	2797
turnon .....	2800
tvconv .....	2801
unirand .....	2803
until .....	2806
unwrap .....	2808
upsamp .....	2811
urandom .....	2813
urd .....	2816
vactrol .....	2819
vadd .....	2821
vadd_i .....	2824
vaddv .....	2826
vaddv_i .....	2829
vaget .....	2831
valpass .....	2833
vaset .....	2836
vbap .....	2838
vbapmove .....	2841
vbapg .....	2844
vbapgmove .....	2847
vbap16 .....	2850
vbap16move .....	2852
vbap4 .....	2854
vbap4move .....	2857
vbap8 .....	2860
vbap8move .....	2862
vbaplsinit .....	2865
vbapz .....	2868
vbapzmove .....	2870
vcella .....	2872
vco .....	2875
vco2 .....	2878
vco2ft .....	2882
vco2ift .....	2884
vco2init .....	2886
vcomb .....	2889
vcopy .....	2892
vcopy_i .....	2895
vdelay .....	2897
vdelay3 .....	2899
vdelayx .....	2901
vdelayxq .....	2903
vdelayxs .....	2905
vdelayxw .....	2907



vdelayxwq .....	2909
vdelayxws .....	2911
vdelayk .....	2913
vdivv .....	2914
vdivv_i .....	2917
vecdelay .....	2919
veloc .....	2920
vexp .....	2922
vexp_i .....	2925
vexpseg .....	2927
vexpv .....	2929
vexpv_i .....	2932
vibes .....	2934
vibr .....	2936
vibrato .....	2938
vincr .....	2940
vlimit .....	2943
vlinseg .....	2944
vlowres .....	2946
vmap .....	2948
vmirror .....	2950
vmult .....	2951
vmult_i .....	2955
vmultv .....	2958
vmultv_i .....	2961
voice .....	2963
vosim .....	2966
vphaseseg .....	2971
vport .....	2973
vpow .....	2974
vpow_i .....	2978
vpowv .....	2981
vpowv_i .....	2984
vpvoc .....	2986
vrandh .....	2989
vrandi .....	2992
vsubv .....	2995
vsubv_i .....	2998
vtable1k .....	3000
vtablei .....	3002
vtablek .....	3004
vtablea .....	3006
vtablewi .....	3008
vtablewk .....	3009
vtablewa .....	3011
vtabi .....	3013
vtabk .....	3015
vtaba .....	3017
vtabwi .....	3019
vtabwk .....	3020
vtabwa .....	3021
vwrap .....	3022
waveset .....	3023
websocket .....	3025

weibull .....	3027
wgbow .....	3030
wgbowedbar .....	3032
wgbrass .....	3034
wgclar .....	3036
wgflute .....	3038
wgpluck .....	3040
wgpluck2 .....	3043
wguide1 .....	3045
wguide2 .....	3048
while .....	3051
wiiconnect .....	3053
wiidata .....	3055
wiirange .....	3058
wiisend .....	3059
window .....	3061
wrap .....	3064
writescratch .....	3066
wterrain .....	3068
xadsr .....	3070
xin .....	3072
xout .....	3074
xscanmap .....	3076
xscansmap .....	3079
xscans .....	3080
xscanu .....	3084
xtratim .....	3088
xyin .....	3092
xyscale .....	3094
zacl .....	3096
zakinit .....	3098
zamod .....	3101
zar .....	3103
zarg .....	3105
zaw .....	3107
zawm .....	3109
zdf_1pole .....	3112
zdf_1pole_mode .....	3114
zdf_2pole .....	3116
zdf_2pole_mode .....	3119
zdf_ladder .....	3121
zfilter2 .....	3124
zir .....	3126
ziw .....	3128
ziwm .....	3130
zkcl .....	3132
zkmod .....	3134
zkr .....	3136
zkw .....	3138
zkwm .....	3140
Instructions de partition et routines GEN .....	3143
Instructions de partition .....	3143
Instruction a (ou instruction avancer) .....	3144
Instruction b .....	3146

Instruction C .....	3148
Instruction d (instruction de note) .....	3150
Instruction e .....	3152
Instruction f (ou instruction de table de fonction) .....	3154
Instruction i (instruction d'instrument ou de note) .....	3157
Instruction m (instruction de marquage) .....	3161
Instruction n .....	3163
Instruction q .....	3165
Instruction r (instruction répéter) .....	3167
Instruction s .....	3169
Instruction t (instruction de tempo) .....	3171
Instruction v .....	3173
Instruction x .....	3175
Instruction y (ou instruction graine) .....	3177
Instruction { .....	3179
Instruction } .....	3182
Routines GEN .....	3182
GEN01 .....	3186
GEN02 .....	3189
GEN03 .....	3191
GEN04 .....	3194
GEN05 .....	3197
GEN06 .....	3199
GEN07 .....	3201
GEN08 .....	3203
GEN09 .....	3205
GEN10 .....	3208
GEN11 .....	3211
GEN12 .....	3213
GEN13 .....	3216
GEN14 .....	3220
GEN15 .....	3223
GEN16 .....	3231
GEN17 .....	3234
GEN18 .....	3236
GEN19 .....	3239
GEN20 .....	3241
GEN21 .....	3244
GEN23 .....	3248
GEN24 .....	3250
GEN25 .....	3252
GEN27 .....	3254
GEN28 .....	3256
GEN30 .....	3259
GEN31 .....	3261
GEN32 .....	3263
GEN33 .....	3265
GEN34 .....	3268
GEN40 .....	3271
GEN41 .....	3273
GEN42 .....	3275
GEN43 .....	3277
GEN49 .....	3278
GEN51 .....	3280

GEN52 .....	3283
GEN53 .....	3286
GENtanh .....	3288
GENexp .....	3290
GENsone .....	3292
GENquadbezier .....	3294
GENfarey .....	3297
GENwave .....	3302
GENpadsynth .....	3305
Opcodes de l'orchestre expérimentaux et routines GEN .....	3309
Opcodes de l'orchestre expérimentaux .....	3309
cudanal .....	3310
cudasynth .....	3313
cudaslicing .....	3315
Opcodes de l'orchestre et routines GEN obsolètes .....	3317
Opcodes de l'orchestre obsolètes .....	3317
abetarand .....	3318
abexprnd .....	3319
acauchy .....	3320
aexprand .....	3321
agauss .....	3322
agogobel .....	3323
alinrand .....	3324
apcauchy .....	3325
apoisson .....	3326
apow .....	3327
array .....	3328
atrirand .....	3330
aunirand .....	3331
aweibull .....	3332
bformdec .....	3333
bformenc .....	3335
clock .....	3337
hrtfer .....	3338
ibetarand .....	3340
ibexprnd .....	3341
icauchy .....	3342
ictrl14 .....	3343
ictrl21 .....	3344
ictrl7 .....	3345
iexprand .....	3346
igauss .....	3347
ilinrand .....	3348
imidic14 .....	3349
imidic21 .....	3350
imidic7 .....	3351
instimek .....	3352
instimes .....	3353
ioff .....	3354
ion .....	3355
iondur2 .....	3356
iondur .....	3357
ioutat .....	3358
ioutc14 .....	3359

ioutc .....	3360
ioutpat .....	3361
ioutpb .....	3362
ioutpc .....	3363
ipcauchy .....	3364
ipoisson .....	3365
ipow .....	3366
is16b14 .....	3367
is32b14 .....	3368
islider16 .....	3369
islider32 .....	3370
islider64 .....	3371
islider8 .....	3372
itablecopy .....	3373
itablegpw .....	3374
itablemix .....	3375
itablew .....	3376
itrirand .....	3377
iunirand .....	3378
iweibull .....	3379
kbetarand .....	3380
kbexprnd .....	3381
kcauchy .....	3382
kdump2 .....	3383
kdump3 .....	3384
kdump4 .....	3385
kdump .....	3386
kexprand .....	3387
kfilter2 .....	3388
kgauss .....	3389
klinrand .....	3390
kon .....	3391
koutat .....	3392
koutc14 .....	3393
koutc .....	3394
koutpat .....	3395
koutpb .....	3396
koutpc .....	3397
kpcachy .....	3398
kpoisson .....	3399
kpow .....	3400
kread2 .....	3401
kread3 .....	3402
kread4 .....	3403
kread .....	3404
ktableseg .....	3405
ktrirand .....	3406
kunirand .....	3407
kweibull .....	3408
sndload .....	3409
peakk .....	3411
pop .....	3412
pop_f .....	3414
push .....	3415

push_f .....	3417
soundout .....	3418
soundouts .....	3420
stack .....	3421
tb .....	3423
tableiw .....	3426
Routines GEN obsolètes .....	3429
GEN22 .....	3430
Les programmes utilitaires .....	3431
Répertoires. ....	3431
Formats des fichiers son. ....	3431
Génération d'un fichier d'analyse (ATSA, CVANAL, HETRO, LPANAL, PVA-	
NAL) .....	3432
Requêtes sur un fichier (SNDINFO) .....	3443
Conversion de fichier (, HET_EXPORT, HET_IMPORT, PVLOOK, PV_EX-	
PORT, PV_IMPORT, SDIF2AD, SRCONV) .....	3444
Autres utilitaires de Csound (CS, CSB64ENC, ENVEXT, EXTRACTOR, MA-	
KECSD, MIXER, SCALE, MKDB) .....	3463
Cscore .....	3477
Événements, listes et opérations .....	3477
Ecrire un programme de contrôle Cscore .....	3480
Compiler un programme Cscore .....	3485
Exemples plus avancés .....	3488
Csbeats .....	3490
IV. Référence Rapide des Opcodes .....	3495
Référence Rapide des Opcodes .....	3497
A. Liste des exemples .....	3559
B. Conversion de hauteur .....	3604
C. Valeurs d'intensité du son .....	3608
D. Valeurs de formant .....	3609
E. Rapports de fréquence modale .....	3613
F. Fonctions fenêtres .....	3615
G. Format de fichier SoundFont2 .....	3620
Glossaire .....	3621

---

# Préface

## Table des matières

Préface du manuel de Csound .....	xxxv
Histoire du manuel de référence canonique de Csound .....	xxxvi
Mentions de copyright .....	xxxviii
Débuter avec Csound .....	xxxix
Les nouveautés de Csound 6.14.0 .....	xli

## Préface du manuel de Csound

Barry Vercoe, MIT Media Lab

La réalisation de musique par ordinateur nécessite la synthèse de signaux audio avec des points discrets ou échantillons représentant des formes d'onde continues. Il y a de nombreuses façons de faire ceci, chacune offrant un type de contrôle différent. La synthèse directe génère des formes d'onde en échantillonnant une fonction enregistrée représentant une simple période ; la synthèse additive génère les nombreux partiels d'un son complexe, chacun ayant sa propre enveloppe d'intensité ; la synthèse soustractive démarre avec un son complexe pour le filtrer. La synthèse non-linéaire utilise la modulation de fréquence et la distorsion non-linéaire pour donner des caractéristiques complexes à des signaux simples, tandis que l'échantillonnage et l'enregistrement d'un son naturel permettent de l'utiliser à volonté.

Comme la spécification détaillée d'un son point par point est vite ennuyeuse, le contrôle est opéré de deux manières : 1) à partir d'instruments dans un orchestre, et 2) à partir d'événements dans une partition. Un orchestre est en fait un programme d'ordinateur qui peut produire des sons, tandis qu'une partition est un ensemble de données auxquelles ce programme réagit. Qu'une durée d'attaque soit une constante fixée dans un instrument, ou une variable de chaque note dans la partition, dépend de la façon dont l'utilisateur veut la contrôler.

Les instruments d'un orchestre de Csound (voir *Syntaxe de l'orchestre*) sont définis dans une syntaxe simple qui invoque des procédures de traitement audio complexe. Une partition (voir *La partition numérique standard*) passée à cet orchestre contient des informations de hauteur et de contrôle codées dans un format numérique standard. Bien que la plupart des utilisateurs se contentent de ce format, des langages de traitement de partition de plus haut niveau sont souvent pratiques.

Les programmes constituant le système Csound ont une longue histoire de développement, qui a commencé avec le programme Music 4 écrit aux Bell Telephone Laboratories au début des années 1960 par Max Mathews. C'est là que fut conçu le concept de table d'onde ainsi qu'une grande partie de la terminologie qui a permis depuis aux chercheurs de l'informatique musicale de communiquer. D'importantes additions furent apportées à Princeton par feu Godfrey Winham dans Music 4B ; mon propre Music 360 (1968) doit beaucoup à ce travail. Avec Music 11 (1973) j'ai pris une voie différente : les deux structures distinctes des signaux de contrôle et des signaux audio sont issues de mon engagement intensif lors des années précédentes dans la conception et l'élaboration de synthétiseurs numériques. Cette division a été retenue dans Csound.

Parce qu'il est entièrement écrit en C, on peut installer facilement Csound sur n'importe quelle machine équipée de Unix ou du langage C. Au MIT il tourne sur des stations VAX/DEC sous Ultrix 4.2, sur des machines SUN sous OS 4.1, sur SGI sous 5.0, sur IBM PC sous DOS 6.2 et Windows 3.1, et sur le Macintosh d'Apple sous ThinkC 5.0. Avec ce seul langage de définition de traitement numérique du signal et des formats audio portables comme AIFF et WAV, les utilisateurs peuvent passer facilement d'une machine à l'autre.

La version de 1991 apporta le vocodeur de phase, FOF, et les types de données spectrales. 1992 vit l'arrivée des convertisseurs et des unités de contrôle MIDI, permettant de piloter Csound depuis des fichiers MIDI (midifiles) et des claviers externes. En 1994 les programmes d'analyse du son (lpc, pvoc) furent intégrés dans le module principal, permettant de lancer tous les traitements de Csound depuis un seul exécutable, et Cscore pouvait passer les partitions directement à l'orchestre pour une réalisation itérative. La version de 1995 introduisit un ensemble MIDI étendu avec linseg basé sur MIDI, les filtres de Butterworth, la synthèse granulaire, et un détecteur de hauteur amélioré, dans le domaine fréquentiel. L'addition d'outils de génération d'évènements (Cscore et MIDI) fut particulièrement importante, permettant des configurations excitation/réponse qui rendent possible la composition et l'expérimentation interactives. Il est apparu que la synthèse numérique par programme en temps réel était désormais réellement prometteuse.

## Histoire du manuel de référence canonique de Csound

La version initiale de ce manuel pour les premières versions de Csound fut démarrée au MIT par Barry L. Vercoe et y fut maintenue durant les années 1980 et le début des années 1990. Une partie du manuel provient de documents pour des programmes des années 1970 comme *Music11*. Ce manuel original fut amélioré et développé par Richard Boulanger, John ffitch, Jean Piché et Rasmus Ekman.

Ce manuel évolua vers le manuel de référence officiel de Csound que l'on trouve toujours à <http://www.lakewoodsound.com/ksound> [http://www.lakewoodsound.com/ksound/hypertext/manual.htm], pour la version 4.16 de Csound de novembre 1999, qui était maintenu par David M. Boothe.

Une version parallèle du manuel appelée le manuel de référence alternatif de Csound, fut développée par Kevin Conder en utilisant *DocBook/SGML* [http://www.docbook.org/]. Cette version devint plus tard la version canonique.

Quand le MIT plaça Csound sous license LGPL en 2003, le manuel passa sous license GFDL et fut placé sur Sourceforge avec les sources de Csound.

Durant l'hiver 2004, le manuel canonique fut converti en DocBook/XML par Steven Yi afin de permettre à plus de gens d'assurer sa compilation et sa maintenance.

Le manuel est actuellement maintenu par Andrés Cabrera avec des contributions continues de la communauté de Csound.

Le manuel est toujours un projet communautaire qui dépend des contributions des développeurs et des utilisateurs afin d'aider à affiner l'étendue et la précision de son contenu. Toutes les contributions sont les bienvenues et sont appréciées.

### Tableau 1. Autres collaborateurs

Mike Berry
Eli Breder
Michael Casey
Michael Clark
Perry Cook
Sean Costello
Richard Dobson
Mark Dolson



Dan Ellis

Tom Erbe

Bill Gardner

Michael Gogins

Matt Ingalls

Richard Karpen

Anthony Kozar

Victor Lazzarini

Allan Lee

David Macintyre

Gabriel Maldonado

Max Mathews

Hans Mikelson

Peter Neubäcker

Peter Nix

Ville Pulkki

Maurizio Umberto Puxeddu

John Ramsdell

Marc Resibois

Rob Shaw

Paris Smaragdis

Greg Sullivan

Istvan Varga

Bill Verplank

Robin Whittle

Steven Yi

François Pinot

Andrés Cabrera

Gareth Edwards

Joachim Heintz

John ffitich

Oeyvind Brandtsegg

Menno Knevel

Felipe Sateler

And many others.

Cette liste n'est en aucune façon exhaustive. On peut obtenir plus d'information par le fichier Changelog dans l'entrepôt des sources du manuel.

# Mentions de copyright

Cette version du manuel de Csound ("Le manuel canonique de Csound") est délivrée sous la GNU Free Documentation Licence [<http://www.gnu.org/licenses/fdl.txt>]. Les mentions de copyright antérieures et le crédit de leurs auteurs sont donnés ci-dessous, pour des raisons historiques.

## Mentions de copyright antérieures

Copyright © 1986, 1992 par le Massachusetts Institute of Technology. Tous droits réservés.

Développé par *Barry L. Vercoe* au Experimental Music Studio, Media Laboratory, M.I.T., Cambridge, Massachusetts, avec le support partiel de la System Development Foundation et du National Science Foundation Grant # IRI-8704665.

## Manuel

Copyright © 2003 by Kevin Conder pour les modifications apportées au manuel de référence public de Csound.

Il est permis de copier, distribuer et/ou modifier ce document selon les termes de la GNU Free Documentation License, Version 1.2 ou toute version ultérieure publiée par la Free Software Foundation ; sans aucune partie non modifiable, aucun texte de première de couverture et aucun texte de quatrième de couverture. Une copie de cette licence est disponible dans le sous-répertoire des exemples [<examples/fdl.txt>] ou à : [www.gnu.org/licenses/fdl.txt](http://www.gnu.org/licenses/fdl.txt) [<http://www.gnu.org/licenses/fdl.txt>].

La documentation du langage Csound de ce manuel est dérivée du *manuel de référence alternatif de Csound* de Kevin Conder, qui est lui-même dérivé du *manuel de référence public de Csound*.

Copyright © 2004-2005 par Michael Gogins pour les modifications faites au *manuel de référence alternatif de Csound*.

Cette mention légale provient du *manuel de référence public de Csound* : « L'édition hypertexte originale du manuel de csound du MIT fut préparée pour le World Wide Web par *Peter J. Nix* du Department of Music at the University of Leeds et *Jean Piché* de la faculté de musique de l'université de Montréal. Une édition d'impression, en format Adobe Acrobat, fut ensuite maintenue par *David M. Boothe*. Les éditeurs reconnaissent entièrement les droits des auteurs de la documentation et des programmes originaux, comme décrits ci-dessus, et demandent en conséquence que cette mention soit citée chaque fois que ce matériel est utilisé. »

La dernière url connue du manuel de référence public de Csound était <http://www.lakewood-sound.com/csound/hypertext/manual.htm>.

L'url du manuel de référence alternatif de Csound, pour les copies transparentes et les copies opaques, est <http://kevindumpscore.com/download.html#csound-manual>.

L'url du manuel de Csound et de CsoundAC est <http://sourceforge.net/projects/csound>.

Traduction française du manuel par François Pinot.

La traduction française du manuel est placée sous GNU Free Documentation License, Version 1.2 ou ultérieure, comme la version anglaise originale.

## Csound et CsoundAC

Csound est protégé par copyright de 1991 à 2008 par Barry Vercoe, John ffitch et les autres développeurs.

CsoundAC est protégé par copyright de 2001 à 2008 par Michael Gogins.

Csound et CsoundAC sont des logiciels libres ; vous pouvez les redistribuer et/ou les modifier selon les termes de la GNU Lesser General Public License tels que publiés par la Free Software Foundation ; soit la version 2.1 de la License, soit (à votre choix) n'importe quelle version ultérieure.

Csound et CsoundAC sont distribués dans l'espoir qu'il seront utiles, mais SANS AUCUNE GARANTIE ; sans même la garantie implicite de la VALEUR COMMERCIALE ou de l'ADEQUATION A UNE UTILISATION SPECIALE. Consultez la GNU Lesser General Public License pour plus de détails.

Vous devez avoir reçu une copie de la GNU Lesser General Public License en même temps que Csound et CsoundAC ; si ce n'est pas le cas, écrivez à la Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA..

# Débuter avec Csound

## Téléchargement

Si vous n'avez pas déjà installé Csound (ou si vous avez une ancienne version) téléchargez la version de Csound adaptée à votre plate-forme depuis la *page de téléchargement de Csound sur Github* [<http://csound.github.io/download.html>]. Les programmes d'installation pour Windows ont un suffixe '.exe' et ceux pour le Mac '.dmg'. Si le nom du programme d'installation se termine en '-d' cela veut dire qu'il a été construit avec la *double* précision (64-bit) qui produit une sortie de meilleure qualité que la *simple* précision (32-bit). Les versions en *simple* précision produisent une sortie plus rapide, ce qui peut être important si l'on utilise Csound en temps réel. Vous pouvez aussi télécharger les sources et les compiler, mais cela réclame plus d'expertise (voir la section *Construire Csound*).

Il est aussi utile de télécharger la version la plus récente de ce manuel, que vous trouverez également sur ce site.

## Exécution

Il y a différentes manières d'exécuter Csound. Comme Csound est un programme en ligne de commande (DOS dans la terminologie Windows), double-cliquer sur l'exécutable de Csound n'aura aucun effet. On doit appeler Csound soit depuis un terminal (ou invite DOS), soit depuis un frontal. Pour utiliser Csound en ligne de commande, vous devez ouvrir un *terminal* (une invite de commande DOS sous Windows ou un terminal sous MacOS). L'utilisation de Csound en ligne de commande pouvant sembler difficile si vous n'avez jamais utilisé de terminal, vous voudrez peut-être essayer un des frontaux, soit QuteCsound, qui est inclus dans les distributions récentes, soit un autre frontal. Un *frontal* est un programme graphique qui facilite l'exécution de Csound. La plupart des frontaux comprennent des éditeurs de texte permettant d'éditer les fichiers csound, et plusieurs d'entre eux offrent d'autres possibilités intéressantes.

Que ce soit avec un frontal ou en ligne de commande, l'exécution de Csound nécessite deux choses :

- Un fichier Csound ('.csd' ou bien un fichier '.orc' et un fichier '.sco')
- Une liste d'options de ligne de commande qui configurent l'exécution. Ils déterminent des éléments comme le nom et le format du fichier de sortie, si l'audio en temps réel et MIDI sont actifs, quelle carte son utiliser, la taille des tampons, les types de messages imprimés, etc. On peut inclure ces options dans le fichier '.csd' lui-même ; ainsi dans le cas des exemples inclus dans ce manuel, *vous ne devriez pas avoir à vous en soucier*. Les programmes frontaux ont souvent des boîtes de dialogue permettant de fixer les options de ligne de commande. On peut trouver la liste complète et très longue des options de ligne de commande *ici*, mais vous voudrez peut-être la consulter plus tard...

Consultez la section *Configuration* si vous rencontrez des problèmes avec Csound.

Cette documentation comprend de nombreux fichiers '.csd' que vous pouvez tester, et qui devraient fonctionner directement depuis la ligne de commande ou depuis n'importe quel frontal. *oscil.csd* [exemples/oscil.csd] est un exemple simple que l'on peut trouver dans le répertoire des *exemples* de cette documentation. Votre frontal devrait vous permettre de choisir le fichier, et il devrait avoir un bouton "Jouer" ou "Restituer" permettant d'entendre ce fichier. Si l'on veut manipuler ce fichier pour expérimentation, il vaut mieux utiliser la commande "Enregistrer sous..." du frontal pour copier le fichier dans un autre répertoire du disque dur, tel qu'un répertoire "partition csound" créé à cet effet.



## Note pour les utilisateurs de MacCsound

Il peut être nécessaire d'effacer toutes les lignes de la balise des options de commande afin de faire fonctionner les exemples du manuel.

Vous pouvez aussi essayer les exemples à partir de la ligne de commande en vous déplaçant dans le répertoire des exemples du manuel avec ce type de commande sous Windows (en supposant que le manuel est situé en c:\Program Files\Csound>manual\) :

```
cd "c:\Program Files\Csound>manual\examples"
```

ou quelque chose comme :

```
cd /manualdirectory/manual/examples
```

pour les terminaux Mac ou linux et en tapant ensuite :

```
csound oscil.csd
```

Les fichiers exemples étant configurés pour fonctionner en temps réel par défaut, vous devriez avoir entendu une onde sinusoïdale de 2 secondes.

## Ecrire vos propres fichiers .csd

Un fichier *.csd* ressemble à ceci (ce fichier est *oscils.csd* [exemples/oscils.csd]) :

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o oscils.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

iflg = p4
asig oscils .7, 220, 0, iflg
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 2 0
```

```
i 1 3 2 2 ;double precision
e
</CsScore>
</CsoundSynthesizer>
```

Les fichiers .csd de Csound comprennent 3 sections principales entre les balises <CsSynthesizer> et </CsSynthesizer> :

- *CsOptions* - Contient les *options de ligne de commande* spécifiques à ce fichier particulier. Ces options peuvent aussi être définies dans le fichier .csound6rc que l'on peut modifier avec un éditeur de texte, ou directement dans la *ligne de commande*. Certains frontaux offrent également des moyens de spécifier les options globales ou locales.
- *CsInstruments* - Contient les instruments ou processus disponibles dans ce fichier. Les instruments sont définis en utilisant les opcodes *instr* et *endin*. La section *CsInstruments* contient aussi l'*en-tête de l'orchestre* qui définit des choses comme le *taux d'échantillonnage*, le *nombre d'échantillons dans une période de contrôle* et le *nombre de canaux de sortie*.
- *CsScore* - Contient les "notes" à jouer et, en option, la définition de ftables. Les notes sont créées en utilisant l'*instruction i*, et les ftables sont créées en utilisant l'*instruction f*. Plusieurs autres *instructions de partition* sont disponibles.

Notez que tout ce qui suit un point-virgule (;) jusqu'à la fin de la ligne est un commentaire, et est ignoré par Csound.

Vous pouvez écrire les fichiers csd dans n'importe quel éditeur de texte pur comme notepad ou textedit. Si vous utilisez un traitement de texte (non recommandé), assurez vous de sauvegarder le fichier en texte pur (et non en texte enrichi). De nombreux *frontaux* proposent des capacités d'édition avancées avec coloration syntaxique et complétion automatique du code.

Vous pouvez trouver ici [<http://michael-gogins.com/archives/tutorial.pdf>] un tutoriel détaillé pour débiter avec Csound, écrit par Michael Gogins.

## Les nouveautés de Csound 6.14.0

### Notes de parution de Csound 6.14 (janvier 2020)

Un certain nombre de corrections de bogues et d'améliorations, mais aussi des changements potentiellement significatifs ont été ajoutés.

Pour les codeurs en direct, les macros d'orchestre sont conservées entre les appels à compilerstr. Ça ne change pas le comportement des orchestres actuellement valides mais pourra être utile pour le codage en direct.

Les périphériques MIDI incluent maintenant l'association de périphériques multiples à des canaux de numéros supérieurs. Les détails sont dans la section MIDI du manuel.

- Nouveaux opcodes :
  - randc est comme randi mais il utilise une interpolation cubique.
  - mp3out est une implémentation expérimentale d'écriture d'un fichier mp3. Il pourra être remplacé par le travail développé actuellement dans libsndfile pour traiter les fichiers MPEG.

Exemple simple :

```
<CsoundSynthesizer
```

```
<CsInstruments
ksmps = 1000
instr 1
  aa diskin "fox.wav", 1
  mp3out aa, aa, "test.mp3"
endin
</CsInstruments>
<CsScore
i1 0 3
e
</CsScore
</CsoundSynthesizer
```

La syntaxe est "mp3out aleft, aright, Sfilename" avec trois arguments facultatifs "mode" (0=stéréo, 1=stéréo entrelacé (par défaut), 3=Mono), "bitrate" 256 par défaut, et "quality" (entre 1 et 7), 2 par défaut (qualité élevée).

- *metro2* est comme *metro* mais avec l'ajout d'un rubato contrôlable.
  - *flexists* reports whether a numbered ftable exists.
  - *schedulek* est un opcode de taux-k identique à *schedule*.
  - Nouveaux opcodes de canaux basés sur des tableaux : *chngeti*, *chngetk*, *chngeta*, *chngets*, *chnseti*, *chnsetk*, *chnseta*, *chnsets*.
  - *lastcycle* identifie le dernier k-cycle d'une instance d'instrument.
  - *strstrip* supprime les espaces aux deux extrémités d'une chaîne de caractères.
  - Orchestre :
    - L'expression conditionnelle a?b:c calculait toujours b et c avant de choisir lequel retourner. Cela pouvait donner des erreurs de division par 0 ou causer des évaluations multiples d'opcodes inattendues. Elle implémente maintenant la sémantique commune du langage C.
    - Les macros de l'orchestre sont maintenant persistantes, si bien qu'elles s'appliquent à chaque compilation suivant leur définition jusqu'à ce qu'elles soient explicitement indéfinies. Ce changement a été effectué en particulier pour le codage en direct. Les orchestres valides ne devraient pas être affectés.
    - Après une erreur de syntaxe, il y avait des cas où Csound donnait une erreur de segmentation. C'est maintenant corrigé.
  - Options :
    - La nouvelle option *simple-sorted-score* crée le fichier *score.srt* dans un format plus agréable.
    - Traitement de *CsOptions* revu avec les guillemets et les espaces qui doivent être échappés.
    - En positionnant le bit 1024 dans *-m*, on supprime l'affichage des messages sur l'utilisation d'opcodes obsolètes. Cette option est elle-même obsolète.
  - Opcodes et Gens modifiés :
    - *squinewave* traite maintenant un argument facultatif de taux-a ou -k.
    - L'opcode *pindex* traite des champs chaîne de caractères ou numériques.
- 

• *sflooper* a été modifié pour éviter un plantage et donne des avertissements.

- event\_i etd schedule acceptent un p1 fractionnaire.
- Meilleurs tests dans les opcodes soundfont. De plus ils ne chargent plus plusieurs copies d'un soundfont, mais ils réutilisent celui qui est déjà chargé.
- fluidControl a un nouvel argument facultatif pour contrôler l'affichage de messages.
- bpf a une version audio maintenant.
- stsend/strecv peuvent travailler avec des taux-k différents.
- pvstrace a de nouveaux arguments facultatifs.
- lpfreson verifie le nombre de pôles.
- syncloop avait une petite erreur de frappe qui provoquait des plantages.
- bpfcs a de nouvelles versions avec tableaux.
- On peut omettre le second argument de zacl, ce qui par défaut n'efface que le canal donné.
- outvalue essayait d'utiliser une valeur de taux-k qui pouvait ne pas être valide à cet instant. C'est avant tout un petit problème d'exécution et il est maintenant supprimé.
- Les noms de canal pour les opcodes chnget et chnset peuvent maintenant être changés au taux-k si bien qu'on peut les appeler depuis une boucle.
- copya2ftab a maintenant un argument facultatif supplémentaire qui est un décalage dans la ftable dans laquelle on copie le tableau.
- Utilités :
  - lpanal vérifie maintenant que suffisamment de pôles sont requis.
- Frontaux :
  - CsoundQt : à paraître en même temps que Csound 6.14, il y a CsoundQt 0.9.7. Voir les notes de parution [https://github.com/CsoundQt/CsoundQt/blob/master/release\\_notes/Release%20notes%200.9.7.md](https://github.com/CsoundQt/CsoundQt/blob/master/release_notes/Release%20notes%200.9.7.md) [[https://github.com/CsoundQt/CsoundQt/blob/master/release\\_notes/Release%20notes%200.9.7.md](https://github.com/CsoundQt/CsoundQt/blob/master/release_notes/Release%20notes%200.9.7.md)].
- Usage général :
  - Les commentaires // en début de ligne sont maintenant acceptés dans la section CsOptions d'un fichier csd.
  - L'option --orc a été corrigée pour fonctionner sans partition : l'exécution dure indéfiniment jusqu'à ce que soit rencontrée une condition de sortie.
- Boques corrigés :
  - shiftin corrigé.
  - exitnow donne le code de retour correspondant à la documentation.
  - Boque corrigé dans beosc, où le générateur de bruit gaussien n'était pas initialisé.

- OSCraw corrigé.
- ftkloadk pouvait sélectionner un code interne incorrect causant un plantage.
- GEN01 lorsqu'il était utilisé pour lire un seul canal d'un fichier multi-canaux donnait une longueur incorrecte.
- ftgenonce avait un problème de dépassement qui pouvait causer l'écriture dans une table déjà active.
- Une situation de compétition dans les opcodes Jacko a été améliorée (issue #1182).
- syncloop avait une petite erreur de frappe qui causait des plantages.
- lowresx était incomplet et ne fonctionnait pas comme prévu : réécrit (issue #1199).
- Si outch recevait de manière incorrecte un nombre impair de paramètres, il donnait une erreur de segmentation. Cela produit maintenant un message d'erreur.
- Changements système :
  - Une nouvelle classe de greffon pour des opcodes sans sortie a été écrite. Les erreurs d'exécution et d'initialisation sont aussi retournées dans le code de retour du système en ligne de commande. La nouvelle fonction de l'API GetErrorCnt est disponible pour faire quelque chose de similaire sous d'autres variantes.
- API:
  - La fonction GetErrorCnt donne le numéro des erreurs d'exécution et ajoute les erreurs d'initialisation à la fin du rendu.
  - La fonction FTnp2Find n'affiche plus de messages si la table n'est pas trouvée, mais retourne NULL. Le comportement précédent est fourni par FTnp2Finde.
  - csoundGetInstrument() a été ajoutée.
- Changements sur les plates-formes :
  - WebAudio :
    - csoundCompile() a été ajoutée à CsoundObj, qui ajoute des args de ligne de commande ce qui permet de remplacer les valeurs de CsOptions dans les CSDs.
    - getPlayState(), addPlayStateListener() et d'autres méthodes ont été ajoutées à CsoundObj pour demander et tracer les changements d'état de l'exécution.
  - Windows :
    - stsend retravaillé pour la bibliothèque winsock.

## Notes de parution de Csound 6.13 (juillet 2019)

Peu de nouveaux opcodes, mais un nombre significatif d'opcodes ont été étendus pour utiliser les tableaux de différentes manières, élargissant les options pour les utilisateurs. Il y a eu de nombreuses corrections du code de base ainsi que des opcodes.

- Nouveaux opcodes :



- *string2array* est une variante de *fillarray* avec des données provenant d'une chaîne de caractères contenant des valeurs séparées par des espaces.
- *nstrsr* retourne le nom d'un instrument numéroté ou une chaîne de caractères vide si le numéro ne fait pas référence à un instrument nommé.
- *ntof* convertit un nom de note en fréquence aux taux -i et -k.
- *ampmidicurve* est un nouvel opcode qui fait correspondre à une vélocité MIDI un facteur de gain avec une valeur maximale de 1, modifiant le gain de sortie par un intervalle dynamique et un exposant de mise en forme.
- Orchestre :
  - La consistance entre *kr*, *sr* et *ksmps* a été revue pour le cas spécial où il y a dépassement.
  - Correction de l'usage par défaut de *Obdfs*.
  - la résolution du chemin des fichiers pour *#include* a été revue de manière plus libérale.
  - La lecture et l'écriture dans les tableaux multidimensionnels étaient erronées. C'est corrigé.
  - Meilleure vérification des tableaux de type inconnu (issue #1124).
  - Dans toutes les opérations de tableau la taille d'un tableau est déterminée à l'initialisation et aucune allocation n'a lieu pendant l'exécution.
  - L'arithmétique des tableaux respecte dorénavant *--sample-accurate*.
- Partition :
  - Les caractères *n* et *m* pouvaient être ignorés de manière erronée dans les partitions.
  - la résolution du chemin des fichiers pour *#include* a été revue de manière plus libérale.
  - Après une erreur, le traçage des fichiers et des macros lisait de manière erronée l'information pour l'orchestre plutôt que pour les partitions. C'est corrigé.
  - La fin d'une section *r* (répétition) n'était pas toujours correcte.
  - Les parties *{ }* imbriquées d'une partition pouvaient conduire à des erreurs.
  - Après une instruction *s* une nouvelle ligne était nécessaire ; ce n'est plus le cas.
  - Les boucles de partition *{ }* ont été modifiées pour permettre les macros et les expressions dans le compteur de boucle.
- Options :
  - La nouvelle option *--use-system-sr* fixe le taux d'échantillonnage à la valeur du matériel/système.
- Opcodes et Gens modifiés :
  - Un échec de l'envoi dans OSC provoque maintenant un avertissement plutôt qu'une erreur.
  - *passign* peut maintenant avoir un tableau pour cible.
  - Des versions de *bpf/bpfc* permettant la définition des points par des tableaux ont été ajoutées.

- grain peut maintenant utiliser des tables de n'importe quelle taille, pas seulement des puissances de 2.
- Le changement de couleur dans les widgets FL fonctionne (la couleur n'était pas rafraichie précédemment).
- fillarray peut lire des valeurs dans un fichier ce qui permet de dépasser la limite du nombre d'arguments.
- sumarray fonctionne maintenant aussi bien avec des tableaux audio qu'avec des valeurs scalaires.
- L'affectation d'une valeur audio à un tableau audio fonctionne maintenant.
- monitor ne fonctionnait pas dans le mode tableau.
- gendyc respecte maintenant le mode sample-accurate.
- mtof et ftom ont maintenant une version tableau.
- sc\_lag et sc\_lagud utilisent maintenant la première entrée de taux-k ou -a lorsqu'aucune valeur initiale n'est donnée.
- printarray fonctionne maintenant avec les tableaux de chaînes de caractères.
- changed2 fonctionne maintenant avec les chaînes de caractères.
- diskgrain, syncgrain et syncloop peuvent maintenant pondérer le taux d'échantillonnage.
- GEN01 lit correctement les fichiers audio bruts quand c'est nécessaire.
- ftaudio peut maintenant prendre deux arguments optionnels pour le début et la fin de la table de données à écrire dans un fichier.
- sensekey a été réécrit dans le mode 'key down'.
- loscilx peut retourner un tableau audio.
- schedule rapporte les instruments indéfinis dans tous les cas.
- event\_i accepte maintenant les numéros d'instrument étiquetés.
- printarray traite %d correctement.
- beadsynt travaille maintenant avec les tableaux i ainsi que les tableaux k comme indiqué dans le manuel.
- Utilités :
  - hetro a reçu plusieurs corrections et améliorations.
- Usage général :
  - Il y a eu un certain nombre d'améliorations dans la sémantique du multicoeur ; la plupart de celles-ci sont des corrections avec quelques gains en efficacité.
  - Il y a un nombre maximum d'arguments pour un opcode qui n'avait jamais été explicite ni contrôlé. Les tentatives d'utilisation de trop d'arguments génèrent maintenant une erreur de syntaxe.

- Bogues corrigés :
  - FLgetsnap corrigé.
  - directory corrigé pour les extensions de fichier.
  - FLsetText réutilisait une chaîne de caractères de manière incorrecte ce qui conduisait à des valeurs erronées.
  - fmb3 n'arrivait pas à initialiser le taux du lfo dans certains cas.
  - ftaudio au taux-i était complètement inopérant.
  - Après un reinit printks pouvait être ignoré ; corrigé.
  - printks corrigé pour afficher à des dates correctes.
  - tabrowlin et getrowlin pouvaient calculer une taille erronée sous certaines conditions.
- Changements système :
  - L'implantation des tables de hachage a été modifiée pour permettre de meilleures performances en charge quand le dictionnaire contient un grand nombre d'entrées.
  - Les fonctions de greffon GEN peuvent avoir une longueur nulle, mais le code doit tester cela et agir en conséquence. Cela permet les allocations différées.
  - schedule rapporte les numéros/noms d'instruments indéfinis et continue plutôt que de causer une erreur.
  - Les appels multiples aux contrôles midi en sortie sont permis.
- API:
  - find\_opcode\_new et find\_opcode\_exact sont maintenant exposés dans l'API.
  - Après un reset un indicateur de fonction de rappel de message chaîne de caractères est configuré par défaut.
  - La nouvelle fonction csoundSystemSra a été ajoutée à l'API pour lire le taux d'échantillonnage imposé par le matériel.
- Changements sur les plates-formes :
  - WebAudio : libsndfile est maintenant compilée avec le support de FLAC et d'OGG.
  - Dans l'orchestre et la partition, le traçage du chemin dans #include suppose un séparateur \.
  - Le portage sur Haiku est maintenant disponible.
  - Les entrées et sorties analogiques avec différents numéros de canaux sont permises.

## Notes de parution de Csound 6.12 (novembre 2018)

Les changements apportés dans la version 6.11 à la lecture en format raw ont été modifiés afin que `gen1` et `diskin` ignorent les formats de fichiers exprimés par un nombre positif et utilisent à la place l'en-tête du fichier, à moins que le format soit exprimé par un nombre négatif dont la valeur absolue est alors utilisée pour le format d'un fichier audio raw. Cela devrait régler la plupart des problèmes de compatibilité.

Il y a plusieurs nouveaux opcodes et améliorations d'opcodes, de nouvelles facilités dans les partitions et plusieurs corrections de bogues.

- Nouveaux opcodes :

- *fluidInfo*  
retrouve l'information de programme dans un soundfont chargé.

- Le nouvel opcode

*ftaudio*  
écrit une ftable dans un fichier audio ; il existe des versions au taux-i et déclenchée au taux-k et la version de taux-k supporte l'écriture synchrone ou asynchrone.

- Une version de

*OSClisten*  
qui écrit les données dans un tableau de taux-k.

- *OSCcount*  
retourne le nombre restant de messages OSC entrants.

- *faustplay*  
et

*faustdsp*  
sont de nouveaux opcodes séparant l'instanciation du TNS faust et son exécution.

- *OSCbundle*  
envoie une collection de messages OSC similaires sous forme d'un paquet unique pour plus d'efficacité.

- *beosc*  
et

*beadsynt*  
sont un oscillateur et un banc d'oscillateurs à bande améliorée.

*bpfcos*  
sert pour des fonctions à points charnière avec interpolation. De même

*lincos*  
ajoute une interpolation par cosinus à

*linlin*  
.

- *printarray*  
et

*reshapearray*  
agissent sur les tableaux.

- *trim*  
et  
  
*trim\_i*  
ajustent la taille d'un tableau unidimensionnel soit en l'augmentant soit en la diminuant, en préservant les données et/ou en complétant avec des zéros.
- Orchestre :
  - Nouvelle option du préprocesseur `#includestr`. Comme `#include` avec en plus développement de macro de la chaîne de caractères délimitées par doubles apostrophes.
  - Utilisation des notes liées dans `subinstr` corrigée.
  - Les appels de macro imbriqués plus de 10 fois provoquaient un plantage ; maintenant l'imbrication n'est plus limitée.
  - Un message d'erreur d'exécution comprend maintenant (habituellement) un numéro de ligne et une trace de fichier/macro.
  - Les affectations multiples telles que `ka, kb = 1, 2` sont à nouveau supportées par le parseur ; cette possibilité avait été annulée par erreur.
  - L'utilisation problématique de `i()` directement avec un élément de tableau est maintenant indiquée comme une erreur.
  - Si un nom de fichier inclu contient un `"/`, alors toute inclusion imbriquée est relative à ce répertoire. (Issue n°973).
  - Redéfinition des UDOs avec aucun argument en sortie corrigée.
  - Un bogue très obscur dans les sous-instrument, qui a été présent pendant presque un an, a été corrigé.
- Partition :
  - Nouvelle option du préprocesseur `#includestr`. Comme `#include` avec en plus développement de macro de la chaîne de caractères délimitées par doubles apostrophes.
  - Opcode de partition `'d'` pour exécution en temps réel. C'était l'issue n°966.
  - Bogue dans l'opération `np` corrigé.
  - L'utilisation de la syntaxe `[]` dans une partition pouvait entraîner une perte de précision pour les nombres supérieurs à 1 million ; l'ancienne version a été restaurée.
  - Les appels de macro imbriqués plus de 10 fois provoquaient un plantage ; maintenant l'imbrication n'est plus limitée.
  - Les formes pour fins de sections retardées (`e 5` ou `s 5`) fonctionnent maintenant avec des retards fractionnaires ; auparavant seule la partie entière était lue.
  - Si un nom de fichier inclu contient un `"/`, alors toute inclusion imbriquée est relative à ce répertoire. (Issue n°973).
- Opcodes et Gens modifiés :
  - Argument facultatif ajouté à `fom` pour arrondir la réponse à un entier.

- flooper2 et syncgrain etc permettent maintenant le rééchantillonnage.
- chnclear peut prendre maintenant une liste de canaux à effacer au lieu d'un seul.
- printf et printf\_i now sont maintenant conformes au manuel : tous les arguments au-delà de format et de trigger sont facultatifs.
- prints et printks peuvent recevoir des arguments chaîne de caractères affichés avec %s.
- GEN2 peut prendre maintenant une taille nulle, qui est interprétée comme la taille suffisante pour le nombre de valeurs fournies.
- faustcompile comprend maintenant un nouveau paramètre facultatif pour permettre l'exécution en mode bloqué. Par défaut (comme précédemment) il est en mode non-bloqué.
- fillarray peut être exécuté au taux-k si n'importe lequel de ses arguments/valeurs est au taux-k.
- slicearray pour les tableaux autres que ceux de taux-i ne s'exécute qu'au taux-k.
- pvstrace a maintenant l'option de retourner aussi un tableau contenant les numéros de bin.
- Utilités :
  - Une erreur de codage dans mixer a été corrigée. L'opcode était très abîmé.
- Bogues corrigés :
  - Une erreur de typographie dans p5glove signifiait que la commande pour lire l'état du bouton comme un bitmap ne retournait que l'état du bouton A.
  - diskinn vers tableau corrigé et également l'utilisation avec de petites valeurs de ksmps.
  - loscil échouait parfois à traiter l'argument ibas ; le traitement est correct maintenant.
  - madsr pouvait provoquer un dépassement sur un compteur interne lorsqu'un p3 négatif était donné.
  - La correspondance entre threads et lua\_States a été corrigée (issue n°959).
  - Le temps de calcul dans flooper2, flooper et syncgrain a été corrigé.
  - Rééchantillonnage et hauteur corrigés dans pvstanal.
  - De rares cas de saturation de tampon ont été corrigés dans faust.
  - pvsftw avait un contrôle de format de TFR incorrect qui produisait un message d'erreur de format ; c'est corrigé.
  - Si ksmps valait 1, l'opcode linenr au taux-a ne fonctionnait pas ; c'est corrigé.
  - Opcode window corrigé.
  - Le contrôle de compatibilité des sous-types de f-valeurs dans plusieurs opcodes pvs était défectueux, provoquant de faux messages d'erreur.
  - cosseg était inopérant pour plus d'un segment. Corrigé maintenant.

- L'opcode monitor ne fonctionnait pas correctement lors de l'utilisation de plusieurs processus ; c'est corrigé.
- Dans l'opcode linenr si la chute commençait avant la fin de l'attaque la valeur de sortie sautait à la valeur finale et ensuite diminuait, provoquant une discontinuité. C'est corrigé (n°1048).
- Changements système :
  - Les différentes options -zN rapportent maintenant le nombre d'opcodes pour la demande et diffèrent ainsi sur les opcodes obsolètes et les opcodes polymorphes.
  - La recompilation des instruments nommés a été totalement repensée pour éviter les erreurs et les fuites de mémoire.
  - L'allocation des noms d'instrument à des numéros internes a été réécrite et devrait rester stable lors des remplacements en codage en temps réel.
  - L'affichage du numéro alloué à un instrument nommé s'effectue maintenant normalement, pas seulement en mode débogage.
  - Si la version 0.29 de liblo est disponible, csound peut être construit pour l'utiliser (avec l'option de compilation LIBLO29) et cela corrige certains bogues liés à une utilisation lourde/complexes de OS-Clisten. Malheureusement l'ancienne version 0.28 version est toujours distribuée sur certaines versions de linux.
  - Le compilateur d'orchestre a plusieurs nouvelles optimisations évitant des affectations inutiles et réalisant un peu plus d'optimisation d'expressions.
- API:
  - csound->ReadScore a été modifié pour avoir le même comportement qu'avec un fichier csd ou sco. Avant cela, il pouvait donner ou pas une partition infinie de manière inattendue.
- Changements sur les plates-formes :
  - Bela : autorise les E/S analogiques avec différents numéros de canaux.

## Notes de parution de Csound 6.11 (mai 2018)

Il y a eu une importante réorganisation en interne qui ne devrait pas affecter la plupart des utilisateurs. Certains composants sont maintenant gérés indépendamment et seront éventuellement installables via un nouveau gestionnaire de paquets. L'option temps réel est considérée comme stable et son étiquette "experimental" a été supprimée. Un support spécial pour la plate-forme Bela a été introduit. L'intégration des opérations arithmétiques impliquant des tableaux-a a bien avancé.



### Note

Noter que les changements dans GEN01 et dans disk2 peuvent ne pas être rétrocompatibles si une valeur non nulle est donnée pour le format.

- Nouveaux opcodes :
  - *loscilphs*, *loscil3phs*, sont comme *loscil* mais ils retournent la phase en plus de la sortie audio.
  - Plus d'arithmétique entre les tableaux de taux-a et les valeurs de taux-a ; cela complète l'arithmétique lorsqu'un ou plusieurs arguments sont des tableaux audio.

- *balance2* est comme *balance* mais il pondère la sortie au taux-a plutôt qu'au taux-k.
- Partition :
  - Les caractères suivant un \ dans une chaîne de caractères de la partition sont échappés. Ainsi \n représente un retour à la ligne plutôt que deux caractères. Les caractères a, b, f, n, r, t et v peuvent être échappés. Les autres caractères suivant un \ ignorent la barre inversée.
- Opcodes et Gens modifiés :
  - *print*, *printk2* prennent maintenant un argument supplémentaire facultatif qui, s'il est non nul, provoque l'affichage du nom de la variable de taux-k en plus de sa valeur.
  - *getrow*, *setrow*, *getcol* et *setcol* peuvent désormais agir sur les tableaux de taux-i.
  - *diskin2* était incorrectement décrit dans le manual quant au paramètre *iformat*. Maintenant si *iformat* est nul le fichier est supposé avoir un en-tête audio ; s'il est compris entre 1 et 10 (au lieu de 0 et 9 comme précédemment), le fichier est ouvert comme fichier brut avec le format d'échantillon spécifié. CECI PEUT ETRE INCOMPATIBLE. Pour la plupart des utilisateurs la valeur zéro sera correcte.
  - *GEN01* utilise maintenant le format 0 pour lire le type du fichier dans son en-tête ; tout autre valeur indique un fichier brut. CECI PEUT ETRE INCOMPATIBLE. Pour la plupart des utilisateurs la valeur zéro sera correcte.
  - *GEN01* n'était pas correctement documenté quant à l'argument du format. Il y a 9 formats brut alors que le manuel n'en renseignait que 6 auparavant.
  - Un petit changement dans *slicearray* devrait permettre son utilisation sous forme fonctionnelle dans la plupart des cas.
  - La famille d'opcodes *tb0* à *tb15* et ses opcodes d'initialisation *tb0\_init* à *tb15\_init* sont obsolètes car les fonctions à arguments multiples sont permises.
  - Le filtre *mode* n'accepte plus une fréquence dans la région instable.
  - Dans *scanu* et *xscanu* la valeur de *kpos* était supposée être dans l'intervalle [0, 1]. C'est maintenant consolidé en traitant toutes les valeurs négatives comme 0 et toutes les valeurs supérieures à 1 comme 1.
- Utilités :
  - *src\_conv* est maintenant disponible sous Windows et sous MacOS.
- Frontaux :
  - *Belacsound* : un nouveau frontal spécifique pour la plate-forme Bela est introduit dans cette version.
- Bogues corrigés :
  - *linen* a été retravaillé pour permettre les longues durées et les recouvrements.
  - La réinitialisation de *Csound* provoquait un plantage si les opcodes Lua étaient utilisés ; c'est corrigé.
  - La famille d'opcodes *poscil* donnait des résultats incorrects s'ils étaient utilisés dans des orchestres multicoeurs ET qu'un autre instrument changeait la table-f. C'est maintenant corrigé.



- L'utilisation de out avec un tableau audio ne testait pas si la dimension du tableau était supérieure au nombre de canaux, ce qui provoquait un plantage. La dimension est maintenant testée et tronquée si elle est trop grande, avec un message d'avertissement.
- Un bogue dans les versions stéréo de loscil provoquait une distortion ; c'est corrigé.
- Erreur d'intervalle dans phs corrigée.
- gen49 effaçait la mémoire si le fichier n'était pas trouvé ; corrigé.
- Le chargement de greffons LADSPA, lorsqu'il reposait sur des chemins de recherche, était défectueux et abrégait les noms ; c'est corrigé.
- Changements système :
  - OPCODE6DIR{64} peut contenir maintenant une liste de répertoires séparés par des double-points.
  - La distinction quelque peu curieuse entre les temps d'exécution de taux-k et de taux-a a été supprimée du débit, si bien que seuls les threads 1, 2 et 3 sont utilisés, la sortie de type-s n'est pas utilisée et certaines entrées de type-x ont été changées pour un polymorphisme explicite. Ceci ne compte que pour les auteurs d'opcodes car les codes s et x et les threads 4, 5, 6 et 7 seront bientôt supprimés.
- Traductions :
  - La traduction française est complète comme d'habitude.
- API :
  - Pas de changement dans l'API.
- Changements sur les plates-formes :
  - WebAudio : à partir de cette distribution, le portage de Csound vers les plates-formes web a été unifié dans un seul paquet nommé désormais "WebAudio Csound". Cela a provoqué beaucoup de changements en interne, l'interface AudioWorklet étant employée partout où elle est supportée, avec l'alternative de ScriptProcessorNode dans les autres cas. Csound est aussi disponible sous formes de Nodes indépendants, en plus de l'API originale CsoundObj. Le frontal csound.js, conçu à l'origine pour PNaCl Csound, a été aussi porté dans WebAudio Csound, suite à l'obsolescence de ce portage de Csound.

## Notes de parution de Csound 6.10 (décembre 2017)

C'est principalement une distribution de correction de bogues, incluant un bogue important introduit récemment dans loscil. Il y a aussi de nouveaux opcodes et des améliorations d'opcode, ainsi qu'un GEN (53) longtemps ignoré et plusieurs petites améliorations internes. Les changements internes ont supprimé un certain nombre de fuites de mémoire.

- Nouveaux opcodes :
  - *midout\_i* semblable à *midout*, mais travaille au taux-i.
  - *chngetks* et *chnsetks* -- version de *chnget* et *chnset* pour canaux de chaînes de caractères qui ne sont actifs que pendant l'exécution.
  - *squinewave*, un oscillateur d'onde carrée-pulsation-dent de scie-sinus à bande limitée et variation de forme avec synchro forcée..

- Nouveaux Gens et macros :
  - *GEN53* qui était présent dans le code depuis plusieurs années mais pas documenté) est maintenant reconnu. Il crée une réponse impulsionnelle à phase linéaire ou à phase minimale à partir d'une table contenant une réponse fréquentielle ou une réponse impulsionnelle.
- Orchestre :
  - L'usage incorrect du `if..then..` de `taux-k` dans l'instrument 0 est maintenant traité au `taux-i`.
  - L'usage incorrect des opérations de `taux-k` dans l'instrument 0 n'est plus traité comme une erreur mais comme un avertissement.
  - Les balises mises en commentaire dans un fichier `csd` étaient parfois interprétées, ce qui produisait un orchestre tronqué. C'est corrigé.
  - On peut assigner des tableaux de `taux-i` et de `taux-k` vers le `taux-k` et le `taux-i` ; auparavant, les `taux` devaient se correspondre.
  - L'utilisation de `!` comme opération booléenne (signifiant la négation) est maintenant supportée, alors qu'avant le parseur l'acceptait mais sans l'utiliser.
  - L'optimisation par résolution des constantes est maintenant largement utilisée dans les calculs arithmétiques.
  - Si on essaie d'utiliser une macro non définie, une erreur de syntaxe est maintenant générée.
  - L'oubli de `"` (ou de tout autre caractère terminal) dans `#include` est notifié et le `#include` est ignoré.
- Partition :
  - Les balises mises en commentaire dans un fichier `csd` étaient parfois interprétées, ce qui produisait une partition tronquée. C'est corrigé.
  - La forme d'évaluation `[..]` peut maintenant être imbriquée.
  - L'option d'extraction (`-x` sur la ligne de commande) fonctionne dorénavant.
  - L'utilisation de l'opcode de partition `x` pouvait générer de faux messages d'erreur qui ont été supprimés.
  - Après l'appel d'une macro non définie le reste de la ligne est ignoré.
  - Une paire de bogues dans les sections répétées (opcode `r`) ont été supprimés.
  - L'oubli de `"` (ou de tout autre caractère terminal) dans `#include` est notifié et le `#include` est ignoré.
- Options :
  - L'option `--tempo` (ou `-t`) peut désormais être fractionnaire ; elle était auparavant limitée aux entiers.
  - nouvelle option : `--udp-console=address:port` redirige la console vers une adresse:port distante.
  - nouvelle option : `--udp-mirror-console=address:port` recopie la console sur une adresse:port distante.
  - nouvelle option : `--udp-echo` affiche en écho les messages envoyés au serveur UDP.
- Opcodes et Gens modifiés :

- loscil/loscil3 acceptent un incrément en virgule flottante.
- OSCraw ferme le socket après usage.
- fout peut maintenant générer le format ogg, et également accepter -l ce qui signifie l'utilisation du même format que celui de l'option -o.
- bitwise et l'opcode (&) au taux-a ont été corrigés pour le mode de précision à l'échantillon (--sample-accurate).
- slicearray a un argument supplémentaire facultatif pour fixer le pas d'incrément de la partie.
- chnset peut maintenant avoir des variables noms de canal.
- Les tableaux de taux-a peuvent être additionnés, soustraits, multipliés et mis à l'échelle. C'est le début d'une arithmétique de taux-a sur les tableaux.
- dssiinit a été amélioré par la suppression de certains plantages.
- partials a été amélioré pour supprimer un problème de limite.
- vco2ift a été corrigé pour l'utilisation d'une table existante.
- Frontaux :
  - Emscripten: compilé comme WebAssembly (s'exécute dans les principaux navigateurs). L'API est maintenant plus conforme aux autres APIs pour HTML5.
  - CsoundQT est construit maintenant depuis la branche principale pour une stabilité accrue.
- Bogues corrigés :
  - Le caractère facultatif du dernier argument dans sc\_phasor est maintenant traité correctement.
  - Le blocage dans dconv a été corrigé.
  - looptseg ne plante plus si on ne lui présente pas assez d'arguments.
  - schedule etc fonctionnent maintenant correctement avec des chaînes de caractères entre guillemets à l'intérieur de chaînes entre { { } }.
  - Le problème de la gestion d'interruption du frontal CLI est corrigé.
  - outs2 était défectueux (il écrivait toujours sur le canal 1 comme outs1).
  - Diverses erreurs dans le système DSSI/ladspa ont été corrigées.
  - vbap était défectueux dans tous les cas excepté à 4 haut-parleurs. C'est corrigé.
  - L'évaluation d'un code d'orchestre de Csound en direct pouvait produire des erreurs étranges et difficiles à diagnostiquer (par exemple des plantages, des divisions par zéro, NaNs). Cela était dû à un bogue lors de la fusion dans la zone des constantes globales de nouvelles constantes trouvées dans le code.
- Changements système :

- Le serveur UDP accepte maintenant de nouvelles commandes qui sont préfixées par un opcode. Cela comprend le support des événements (&) et des partitions (\$) ; l'affectation des canaux de contrôle (@) ; l'affectation des canaux de chaîne de caractères (%) ; la réception des valeurs d'un canal de contrôle via UDP (:@) et du contenu d'un canal de chaîne de caractères (:%).
- Traductions :
  - La traduction française est complète comme d'habitude.
  - La traduction italienne des messages s'est grandement améliorée ; près de la moitié des messages d'erreur et d'avertissement sont maintenant traduits.
  - Quelques progrès ont été faits pour la traduction allemande.
- API :
  - CompileCsdText retourne maintenant une valeur indiquant le succès ou l'échec.
  - Huit nouvelles versions asynchrones de fonctions de l'API sont maintenant disponibles : csoundCompileTreeAsync(), csoundCompileOrcAsync(), csoundReadScoreAsync(), csoundInputMessageAsync(), csoundScoreEventAsync(), csoundScoreEventAbsoluteAsync(), csoundTableCopyOutAsync() et csoundTableCopyInAsync().
  - Pour l'utilisation en serveur, trois nouvelles fonctions de l'API : csoundUDPServerStart, csoundUDPServerStatus et csoundUDPServerClose.
- Changements sur les plates-formes :
  - Windows
    - Désormais compilé avec Microsoft Visual Studio 2017 ou ultérieur.
    - Intégration continue pour Windows avec AppVeyor.
  - GNU/Linux
    - Le back-end ALSA MIDI de GNU/Linux ignore maintenant certains faux codes d'erreur ENOENT.

## Notes de parution de Csound 6.09 (mai 2017)

De nouveaux opcodes et plusieurs corrections et améliorations.

Il y a comme d'habitude un certain nombre de changements internes, y compris plusieurs corrections de fuites de mémoire et un code plus robuste.

- Nouveau opcodes :
  - *select* -- comparaison échantillon par échantillon sélectionnant la sortie audio.
  - *midiar* -- génération d'arpèges basés sur les notes MIDI actuellement tenues.
  - *hilbert2* -- une implémentation de la transformée de Hilbert basée sur la TFD.
  - Opcodes de liaison avec Ableton -- pour synchroniser le tempo et la pulsation sur des réseaux locaux.
  - *pvstrace* -- ne retient que les N bins les plus forts.

- Plusieurs nouveaux opcodes/fonctions unaires pour les tableaux numériques de taux-k et de taux-i : `ceil`, `floor`, `round`, `int`, `frac`, `powoftwo`, `abs`, `log2`, `log10`, `log`, `exp`, `sqrt`, `cos`, `sin`, `tan`, `acos`, `asin`, `atan`, `sinh`, `cosh`, `tanh`, `cbrt`, `limit1`.
  - Plusieurs nouveaux opcodes/fonctions binaires pour les tableaux numériques de taux-k et de taux-i : `atan2`, `pow`, `hypot`, `fmod`, `fmax`, `fmin`.
  - *limit* -- limitation numérique dans un intervalle donné (pour les tableaux).
  - *tvconv* -- un opcode de convolution variant dans le temps (filtre RIF).
  - *liveconv* -- convolution partitionnée avec réponse impulsionnelle rechargeable dynamiquement.
  - *bpf*, *xyscale*, *ntom*, *mtom* -- (de SuperCollider ?).
  - *OSCsend* -- maintenant implémenté directement au moyen des sockets système. L'ancienne version utilisant `liblo` a été gardée sous le nom de `OSCsend_lo`.
  - *OSCraw* -- pour écouter tous les messages OSC sur un port donné.
  - *sorta* et *sortd* -- trient les éléments d'un tableau.
  - *dot* -- calcule le produit scalaire de deux tableaux.
  - Filtres sans retard -- *zdf\_1pole\_mode*, *zdf\_2pole\_mode*, *zdf\_ladder*, *zdf\_1pole* et *zdf\_2pole*, *diode\_ladder*, *K35\_hpf* et *K35\_lpf*.
  - *product* -- prend un tableau numérique (de taux-k ou i) et calcule son produit.
  - Générateurs unitaires de supercollider -- *sc\_phasor*, *sc\_lag*, *sc\_lagud*, *sc\_trig*.
  - Orchestre :
    - L'inclusion d'un répertoire de fichiers UDO n'échoue plus s'il y a plus de vingt entrées.
    - `kr`, `sr` et `ksmps` pouvaient être inconsistants dans un cas particulier ; c'est corrigé.
    - Les noms de macro sont mieux contrôlés et appariés avec des crochets.
    - Des valeurs octales sous la forme `\000` peuvent apparaître dans des chaînes de caractères.
    - (à partir de la 6.09.1) Dans un UDO, l'opcode `out*` fonctionne maintenant alors qu'auparavant il ne fonctionnait qu'épisodiquement.
  - Partition :
    - Report du numéro de ligne amélioré dans l'opcode `r` et lors de l'implémentation de macro.
    - opcodes `m` et `n` corrigés.
    - développement de [...] corrigé et amélioré.
    - Amélioration des chaînes de caractères dans les partitions.
    - On peut utiliser le caractère `)` dans un argument de macro s'il est échappé par `\`.
- 
- L'utilisation des caractères `e` ou `s` pouvaient produire des erreurs ; c'est corrigé.

- Les noms de macro sont mieux contrôlés et appariés avec des crochets.
- p2 et p3 ont maintenant une meilleure précision qui n'est plus limitée à six positions décimales.
- Nouvel opcode d pour interrompre des notes infinies (denote) ; agit comme i avec un p1 négatif.
- Les instruments nommés peuvent être interrompus avec i si un - suit le ".
- (à partir de la 6.09.1) Lorsqu'une section d'opcode-r se terminait par un opcode-e, elle s'arrêtait trop tôt.
- Options :
  - Le module midi jack peut maintenant rapporter les périphériques disponibles sous --midi-devices.
  - (à partir de la 6.09.1) La définition de smacros et de omacros sur la ligne de commande n'a lieu qu'une fois.
  - (à partir de la 6.09.1) La définition de smacros sur la ligne de commande fonctionne maintenant.
- Opcodes et Gens modifiés :
  - Traitement des arguments chaîne de caractères amélioré dans ftgentmp.
  - L'opcode hdf5read lit maintenant des ensembles de données complets lorsque le nom de l'ensemble de données est suffixé avec un astérisque.
  - L'utilisation de longueurs différentes d'une puissance de deux est maintenant acceptable là où elle était auparavant inconsistante.
  - ampmidid peut tenir compte de 0dbfs en option.
  - dust et dust2 au taux-k sont maintenant conformes au manuel (NOTE : ceci est une modification incompatible).
  - Dans prints le format %% affiche maintenant un %.
  - OSClisten peut maintenant être utilisé sans sortie de données.
  - GEN18 a été corrigé pour écrire dans l'intervalle requis.
  - sockrev peut maintenant lire des chaînes de caractères.
  - Le système vbap peut autoriser dans certains cas un nombre arbitraire de haut-parleurs via des tableaux (chantier en cours).
  - Le serveur websocket ne peut accepter qu'un seul protocole en sortie, limitant ainsi le type en entrée à un seul argument.
  - L'opcode sum additionne aussi les éléments d'un tableau.
  - pvs2tab et tab2pvs surchargés peuvent maintenant créer et utiliser des tableaux de magnitude et de phase séparés.
- Utilités :
  - dnoise a été corrigé.

- Frontaux :
  - Retrait de l'éditeur HTML5 de Csound qui ne fonctionne plus.
  - Emscripten : Emscripten Csound (asm.js) nécessite maintenant les sources CsoundObj.js et FileList.js séparément de libcsound.js. Ceci pour permettre l'utilisation de la même API JS avec les backends asm.js ou wasm
  - CsoundQT : CsoundQt 0.9.4 est annoncé : [https://github.com/CsoundQt/CsoundQt/blob/develop/release\\_notes/Release notes 0.9.4.md](https://github.com/CsoundQt/CsoundQt/blob/develop/release_notes/Release%20notes%200.9.4.md) [[https://github.com/CsoundQt/CsoundQt/blob/develop/release\\_notes/Release notes 0.9.4.md](https://github.com/CsoundQt/CsoundQt/blob/develop/release_notes/Release%20notes%200.9.4.md)].
  - L'installer Windows inclue PythonQt avec CsoundQt.
- Bogues corrigés :
  - pwd fonctionne sur OSX.
  - Erreur de fencepost dans sensLine corrigée.
  - OSCsend corrigé pour mettre en cache le nom de l'hôte.
  - Bogue dans les opcodes push/pop corrigé (ces opcodes sont maintenant un greffon qui est obsolète).
  - Bogue dans sprintf supprimé.
  - Bogue dans soundin supprimé.
  - losci/losci3 corrigés dans le cas des tables de grande taille.
  - inrg n'a plus fonctionné pendant un certain temps.
  - Les lois de panoramique channelmask de partikkel présentaient une erreur d'indexation qui a été corrigée.
  - Le module audio de jack autorise maintenant des nombres de canaux d'entrée indépendants de ceux des canaux de sortie.
  - Bogue corrigé dans la copie de chaînes de caractères.
  - Il y avait un bogue dans hdf5read lorsque deux opcodes hdf5read étaient placés en série dans un instrument. Les noms d'argument de l'instance du deuxième opcode étaient incorrects à cause du changement direct de la dernière chaîne de caractères du premier lors de la lecture d'un ensemble de données complet.
  - Fuites de mémoire corrigées dans certains opcodes de greffon.
- Changements système :
  - soundin utilise maintenant le code de disk2.
  - La famille d'opcodes out a été retravaillée pour réduire le coût de l'entrelacement et pour tenir compte correctement de la valeur de nchnls.
  - (à partir de la 6.09.1) Un plantage sous Linux i386 a été supprimé en relation avec le mode serveur.

- Nouvelles facilités présentes seulement dans l'en-tête `csound_threaded.hpp`, supprimant l'utilisation de `csPerfThread.cpp` dans certains projets.
- Ajout de la fonction `GetA4`.
- Nouveau cadre pour le développement d'opcode dans un greffon en C++ avec l'utilisation des allocateurs de Csound.
- Ajout de la fonction `StrDup`.
- Dépendances à boost supprimées des interfaces de Csound et de CsoundAC.
- (à partir de la 6.09.1) Deux nouvelles fonctions de l'API, `csoundSetSpinSample` et `csoundClearSpin`.
- Changements sur les plates-formes :
  - iOS
    - Correction + animation de iPad portrait SplitView, redimensionnement de info popover, correction du bouton stop dans Soundfile Pitch Shifter.
    - Mises à jour de l'API de Csound-iOS ; exemples nettoyés, améliorés/étendus et réorganisés. Manuel révisé, étendu et mis à jour. L'API et les exemples supportent iOS 10 et Xcode 8.
  - Android
    - Entrée et sortie multi-canaux permises.
  - Windows
    - Bibliothèque d'importation `csound64.lib` ajoutée à l'installateur Windows.

## Notes de parution de Csound 6.08 (novembre 2016)

Comme d'habitude, il y a un certain nombre de corrections et d'améliorations des opcodes, mais les changements les plus importants apparaissent dans les structures du langage. Tout d'abord, dans le langage de partition, la gestion des macros et le prétraitement ont été entièrement rénovés, pour s'aligner sur les mêmes fonctionnalités de l'orchestre. L'analyse de l'orchestre a reçu un certain nombre de corrections décrites ci-dessous.

Un changement majeur et pas complètement compatible a été opéré dans la lecture et l'écriture des éléments de tableau. Maintenant, le taux de l'indice détermine souvent le taux de traitement ; voir l'entrée *Orchestre* ci-dessous. Ceci simplifie pas mal de code et semble répondre aux attentes des utilisateurs ; le code précédent comportait plusieurs anomalies.

Il y a également de nouveaux opcodes, des corrections internes de fuites de mémoire et un code plus robuste.

- Nouveaux opcodes :
  - *dct*: Transformée en cosinus discrète d'un tableau d'échantillons (DCT-II)
  - *getftargs*: copie des arguments d'un gen dans une variable-S.
  - *mfb*: implante un banc de filtres mel-fréquence pour un tableau de magnitudes en entrée.



- Nouveaux GEN et macros :
  - *quadbezier*: génération de courbes de Bézier dans une table de fonction.
- Orchestre :
  - Le caractère  $\neg$  est maintenant traité correctement comme variante de  $\sim$  pour la négation binaire.
  - Correction d'un bogue d'analyse lexicale qui pouvait altérer les chaînes de caractères.
  - Absence du symbole de nouvelle ligne garantie dans l'analyse lexicale.
  - Petite amélioration du report des numéros de ligne.
  - Vérification améliorée de la syntaxe des macros.
  - Analyse améliorée de l'affectation des étiquettes.
  - Prise en compte des erreurs de crochets non appairés dans la spécification des arguments pour les UDO.
  - Vérification qu'un fichier `#included` n'est pas un répertoire.
  - Meilleure gestion des appels de macros imbriqués sur plusieurs niveaux.
  - Pendant des années Csound fixait la fréquence du la médian à 440 Hz. Maintenant, cette fréquence peut être fixée dans l'en-tête par la nouvelle variable-r A4 et lue avec cette même variable.
  - L'exposant des valeurs en virgule flottante peut être écrit e ou E.
  - La sémantique de l'accès aux tableaux a été clarifiée :
    - `i[i]` => lecture à l'initialisation et pendant l'exécution, écriture seulement à l'initialisation.
    - `i[k]` => lecture pendant l'exécution, l'écriture génère une erreur d'exécution.
    - `k[i]`, `k[k]` => lecture et écriture pendant l'exécution.
    - `a[i]`, `a[k]` => lecture et écriture pendant l'exécution.
    - autres cas (`S[]`, `f[]`) => lecture et écriture en fonction du type de l'indice (i,k).  
En particulier, `i(k[i])` ne fonctionne toujours pas, mais le nouvel opérateur `i(k[],i)` prend en charge ce cas.
  - la validation de `xout` n'échoue plus lorsqu'une constante est fournie.
- Partition :
  - Nouveau code pour gérer les macros et les autres commandes de prétraitement. Aligne cette gestion sur le code de l'orchestre.
  - Nouvelle instruction de partition C introduite pour basculer du report inactif (C 0) au report actif (C 1, par défaut).
- Options :
  - Le tempo peut être fixé maintenant par une valeur en virgule flottante (auparavant, on ne pouvait utiliser que des valeurs entières).

- Nouvelle option --version, affiche l'information de version et termine le programme.
- Opcodes et Gens modifiés :
  - Problèmes dans centroid corrigés.
  - Meilleur traitement de l'arrondi dans printks.
  - OSC étendu pour inclure la multidiffusion.
  - Les opcodes faust ont été mis à jour.
  - oscil1 et oscili peuvent prendre une durée négative.
  - la documentation de l'opcode fout a été clarifiée.
  - La durée de relâchement de mxadsr a été corrigée.
  - L'opcode centroid accepte également un tableau en entrée.
  - Les opcodes ptable sont maintenant identiques à ceux de la famille table.
  - ftgen prend maintenant un tableau d'entrée en option.
  - subinstr peut maintenant avoir des arguments chaîne de caractères.
  - Le format i() peut maintenant aussi travailler sur des tableaux de taux-k, le premier argument étant un tableau, suivi par des indices.
- Utilités :
  - pvlook affiche maintenant explicitement le nom de la fenêtre d'analyse.
- Frontaux :
  - HTML5
    - csound.node : implanté pour linux, correction mineure de l'API.
    - pnacl : méthode compileCsdText ajoutée à l'objet csound.
- Usage général :
  - Test de la validité des noms de macro amélioré.
  - #undef corrigé.
- Bogues corrigés :
  - Corrections dans l'utilisation du format pour prints.
  - jitter2 retravaillé pour être plus conforme au manuel.
  - oscbank a reçu plusieurs corrections et fonctionne maintenant selon sa description.
  - bformdec1 avec tableaux et type 4 corrigé.
  - Bogues dans pvsceps corrigés.

- Dans différents opcodes print, des caractères aléatoires étaient parfois ajoutés -- corrigé.
- l'affectation de variables avec --sample-accurate pouvait donner des résultats inattendus ; il semble que ce soit corrigé maintenant.
- La forme carrée du profil de padsynth a été corrigée et l'opcode est moins dépendant du niveau d'avertissements.
- gen31 corrigé.
- gen41 corrigé.
- Bogue dans sensekey corrigé.
- Plusieurs problèmes dans centroid corrigés.
- Changements système :
  - Nouvelle analyse lexicale et prétraitement de la partition.
  - Les fins de ligne dans le style MAC refonctionnent.
  - Les messages d'information système (taux d'échantillonnage, etc) sont maintenant dirigés sur stdout.
  - rtjack retravaillé pour la gestion des noms et des nombres.
  - L'affichage de la version comprend maintenant l'information de commit afin que les développeurs sachent quels patches ont été appliqués.
- API :
  - La version de l'API est maintenant 4.0.
  - Les gens nommés sont maintenant supportés.
  - fterror est maintenant dans l'API.
  - Les fonctions SetOutput et GetOutputFormat ont été corrigées.
  - De nombreuses fonctions de l'API utilisent maintenant const si c'est approprié.
  - Les messages peuvent maintenant être redirigés depuis l'API vers stdout en utilisant l'attribut CSOUNDMSG\_STDOUT.
  - Nouvelles interfaces Lisp CFFI et FFI testées avec Steel Bank Common Lisp (architecture CPU 64 bit), exécutées dans un thread séparé.
  - ctsound.py, une nouvelle interface FFI pour Python avait été introduite dans la version 6.07. C'est maintenant l'interface recommandée pour Python, csnd6.py étant obsolète.
- Changements sur les plates-formes :
  - Android.
    - les entrées et les sorties multicanaux sont permises.
  - Windows.

- La bibliothèque d'importation `csound64.lib` a été ajoutée à l'installateur.

## Notes de parution de Csound 6.07 (mars 2016)

De nombreux bogues corrigés, dont certains importants, des éléments internes sont inclus, ainsi que de nouvelles fonctionnalités et extensions. Comme d'habitude, il y a aussi des améliorations du code.

- Nouveaux opcodes :
  - *compress2*: comme *compress* mais faisant un usage plus normal des dB (0.0 à pleine échelle).
  - (Expérimental, seulement dans le code source), nouveaux opcodes cuda : *cudasynth2* et *cudanal2*.
  - *directory* : lit un répertoire et restitue une liste des noms de fichier dans un tableau de chaînes de caractères.
  - *ftsamplbank* : pour charger une bibliothèque d'échantillons depuis un répertoire.
  - *mvclpf1*, *mvclpf2*, *mvclpf3*, *mvclpf4*, *mvchpf* : simulation de filtres de Moog contrôlés en fréquence d'après Fons Andriaensen.
  - *S()*, convertit un nombre de taux-k ou de taux-i en une chaîne de caractères.
  - Opcode *cepsinv* pour calculer le cepstre inverse d'un tableau.
  - *moogladder2* est plus rapide et moins précis que *moogladder*.
  - L'opcode *paulstretch* est une mise en oeuvre légère de l'algorithme d'étirement temporel PaulStretch par Nasca Octavian Paul. Il est idéal pour allonger la durée d'un signal dans de grandes proportions.
  - *mp3scal* implémente un traitement vocoder à verrouillage de phase à partir de fichiers mp3 sur disque, avec rééchantillonnage si nécessaire.
  - *filesca* implémente un traitement vocoder à verrouillage de phase à partir de fichiers sur disque, avec rééchantillonnage si nécessaire.
- Orchestre :
  - La construction booléenne `? .. : ..` peut maintenant avoir un résultat de type chaîne de caractères, alors qu'auparavant elle n'autorisait que les nombres.
  - Le numéro de ligne rapporté lors de la détection d'une erreur à la fin d'une ligne est maintenant correct. Également des améliorations dans les numéros de ligne pour une syntaxe complexe.
  - Boucle *while* améliorée/corrigée.
  - Lecture des commentaires meilleure et consistante.
  - Lignes de continuation mieux gérées, spécialement par le respect des numéros de ligne.
  - opcode : *S()* syntaxe maintenant permise.
- Partition :
  - La liste des points de tempo dans l'opcode de partition *t* est maintenant arbitrairement longue.
  - Une incohérence dans les opcodes *r* et *{* a été corrigée.

- Options :
    - L'option `-z` supprime maintenant les opcodes obsolètes, sauf si on lui donne comme argument 2 ou 3.
    - La nouvelle option `--fftlb` détermine la bibliothèque de TFR réelle à utiliser en interne (FFTLIB = 0, PFFFT = 1, vDSP = 2).
  - Opcodes et Gens modifiés :
    - Dans les opcodes *OSC* on peut maintenant envoyer et recevoir des tableaux, des tables et de l'audio.
    - Meilleur diagnostic si *diskin2* échoue.
    - *rezzy* détecte maintenant une instabilité du filtre et le modifie vers une version stable.
    - *adsr* a été réécrit pour lever une erreur si la longueur des segments dépasse p3.
    - L'utilisation de *diskin* vers un tableau redimensionne le tableau de sortie si nécessaire.
    - *chnget* teste maintenant un changement de nom de canal comme de type de données.
    - *interp* peut prendre un argument facultatif supplémentaire pour donner une valeur initiale.
    - *osciliks* utilise une lecture de table plus permissive.
    - L'opcode *in* peut lire en mono ou en stéréo, évitant ainsi la distinction *in/ins*.
    - *sensekey* a été réécrit pour fournir de meilleurs diagnostics et corriger un bogue mineur.
    - Correction d'un cas de définition de macro avec des arguments.
    - *sockrecv* now works at a and k rate.
    - *GEN49* travaille maintenant depuis un appel de *ftgen*.
    - *GEN34* est plus tolérant dans les tables qu'il accepte.
    - *chnget* autorise maintenant le changement des noms de canal pendant l'exécution.
    - *iceps* renommé *cepsinv* pour éviter les conflits.
    - Support du mp3 amélioré sur plusieurs petits détails.
    - Petite correction pour autoriser le rechargement d'un greffon *lapsda* par l'hôte.
  - Frontaux :
    - *csdebugger* :
      - Quelques problèmes de mémoire réglés.
    - HTML5 :
      - CsoundQT a maintenant ses propres notes à [https://github.com/CsoundQt/CsoundQt/blob/develop/release\\_notes/Release%20Notes%200.9.2.1.md](https://github.com/CsoundQt/CsoundQt/blob/develop/release_notes/Release%20Notes%200.9.2.1.md) [[https://github.com/CsoundQt/CsoundQt/blob/develop/release\\_notes/Release%20Notes%200.9.2.1.md](https://github.com/CsoundQt/CsoundQt/blob/develop/release_notes/Release%20Notes%200.9.2.1.md)].
- 

- Usage général :

- Les options multicoeur ont été améliorées.
- Lors du remplacement d'instruments, la nouvelle version hérite de *maxalloc* et des options actives.
- Le code multicoeur fonctionne maintenant avec des instruments MIDI.
- Les opérations MIDI sont maintenant disponibles via le module jack *rtmidi* (*-+rtmidi=jack*).
- Bogues corrigés :
  - Correction de *trigseq*.
  - Une erreur majeure dans *rezzy* a été corrigée.
  - *p()* a été corrigé pour un grand nombres de p-arguments.
  - *p()* fonctionne maintenant avec les évènements MIDI.
  - Le générateur de nombres aléatoire de 31 bit pouvait produire une boucle très courte s'il était initialisé avec zéro ; corrigé.
  - Les macros dans les fichiers .orc fonctionnent maintenant.
  - Un bogue très ancien dans l'affichage de graphes a été corrigé.
  - Corrections dans les opcodes d'enveloppe *linen*, *expsegr*, *linsegr*, *cossegr*, *transegr*, *envlpx*, comprenant de nouveaux avertissements.
  - De nombreuses corrections dans les opcodes de chaîne de caractères (*strsub*, *strcpy* et d'autres).
  - Bogues corrigés dans les formats d'impression (*sprintf*, *printf*, *prints*).
  - Il manquait une initialisation à *pvsMOOTH*.
  - L'initialisation des tableaux est maintenant plus robuste.
  - Bogue corrigé dans *copya2ftab*.
  - Correction dans *cuDAPvsanal* (EXPERIMENTAL).
  - *partikkel* corrigé pour le placement de grain de sous-échantillon.
  - Les opcodes nécessitant un nombre impair d'arguments sont maintenant testés proprement.
  - *pvsWrite* tient maintenant compte correctement de 0dbfs.
  - *GEN34* autorise maintenant les tables sources de longueur différente d'une puissance de deux.
  - *strcat* réparé.
  - *nstance* réparé.
  - Petits défauts corrigés dans le module *rtauhal*.
  - L'installateur pour Windows sur architecture CPU 64 bit fixe maintenant correctement les variables d'environnement au niveau système comme il se doit.

- transeg au taux-k avec itype différent de zéro avait un retard initial d'un cycle.
- Fonction log pour les tableaux corrigée.
- Pondération des GENs nommés corrigée.
- Une erreur de terminaison dans la lecture des fichiers ATS était supposée corrigée ; elle générait de fausses erreurs indiquant trop tôt la fin de fichier.
- Erreur d'initialisation dans pvbandp corrigée.
- Allocation de chaîne de caractères dans readfi corrigée.
- Changements système :
  - Utilisation intensive de fichiers en mémoire. Les fichiers .orc et .sco sont toujours traités ainsi, de même que .csound6rc. Cela simplifie grandement le fonctionnement interne.
  - Les longjmps imbriqués ont été corrigés, ce qui affectait certaines utilisations de l'API.
  - Si un fichier .csd qui n'existe pas est présenté, Csound ne plante plus.
  - La valeur kcounter est maintenant une valeur consistante non-signée sur 64 bit, ce qui permet de longues exécutions.
- API :
  - csoundCompileCsdText : nouvelle fonction pour compiler un CSD depuis un texte sous forme de chaîne de caractères.
  - L'appel de fonction csound->GetKcounter(csound) retourne maintenant un entier non-signé sur 64 bit. Auparavant, il retournait un entier long ce qui n'était pas clair.
  - ctsound.py est un nouveau fichier de liaison entre le langage Python et l'API de Csound. C'est un fichier en pure Python qui utilise le module FFI (Foreign Function Interface) ctypes. Il ne dépend pas de Swig, et il fonctionne aussi bien avec Python2 qu'avec Python3. Il utilise numpy pour ses structures de données, numpy était le paquet fondamental du calcul scientifique avec Python.
- Changements sur les plates-formes :
  - Windows.
    - L'installateur de Csound pour Windows contient maintenant tous les exécutables construits pour une architecture CPU 64 bit.
    - L'installateur pour Windows sur architecture CPU 64 bit fournit maintenant un NW.js fonctionnel. Il exécute toutes les fonctionnalités de Csound ainsi que celles d'HTML5 dans un runtime semblable à un navigateur avec le langage JavaScript, et il comprend un éditeur Csound implémenté en HTML. L'éditeur exécute des pièces sous forme de fichiers .csd incluant une balise <html> ou sous forme de fichiers .html appelant l'objet csound, et il fournit des fonctions de débogage de JavaScript.
    - Un nouveau système simplifié se trouve dans le répertoire csound/mingw64 pour construire Csound à l'aide de la suite d'outils MSYS2/mingw64.
    - Le lien avec pthreads et d'autres bibliothèques est maintenant statique.
    - Support limité des caractères non ASCII dans les noms de fichier.

- L'installateur pour Windows fournit maintenant Csound et tous ses composants, interfaces, greffons, utilitaires et frontaux pour une architecture de CPU sur 64 bit, compilés en utilisant la suite MSYS2/Mingw64-w64. CsoundQt et csound.node sont construits avec MSVC 2013.
- OSX.
  - Installateur corrigé pour les noms de liens pour `_csnd6` et `_CsoundAC`.
- GNU/Linux.
  - L'opcode *date* est plus précis.

## Notes de parution de Csound 6.06 (septembre 2015)

De nombreux bogues, certains assez importants, ont été corrigés et de nouvelles facilités et extensions ont été ajoutées.

- Nouveau opcodes :
  - `getseed` lit l'état du générateur PRN à l'opposé de l'opcode `seed`.
  - `tabifd` — Distribution de Fréquence Instantanée, analyse en phase et magnitude.
  - `websocket` — Lecture et écriture de signaux et de tableaux au moyen d'une connexion websocket.
  - `framebuffer` — Lit des signaux audio dans des tableaux unidimensionnels de taux-k et vice-versa avec une taille de tampon spécifiée.
  - `olabuffer` — additionne des trames audio tuilées dans des tableaux de taux-k et retourne un signal audio.
- Orchestre :
  - Les étiquettes sont permises dans l'instrument 0.
- Partition :
  - La taille maximale des chaînes de caractères dans les partitions a été augmentée à 1024.
- Opcodes et Gens modifiés :
  - La version tableau de `diskin2` utilise la taille du tableau pour donner le nombre de canaux dans le format de fichier brut.
  - Le paramètre `kpitch` de `diskin2` vaut 1 par défaut pour les utilisations simples.
  - La ftable de vibrato dans `wgflute` et dans `wgcar` est facultative, étant définie par défaut sur une onde sinus.
  - `schedule` accepte maintenant les arguments chaîne de caractères.
  - `urandom` est maintenant disponible sur les plates-formes OSX.
  - GEN18 avait un problème de délimitations ; réécriture importante.
  - Dans la famille des opcodes `poscil` il est possible de passer la phase d'initialisation.



- svfilter peut maintenant passer l'initialisation.
- Lors de l'ouverture d'un fichier d'entrée, nchncls\_i est utilisé à la place de nchnls. C'est un changement qui aurait du suivre l'existence de nchnls\_i.
- Le module rtjack rapporte maintenant le taux d'échantillonnage.
- Les opcodes rfft, rifft, fft, fftinv, r2c et c2r ont maintenant des versions de taux-i.
- Nouveau paramètre facultatif de seuil dans les opcodes tradsyn, sinsyn et resyn.
- Nouvelle option de seuil pour partiels.
- Utilités :
  - L'extraction a été corrigée.
  - src\_conv a été amélioré et intégré dans les options -U.
  - Corrections dans atsa et heti.
- Frontaux :
  - pnacl :
    - Support du taux d'échantillonnage fixe de 48000 Hz.
  - csound~ :
    - Changements dans le système de processus légers. L'initialisation du canal de chaînes de caractères a été corrigée. D'autres bogues ont également été corrigés.
- Emscripten :
  - L'objet Javascript Csound peut maintenant recevoir des données de l'opcode outvalue.
- HTML5 :
  - Intégration d'HTML, de JavaScript et d'autres éléments d'HTML5 dans Csound, soit en insérant une page web comme élément <html> dans un fichier CSD pour CsoundQt ou Csound pour Android, soit en intégrant Csound dans le contexte JavaScript d'un navigateur web autonome (Emscripten, PNaCl) ou embarqué (csound.node).
- Utilisation générale :
  - Les macros malformées dans un orchestre sont maintenant interceptées.
- Bogues corrigés :
  - L'utilisation de variables d'environnement dans le style de Windows pour INCDIR, etc, fonctionnent maintenant avec les numéros de périphériques.
  - Opcode vibrato corrigé.
  - Les clics en cas d'échantillonnage de précision en temps réels ont été corrigés.
  - La copie de chaînes de caractères est maintenant correcte ; il y avait parfois des erreurs mémoire.

- Boque dans pvstanal a été corrigé.
- Erreur d'arrondi dans cpspch corrigée.
- Plus de plantage lors de la recompilation d'un instrument nommé.
- Correction du bogue d'interpolation dans tablexkt.
- Correction dans plltrack lorsque ksmps vaut 1.
- Changements système :
  - Le message d'erreur du greffon STK est maintenant un avertissement.
- API :
  - Correction de la redéfinition des opcodes et des UDOs.
- Changements sur les plates-formes :
  - OSX.
    - Le lien sur csnd6.jar a été installé au bon endroit.
    - Les problèmes de liaison avec Java JNI sont résolus.
    - Le nom du lien pour libpng dans libfltk\_image a été corrigé.

## Notes de parution de Csound 6.05 (Avril 2015)

Comme toujours il y a de nouvelles facilités et de nombreuses corrections de bogues. Une grande partie de cette livraison consiste en la suppression de nombreuses fuites et de saturation de mémoire. Naturellement ces changements ne sont pas très visibles à part une empreinte mémoire moindre. Noter que nous traitons les bogues et les demandes d'amélioration via le system de questions de github, et celles-ci ont grandement affecté cette parution.

- Opcodes :
  - L'opcode sndload est maintenant obsolète.
- Nouveaux Gens et Macros :
  - L'algorithme padsynth de Paul Octavian Nasca a été implanté sous la forme d'un gen.
- Partition :
  - Le bogue du calcul de la position d'une chaîne de caractères lors du traitement des lignes d'une partition a été corrigé [fixes #443]
- Options :
  - Une option de copyright au format court est disponible avec un nombre fixe de licences connues (CC, etc)
  - Nouvelle option de ligne de commande pour rapporter les périphériques MIDI dans un format simple.

- Nouvelle option de ligne de commande pour fixer ksmps.
- Opcodes et Gens modifiés :
  - *adsynt* gère mieux les changements d'amplitude.
  - *sfont* détecte mieux les défauts.
  - Meilleure vérification des fréquences hors-limites dans les modèles physiques.
  - *fgenonce* et d'autres acceptent les paramètres chaînes de caractères.
  - *gausstrig* a été retravaillé et étendu avec de nouvelles caractéristiques.
  - La fonction *p()* ne donne plus d'avertissement de dépassement de pcnt.
  - Correction de *midirecv*
  - Le nettoyage de fin d'utilisation d'*OSCsend* a été amélioré.
  - *fillarray* est limité à des tableaux à 1 ou 2 dimensions ; en fait, auparavant, il échouait silencieusement pour les dimensions supérieures ou égales à 3.
  - *oscbnk* fonctionne maintenant lorsque l'égaliseur est utilisé.
  - *mp3in* fonctionne maintenant avec des fichiers mono et stéréo.
  - *flooper* & *flooper2* acceptent maintenant les tables stéréo.
  - La phase de relâchement d'*expsegr* a été corrigée.
  - Les ftables créées avec un grand nombre d'arguments pouvaient saturer la mémoire ; c'est corrigé maintenant.
  - L'exécution de *plltrack* a été améliorée.
  - L'initialisation des tableaux a été clarifiée et testée.
  - *gen23* a été corrigé pour stopper une boucle infinie.
  - *alwayson* démarre maintenant depuis le décalage de la partition.
  - *invalute* teste dorénavant la taille de chaîne de caractères en sortie et réalloue de la mémoire si elle est inférieure à la taille par défaut (fixée à 256 octets pour une compatibilité ascendante)
- Utilitaires :
  - L'utilitaire *srconv* a été amélioré mais il ne fonctionne pas très bien, avec des passages bruiteux dans une sortie correcte. Nous recommandons l'utilisation du code Secret Rabbit de Erik de Castro Lopo (*libsamplerate*) qui produit une conversion du taux d'échantillonnage de grande qualité. *srconv* sera supprimé prochainement pour un remplacement possible par un utilitaire basé sur SRC.
- Frontaux :
  - *pnacl*
    - Ajout d'une interface permettant l'utilisation du système d'entrée MIDI de Csound.

- L'entrée audio est conforme à la dernière spécification de l'API Pepper.
- Bogues corrigés :
  - Bogues corrigés dans fastabi,oscktp, phasorbnk, adsr, xadsr, hrtfer.
  - Bogues corrigés dans harmon. harmon2, harmon3 et harmon4.
  - Csound pouvait planter après une erreur d'analyse ; ce cas est maintenant supprimé.
- Changements système :
  - Il y a de nouveaux tests sur la concordance des types de xin/xout avec leur définition dans un UDO.
  - Jack a maintenant un timeout.
- Changements internes :
  - Plusieurs défauts trouvés par coverity ont été corrigés ou le code changé. Cela devrait renforcer Csound dans les cas limites.
  - Des changements du parser simplifient l'allocation des variables temporaires, avec de nouvelles optimisations.
  - Le code du rendu multi-thread a été amélioré et stabilisé quant à la redéfinition des instruments.
- Changements sur les plates-formes :
  - iOS.
    - La fonction de rappel audio a été corrigée pour fonctionner correctement avec la sortie lumineuse et la TV Apple.
  - Android.
    - Nouveau mode d'E/S audio expérimental : csoundPerformKsmpls() est appelée depuis la fonction de rappel de l'E/S OpenSL. On peut activer ce mode facultativement en passant une valeur "false" au second paramètre du nouveau constructeur de CsoundObj (bool isAsync). Le constructeur par défaut et celui à un paramètre fixent cette valeur à "true" (ce qui permet une compatibilité ascendante avec le code existant).
    - The OSC opcodes are included in distribution.
  - App Android
    - Il y a de nouvelles boîtes de dialogue pour l'ouverture et la sauvegarde de fichier qui permettent d'accéder à une carte SD, si l'appareil en est équipé, en plus du stockage interne.
    - Un nouveau bouton "Save as..." permettant de sauvegarder le csd comme un nouveau fichier avec un nouveau nom.
    - De nombreux exemples de l'archive Android sont maintenant construits dans l'application et peuvent être lancés depuis le menu.
    - Inclue maintenant l'opcode exciter.
- OSX.

- L'installation place maintenant csladspa.so au lieu de csladspa.dylib sur le disque.
- Linux.
- La version Linux est maintenant construite sans les threads FLTK. Cela supprime les gels du système et est conforme aux autres distributions.

## Notes de parution de Csound 6.04 (novembre 2014)

Cette nouvelle version comprend de nombreuses extensions et corrections ; de nouveaux opcodes et un nombre important de réorganisations internes. Il y a un nouveau frontal et les versions iOS et Android présentent de nombreuses améliorations.

Nous suivons les bogues et les demandes d'amélioration via le système de signalement de github. Des propositions pour la parution suivante sont déjà faites, mais la quantité de changements impose une parution dès maintenant.

- Nouveaux opcodes :
  - *pinker* génère du bruit rose de grande qualité.
  - L'opcode *power ^* fonctionne maintenant avec des arguments de type tableau.
  - L'opcode *exciter* est calqué sur le greffon calf.
  - L'opcode *vactrol* simule un suiveur d'enveloppe analogique.
  - Famille d'opcodes hdf5 pour gérer le format de fichiers hdf5.
  - (expérimental non documenté) l'opcode buchla modélise le filtre lowgate de Buchla.
- De nouveaux opcodes de taux-k agissant sur les tableaux :
  - transformées: rfft, rifft, fft, fftinv
  - produit complexe : complxprod
  - conversion polaire - rectangulaire : rect2pol, pol2rect, mags, phs
  - réel - complexe: r2c, c2r
  - fenêtrage : window
  - cepstrum : pvscpes, iceps, ceps
  - accès colonne / ligne : getrow, getcol, setrow, setcol
  - copie données de taux-a - tableau de taux-k : shiftin, shiftout
  - déroulement de phase : unwrap
- Nouveaux Gens et Macros :
  - Numéros de ligne corrigés dans les instructions d'instr.
  - Nouvelle opération de contrôle, la boucle while.

- Un bogue ancien sur les macros qui utilisaient le même nom pour un argument a été corrigé.
- La redéfinition d'un instrument lors d'un appel unique au compilateur est signalé comme une erreur.
- Le saut de l'en-tête ID3 des fichiers mp3 est maintenant correctement implémenté.
- Les erreurs dues à la non-définition de l'endroit où se trouvent les fichiers d'onde bruts de STK ont été supprimées.
- Un bogue qui empêchait les UDO de lire une chaîne de caractères dans un p-champ a été corrigé.
- Opcodes et Gens modifiés :
  - Les opcodes de manipulation de pile sont obsolètes.
  - lenarray étendu pour la gestion des tableaux multi-dimensionnels.
  - ftgenonce accepte correctement les arguments chaîne de caractères.
  - max et min ont maintenant une version de taux-i.
  - gen23 amélioré au vu des commentaires et des problèmes rapportés.
  - Dans OSCsend le port est maintenant une valeur de taux-k.
  - socksend travaille maintenant au taux-k.
  - Plusieurs opcodes générateurs d'enveloppe sont maintenant corrects en mode de précision à l'échantillon.
  - La compilation faust est maintenant protégée par un verrou.
  - mp3 a été modifié pour permettre son utilisation avec reinit.
  - Dans les opcodes de contrôle à distance, la nom du réseau peut être fixé via la variable d'environnement CS\_NETWORK. Vaut par défaut en0 (OSX) ou eth0.
- Frontaux :
  - icsound : le nouveau frontal icsound est maintenant prêt pour tout usage. icsound est une interface Python pour un usage interactif avec le notebook ipython.
  - csdebugger : plusieurs changements et améliorations ont été faits, comme un parcours pas à pas des instruments actifs, une meilleure utilisation des numéros de ligne.
- Usage général :
  - Le module Jack ne stoppe plus Csound en cas d'échec de la connexion automatique.
- Bogues corrigés :
  - atsinnoi corrigé.
  - ftsavek corrigé.
  - sprintf corrigé.

- Correction de gen27, en particulier avec de nombreux arguments, ainsi que de nombreuses erreurs dans les arguments de partition en grand nombre.
- Les opcodes Physem (guiro cabasa, sekere) corrigés pour qu'un second appel fonctionne.
- flooper corrigé dans le mode 2.
- OSCsend, plusieurs corrections.
- UDO corrigé pour le cas d'un ksmps local égal à 1.
- D'autres changements et corrections au code dssi.
- xscanu et scanu corrigés.
- temposcal et mincer corrigés.
- Le plantage dans ftload a été corrigé.
- Changements système :
  - En mode serveur la sortie se fait maintenant proprement.
  - Corrections dans le module rtalsa.
  - Corrections dans le module temps réel de pulseaudio.
  - Correction pour enlever les entrées de fluidEngine d'une instance de Csound (empêche le plantage dans moduleDestroy).
  - Les opcodes appelés en tant que fonction qui retournaient des tableaux ne synthétisaient pas correctement les arguments en types tableau parce qu'ils ne convertissaient pas le spécificateur de l'argument au format interne.
  - Correction du plantage durant la phase d'initialisation de notes liées à cause du code sauté par le goto.
  - Correction de l'initialisation erronée des p-champs lorsqu'ils étaient moins nombreux que ce que l'instrument attendait (décalage d'un p-champ).
- Changements internes :
  - Ajout de l'identification de type à l'exécution pour les variables d'instrument ; on utilise plus XINCODE/XOUTCO.
  - Correction de l'analyse des nombres négatifs pour la taille de malloc et gestion améliorée des nombres négatifs.
  - L'écriture dans circularBuffer est maintenant atomique.
  - Plusieurs fuites de mémoire et des codes potentiellement dangereux ont été corrigés.
  - L'inférence de type a été considérablement retravaillée ainsi que certaines zones du parseur.
- API :
  - Nouvelle fonction de l'API pour retrouver les paramètres GEN utilisés lors de la création d'une table.
- Changement sur les plates-formes :

- iOS :
  - API refactorisée pour avoir des noms de méthode et d'abstraction plus clairs (par exemple Csound-Binding au lieu de CsoundValueCacheable).
  - Nettoyage du code obsolète.
  - Le code a été significativement retravaillé.
- Android :
  - API refactorisée pour avoir des noms de méthode et d'abstraction plus clairs (par exemple Csound-Binding au lieu de CsoundValueCacheable).
  - Changements pour permettre d'avoir HTML 5 avec JavaScript, et WebGL si possible, dans l'app Android Csound6.
  - Changement d'orientation de l'écran dans l'app Csound6 sans avoir à redémarrer l'app.
  - Stockage local (utile pour sauvegarder et restaurer les valeurs des widgets, etc.)
- Windows :
  - Correction de l'arithmétique des pointeurs qui causait un plantage sous Windows.
  - pyexec a été modifié pour utiliser les fonctions d'ouverture de fichier de Python afin d'éviter un plantage sous Windows.
- OSX :
  - CsoundAC est compilé maintenant.
- Linux :
  - Bogues de threadlocks corrigés sous linux.

## Notes de parution de Csound 6.03 (mai 2014)

Cette nouvelle version corrige de nombreux bogues, comprenant la résolution de tickets sur SourceForge et sur GitHub. Elle apporte également des changements internes pour améliorer l'exécution.

- Nouveaux opcodes :
  - `prinks2`  
affiche une nouvelle valeur chaque fois qu'une variable de contrôle change avec une syntaxe de formatage à la `printf()`
  - `mp3sr`, `mp3bitrate` et `mp3nchnls` pour obtenir de l'information sur les fichiers mp3
  - EXPERIMENTAL : les opcodes CUDA pour la convolution partitionnée, la convolution directe et le vocodeur de phase : opcode OpenCL pour la synthèse additive
  - `compilecsd`  
pour compiler des instruments à partir d'un fichier CSD standard
- Orchestre :



- i() attendait comme argument une variable pas une expression. C'est maintenant la norme. (bug #90)
- Partition :
  - Nouvelle instruction de partition y fixant la graine de random (pour ~) durant la lecture
- Options :
  - Il y avait un bogue dans CsOptions ; le dernier argument n'était pas lu (issue #296)
  - Les options de ligne de commande expression-opt et no-expression-opt n'ayant plus d'usage dans Csound6, un avertissement est affiché
- Opcodes et Gens modifiés :
  - Il est possible de spécifier pour la sortie ogg une qualité VBR (taux de bloc variable)
  - Le code de *dsi4cs* a été entièrement retravaillé pour éviter de potentielles fautes de mémoire
  - Plusieurs opérations de tableaux sont désormais disponibles au taux-i ainsi qu'au taux-k
  - *fillarray* fonctionne avec les tableaux de chaînes de caractères
  - L'affichage des TFR (via *dispftr*) a été amélioré avec des options de mise à l'échelle et de zoom
  - Les opcodes de graphe de fluence fonctionnent maintenant avec des signaux tableaux de taux-a
  - Dans le code alsa TR, le taux d'échantillonnage provient du périphérique
  - Le système d'opcodes Faust a été mis à jour par rapport à la dernière API de Faust
- Utilitaires :
  - Bogue corrigé dans lpanal
- csound~ :
  - OSX - correctif pour l'exécution avec une architecture à cpu 32 bit
  - Windows - csound~ est maintenant disponible sous Windows
- Emscripten :
  - Intégré désormais dans la base de code
- Utilisation générale :
  - --displays active maintenant le mode graphique comme prévu
  - Nouvelle option de ligne de commande --get-system-sr pour obtenir le taux d'échantillonnage de la machine
  - Nouvelle option de ligne de commande --devices[=in|out] qui donne une liste des périphériques audio disponibles puis termine le programme
- Bogues corrigés :
  - Lorsqu'une table était remplacée sa taille ne changeait pas

- Plusieurs bogues dans `--sample-accurate` ont été repérés et corrigés. Cela concerne en particulier les opcodes *out*, *outn* et *line*
- Plusieurs bogues dans *grain3* ont été corrigés
- Un bogue de *str\_chanel* pouvait provoquer un plantage
- Petit défaut dans *rtjack*
- Une erreur dans l'opcode *resize* a été corrigée
- Un bogue aléatoire a été corrigé dans *atsa*
- Problème de pause dans *rtauhal* corrigé
- Plusieurs bogues/maladresses corrigés dans *GEN23*
- Test des bornes des tableaux corrigé
- Les canaux de chaînes de caractères n'étaient pas correctement réglés pour les chaînes de taille variable
- L'allocation mémoire pour le formatage de chaîne dans *printfsk* a été corrigée, supprimant ainsi le troncage de chaîne
- *strcat* protégé contre la surcapacité
- une erreur dans la compilation des tableaux a été corrigée (problème #293)
- Correction de *GetPvsChannel* contre un plantage
- Changements système :
  - L'opcode *turnoff* vérifie maintenant que l'instrument affecté est actif
  - *lenarray* accepte n'importe quel type de tableau
  - la manière d'arrondir un numéro de table a été modifiée pour un comportement plus prévisible
  - on peut avoir une nouvelle section dans un fichier *csd* nommée *<CsFile...>* qui est semblable à *csFileB* mais avec un texte non encodé
  - La compilation des *UDO* utilise maintenant le système de types. Les *UDOs* peuvent ainsi utiliser n'importe quel type de tableau
  - Analyse de l'orchestre plus rapide grâce à de meilleurs algorithmes
- Changements internes :
  - Tout le système a été testé avec l'outil d'analyse statique Coverity qui a identifié plusieurs problèmes (mineurs pour la plupart). Ceux-ci ont été traités et vérifiés. En particulier, une meilleure utilisation de l'impression et de la copie de chaîne devrait permettre d'éviter les dépassements de capacité
  - Le système de types et de variables a été entièrement réécrit ; cela a permis un meilleur support des tableaux et des *UDOs*
  - L'alignement des variables est maintenant correct dans tous les cas

- La copie de tableau utilise maintenant le système de types pour copier les valeurs ; cela règle les problèmes de copie des tableaux de chaînes de caractères, de f-sigs, etc
- Csound est réinitialisé lorsqu'il s'arrête restant ainsi dans un état propre ; ce n'était pas le cas lors d'une erreur de compilation si bien que le démarrage suivant ce faisait avec un moteur de Csound invalide (problème #305)
- API :
  - Tous les opcodes, etc, utilisent maintenant les opérations d'allocation de mémoire de l'API, ce qui permet de remplacer complètement l'allocateur de mémoire
  - *csoundCompileCsd* a été ajouté à l'API et associé avec le nouvel opcode *compilecsd*
  - *csoundGetStringChannel* est protégé contre les chaînes courtes ou nulles et un test de la longueur de chaîne a été ajouté
  - Dans plusieurs fonctions de l'API `char*` a été remplacé par `const char*` conformément à l'usage  
Le moteur d'exécution a maintenant des capacités de débogage permettant l'interruption de l'exécution et l'inspection de l'état du moteur et des variables d'instrument. Les fonctions suivantes sont disponibles en incluant l'en-tête `csdebug.h` :

```
void csoundDebuggerInit (CSOUND *csound);
void csoundDebuggerClean (CSOUND *csound);
void csoundSetInstrumentBreakpoint (CSOUND *csound, MYFLT instr, int skip);
void csoundRemoveInstrumentBreakpoint (CSOUND *csound, MYFLT instr);
void csoundClearBreakpoints (CSOUND *csound);
void csoundSetBreakpointCallback (CSOUND *csound, breakpoint_cb_t bkpt_cb, void *userdata);
void csoundDebugContinue (CSOUND *csound);
void csoundDebugStop (CSOUND *csound);
debug_instr_t *csoundDebugGetInstrInstances(CSOUND *csound);
void csoundDebugFreeInstrInstances(CSOUND *csound, debug_instr_t *instr);
debug_variable_t *csoundDebugGetVariables(CSOUND *csound, debug_instr_t *instr);
void csoundDebugFreeVariables(CSOUND *csound, debug_variable_t *varHead);
```

- Windows:
  - Soundfonts dans Windows avait un problème d'alignement interne qui a été corrigé

## Notes de parution de Csound 6.02

Cette nouvelle version comporte de nombreuses corrections de bogues (y compris la résolution générale de tous les tickets sur SourceForge). Elle introduit aussi quelques nouvelles facilités telles qu'un serveur, du code pour exécuter Csound dans un navigateur et une large généralisation des opcodes de filtre pour les doter de paramètres modifiables au taux audio.

- Nouveaux opcodes :
  - L'opcode *nstance* produit une nouvelle instance d'un instrument et mémorise le descripteur de cette instance.
  - L'opcode *turnoff* nouvelle variante pour arrêter une instance d'un instrument donné.
  - *strfromurl* pour remplir une chaîne de caractères à partir d'une URL.
- Orchestre :

- Si la construction de Csound le permet, une chaîne intégrée avec `#include` peut être une URL ou un fichier.
- Il est à nouveau permis d'insérer un espace entre le nom d'une fonction et la parenthèse ouvrante pour toutes les fonctions valables dans Csound5 (mais pas en général).
- La commande Csound peut démarrer avec un CSD vide en mode démon (`--daemon`): ne quitte pas si le CSD/orchestre n'est pas donné, est vide ou ne compile pas).
- Partition :
  - Si la construction de Csound le permet, une chaîne intégrée avec `#include` peut être une URL ou un fichier.
- Opcodes et Gens modifiés :
  - Plusieurs filtres ont été modifiés pour accepter des paramètres de taux-k ou de taux-a. En particulier, les filtres suivants sont concernés :

areson	atonex
butterworth filters	fofilter
lowres	lowresx
lpf18	mode
moogladder	moogvcf
reson	resonr
resonx	resonz
statevar	tonex

- Le nombre maximum de presets dans `sfont` a été augmenté à 16384.
- *cpsmidinn* est désormais plus précis.
- Le comportement de *max\_k* suit maintenant la documentation. Dans certains cas les résultats étaient bizarres.
- Dans l'opcode *alwayson*, modifications pour une meilleure gestion des p-champs et une insertion plus fiable d'une instance d'instrument pour répéter ou redémarrer des sections de partition.
- On a remplacé dans les opcodes de graphe de fluence la bibliothèque multithread OpenMP par `pthreads`, ce qui permet une initialisation en une passe des structures statiques.
- Frontaux :
  - PNaCl est maintenant supporté comme plate-forme, ce qui permet d'exécuter Csound dans le navigateur Chrome sur tous les systèmes d'exploitation qui l'acceptent.
- Bogues corrigés :
  - Opcode *adsynt2*.
  - Opcode *ftgentmpd*.
  - Opcode *dates*.

- Opcode *pvsfilter*.
- Sortie stéréo de *temposcal* et de *mincer*.
- Opcode *pan2*.
- Dépassement d'indice dans *randh* et dans *randi*.
- *pycalln* s'il n'y a pas d'entrée.
- correction/révision des réglages de *ksmps* et de *kr* dans les UDOs.
- correction du problème d'envoi d'un message de partition de max vers csound via csound~ (Ticket 58).
- lorsque *itype* dans *chn\_k* était fixé à 3 et que les valeurs étaient inférieures à 1, Csound6 produisait une erreur INIT. (Ticket 67).
- Plusieurs erreurs de segmentation qui avaient été rapportées ont été traitées.
- l'opcode *xtratim* utilisait une valeur de *ekr* incorrecte venant de csound au lieu d'une instance ; l'utilisation de cet opcode conjointement avec *setksmps* donnait des notes avec une valeur de *xtratim* très longue et ainsi ces notes ne se terminaient jamais.
- Changements système :
  - Il y a maintenant un mode serveur qui accepte des entrées via UDP (avec l'option --port).
  - Un bogue très ancien dans *extract* a été trouvé et corrigé. Cela laisse penser que cet utilitaire est très peu utilisé !
  - La façon dont le générateur de partition externe était écrit a changé de manière conséquente. En particulier, ceci devrait corriger un bogue assez étrange sous Windows.
  - Correction d'un bogue de plantage dans la fonction de rappel d'un canal *invalue* à cause de l'extraction d'un mauvais objet de données depuis les données d'hôte de csound.
  - Correction d'un bogue dans les UDOs sans *ksmps* local lorsque *kcounter* était utilisé de manière incorrecte.
  - Meilleur contrôle dans les canaux.
  - (Expérimental) Si la variable d'environnement CS\_UDO\_DIR est définie, tous les fichiers du répertoire courant avec une extension .udo sont automatiquement inclus au début de l'orchestre. A vérifier pour voir si c'est ce qui était demandé.
  - (Expérimental) Il y a de nouveaux opcodes GPGPU cuda (seulement le code source) : *cudasynth* (3 versions pour la synthèse additive, la synthèse additive de fsigs et la resynthèse par vocoder de phase) et *culdanal* (une version GPGPU de *pvsanal*).
- Changements internes :
  - Plusieurs tentatives d'accélération du code.
  - Amélioration de l'inférence de type et du parseur.
- iOS:

- Correction du plantage lorsqu'aucun `csoundSetHostImplementedMIDIIO` n'est utilisé sous iOS alors que la valeur de `_RTMIDI` est fixée.
- OSX:
  - Correction du nom du périphérique d'entrée pour auhal.

## Notes de parution de Csound6

Csound6 est une réécriture significative d'une grande partie du code. En particulier, l'API n'est plus compatible, mais tous les anciens fichiers au format `orc/sco/csd` devraient fonctionner.

Il y a de nouvelles facilités comme la précision à l'échantillon et le mode temps réel décrit ci-dessous.

**IMPORTANT** : la variables d'environnement localisant les greffons se nomme **OPCODE6DIR64** ou **OPCODE6DIR** (noter le 6) afin que Csound6 puisse coexister avec Csound5.

De même `.csoundrc` est renommé `.csound6rc`.

Les tableaux sont maintenant intégrés avec le support d'une syntaxe et d'opcodes. Ils existent également en format multidimensionnel. Ils sont habituellement créés avec l'opcode *init* ou avec l'opcode *fillarray*.

```
k1[] init 4
```

génère un tableau de taux-k à une dimension, de longueur 4. De même

```
a2[][] init 4, 4
```

crée un tableau carré 4x4 de taux-a.

```
k2[] fillarray 1, 2, 3, 4
```

crée un vecteur de 4 éléments rempli avec 1,...4, qui définit aussi sa longueur.

Les éléments sont atteints par indexation entre [] comme `k1[2]` ou `a2[2][3]`. Les tableaux unidimensionnels remplacent les `tvars` et on peut les utiliser dans des opcodes comme *maxtab*, *mintab* et *sumtab* (voir ci-dessous). Un élément de tableau peut être rempli en le plaçant à la gauche d'un opcode :

```
aSigs[0] vco2 .1, 440  
aSigs[1] vco2 .1, 880
```

Le nouveau mode prioritaire temps réel peut être activé en passant `--realtime` ou en fixant le champ `real-time_mode` de `CSOUND_PARAMS` à 1. Cela provoque les effets suivants :

1. tout opcode de lecture/écriture de fichier audio est traité de manière asynchrone dans un thread séparé.
2. toutes les opérations d'initialisation sont exécutées de manière asynchrone.

Le support du multicoeurs a été entièrement réécrit avec l'utilisation d'un algorithme différent pour la répartition des tâches, ce qui devrait consommer moins de mémoire et moins de verrous.

- Nouveaux opcodes :
  - *faustgen*

- *array* -- plusieurs opcodes nouveaux ou révisés -- voir *Opcodes de tableaux*
- *compileorc* reçoit le nom d'un fichier contenant une collection de définitions d'instrument et les compile en remplaçant les versions existantes s'il y en a. Retourne 0 en cas de réussite.
- *compilestr* est comme *compileorc* mais il reçoit une chaîne de caractères.
- *readscore* exécute le préprocesseur de partition sur une chaîne de caractères et organise ensuite les nouveaux événements via le mécanisme d'événements en temps réel, retournant 0 en cas de réussite.
- Orchestre
  - Les événements de note peuvent commencer et se terminer au milieu d'un cycle-k. Comme c'est un changement incompatible, ce n'est invoqué que si l'option --sample-accurate est spécifiée dans la ligne de commande. A noter que cela ne fonctionne pas avec les notes liées et que lorsque l'initialisation d'une note est ignorée cela peut poser problème.
  - Les instruments peuvent fonctionner avec des valeurs de *ksmps* locales en utilisant *setksmps ikmps* comme dans les UDOs de *Csound5*.
  - La compilation peut s'effectuer à n'importe quel moment, les nouveaux instruments étant ajoutés ou remplaçant les anciens. Les instances actives des anciennes définitions d'instrument ne sont pas affectées. La seule limitation est que les constantes de l'en-tête dans *instr0* ne sont lues qu'une fois lors de la première compilation. Du code d'initialisation peut être placé en dehors des instruments dans l'espace global, et il ne sera exécuté qu'une seule fois après la compilation. Dans ce cas, la génération d'événements de partition peut être complètement remplacée par du code de l'orchestre. Voir aussi les nouveaux opcodes *compileorc* et *compilestr*.
  - Nouvelle syntaxe avec les opérateurs +=, -=, \*= et /=. Ce sont plus que des bonus de syntaxe ; il est recommandé d'utiliser += et -= pour les réverbérations avec accumulation car ils donnent un meilleur comportement multicoeurs.
  - Les opcodes *add*, *sub*, *mul* et *div* ont été supprimés ; utiliser les formes + - \* /. Peu de gens connaissent l'existence de ces opcodes.
  - Tout opcode avec une seule entrée et aucune sortie peut être utilisé comme une fonction. Certains opcodes peuvent nécessiter une annotation de type pour résoudre les ambiguïtés. Plus de détails dans *Syntaxe fonctionnelle dans Csound6*.
  - Une instruction peut aller à la ligne après une virgule, un signe égal ou une opération arithmétique.
  - Il y a un ensemble d'opérations nouvelles ou recodées sur les tableaux de valeurs-k, la plupart restreintes aux tableaux à une dimension (vecteurs) :

<code>kans minarray ktab</code>	retourne la plus petite valeur du tableau (peut-être) multidimensionnel
<code>kans maxarray ktab</code>	comme <code>minarray</code>
<code>kabs sumarray ktab</code>	somme de toutes les valeurs dans le tableau
<code>ktab genarray imin, imax[, inc]</code>	génère un vecteur de valeurs entre <code>imin</code> et <code>imax</code> par pas de <code>inc</code> (1 par défaut)
<code>ktab2 maparray ktab1, "sin"</code>	applique le fonction de <code>sin</code> à 1 arg dans la chaîne, à chaque élément du vecteur
<code>ktab2 maparray_i ktab1, "sin"</code>	applique le fonction de <code>sin</code> à 1 arg dans la chaîne, à chaque élément du vecteur
<code>ktab2 slicearray ktab1, istart, iend</code>	retourne un bout de <code>ktab1</code> allant de <code>ktab1[istart]</code>

	à <i>ktab1[iend]</i>
<i>copyf2array</i> <i>ktab</i> , <i>kfn</i>	<i>copie les données d'une ftable dans un vecteur</i>
<i>copya2ftab</i> <i>ktab</i> , <i>kfn</i>	<i>copie les données d'un vecteur dans une ftable</i>

L'arithmétique sur les tableaux est permise. En particulier l'addition, la soustraction, la multiplication et la division élément par élément sont accessibles en format arithmétique. Les mêmes opérations sont permises entre un tableau et un scalaire.

- Chaque instance d'instrument a un bloc-notes de quatre valeurs persistantes ; on peut ainsi reporter ces valeurs dans une nouvelle utilisation de l'instrument ; peut être utile pour le legato, etc.
- Si le numéro de table -1 est passé, une onde sinus interne équivalente à *f. 0 16382 10 1* est utilisée. L'écriture dans cette table produira des résultats imprévisibles, mais elle n'est pas interdite. La valeur 16382 peut être changée par l'option de ligne de commande *--sine-size=#* où # est arrondi à une puissance de deux.
- Plusieurs opcodes *oscil* ont maintenant un paramètre de *ftable* facultatif, prenant par défaut l'onde sinus interne. (*oscil1*, *oscilli*, *oscil*, *oscil3*, *oscili*, *foscil*, *foscil1*, *loscil*, *loscil3*).
- Partition
  - Les lignes de la partition peuvent avoir plusieurs chaînes de caractères.
  - Changement pour les caractères d'échappement dans les chaînes de caractères de la partition -- ils sont inactifs.
  - Noter aussi l'opcode *readscore*.
- Opcodes et gens modifiés
  - La fonction *k()* peut prendre un argument de *taux-a* et produit dans ce cas un appel à *downsamp*.
- Utilitaires
  - les fichiers d'analyse *hetro/adsyn* peuvent être indépendants de l'ordre des octets s'ils sont créés avec -X. La contrepartie est un fichier un peu plus long et un chargement un peu plus lent. L'utilitaire *het\_export* crée le format indépendant à partir de l'ancien, et *het\_import* n'est plus nécessaire.
  - *cvanal* et *lpanal* produisent des fichiers indépendants de la machine si l'option -X est utilisée. Les opcodes *convolve*, *lpread*, etc, acceptent les deux formats. Il est recommandé d'utiliser le format indépendant de la machine. Les fichiers d'analyse produits avec -X peuvent être utilisés sur d'autres systèmes.
- Frontaux
- Corrections de bogues
- Changements système
  - Sous Linux et OSX le traitement multilingue est maintenant sécurisé pour les threads et local.
- Changements de plate-forme
- API
 

Nouvelles fonctions de l'API...

  - Nouvelles fonctions de configuration et de paramètres



```
PUBLIC int csoundSetOption(CSOUND *csound, char *option);
PUBLIC void csoundSetParams(CSOUND *csound, CSOUND_PARAMS *p);
PUBLIC void csoundGetParams(CSOUND *csound, CSOUND_PARAMS *p);
PUBLIC void csoundSetOutput(CSOUND *csound, char *name, char *type,
                             char *format);
PUBLIC void csoundSetInput(CSOUND *csound, char *name);
PUBLIC void csoundSetMIDIInput(CSOUND *csound, char *name);
PUBLIC void csoundSetMIDIFileInput(CSOUND *csound, char *name);
PUBLIC void csoundSetMIDIOutput(CSOUND *csound, char *name);
PUBLIC void csoundSetMIDIFileOutput(CSOUND *csound, char *name);
```

- Nouvelles fonctions d'analyse et de compilation

```
PUBLIC TREE *csoundParseOrc(CSOUND *csound, char *str);
PUBLIC int csoundCompileTree(CSOUND *csound, TREE *root);
PUBLIC int csoundCompileOrc(CSOUND *csound, const char *str);
PUBLIC int csoundReadScore(CSOUND *csound, char *str);
PUBLIC int csoundCompileArgs(CSOUND *, int argc, char **argv);
```

- Nouvelle fonction pour démarrer Csound après la première compilation

```
PUBLIC int csoundStart(CSOUND *csound);
```

- Nouveaux accès au bus logiciel sécurisés pour les threads

```
PUBLIC MYFLT csoundGetControlChannel(CSOUND *csound, const char *name);
PUBLIC void csoundSetControlChannel(CSOUND *csound, const char *name, MYFLT val);
PUBLIC void csoundGetAudioChannel(CSOUND *csound, const char *name, MYFLT *samples);
PUBLIC void csoundSetAudioChannel(CSOUND *csound, const char *name, MYFLT *samples);
PUBLIC void csoundSetStringChannel(CSOUND *csound, const char *name, char *string);
PUBLIC void csoundGetStringChannel(CSOUND *csound, const char *name, char *string);
```

- Nouvelles fonctions de copie de table sécurisées pour les threads

```
PUBLIC void csoundTableCopyOut(CSOUND *csound, int table, MYFLT *dest);
PUBLIC void csoundTableCopyIn(CSOUND *csound, int table, MYFLT *src);
```

L'API a été sécurisée pour les threads si bien que l'exécution et le contrôle peuvent avoir lieu dans des threads séparés (après un appel à *csoundStart()* ou à *csoundCompile()*). La sécurité pour les threads est assurée par

1. l'usage de lecture/écriture atomique sur les canaux de contrôle
2. des verrous tournants pour les canaux audio et de chaîne de caractères
3. des exclusions mutuelles protégeant la compilation, les événements de partition et les accès aux tables.

- Interne

- Le système de construction est maintenant cmake (et plus scons comme dans Csound5).
- Plusieurs opcodes d'accès aux tables ont été réécrits mais ils se comportent de la même façon. De même *diskin* et *diskin2* utilisent maintenant le même code si bien que *diskin* devrait être plus stable.

- 
- L'ancien parseur est supprimé.

- Nouvelles fonctions internes dans Csound

```
void (*FlushCircularBuffer)(CSOUND *, void *);
void (*FileOpenAsync)(CSOUND *, void *, int, const char *, void *,
                      const char *, int, int, int);
unsigned int (*ReadAsync)(CSOUND *, void *, MYFLT *, int);
unsigned int (*WriteAsync)(CSOUND *, void *, MYFLT *, int);
int (*FSeekAsync)(CSOUND *, void *, int, int);
char *(*GetString)(CSOUND *, MYFLT);
    Extrait une chaîne d'un argument d'évènement de partition.
```

- Fonctions supprimées

```
void (*FileOpen)(CSOUND *, void*, int, const char*, void*, const char*);
```

- La partie "privée" de l'API a considérablement changé. Des structures comme EVTBLK ont également changé.
- Les macros LINKAGE1/FLINKAGE1 ont été renommées en LINKAGE\_BUILTIN/FLINKAGE\_BUILTIN.
- Le modèle pour la passe d'exécution des opcodes de taux-a est

```
int perf_myopcode(CSOUND *csound, MYOPCODE *p)
{
    uint32_t offset = p->h.insdshead->ksmps_offset;
    uint32_t early = p->h.insdshead->ksmps_no_end;
    uint32_t nsmps = CS_KSMPS;
    ...
    if (UNLIKELY(offset)) memset(p->res, '\0', offset*sizeof(MYFLT));
    if (UNLIKELY(early)) {
        nsmps -= early;
        memset(&p->res[nsmps], '\0', early*sizeof(MYFLT));
    }
    for (n=offset; n<nsmps; n++) {
        ....
        p->res[n] = ....
    }
    return OK;
}
```

- Les variables chaîne de caractères ont été réimplantées/
- La structure OENTRY a changé et elle a un nouveau champ de dépendance ; utiliser ce champ selon les nécessités de la sémantique multicoeurs. On peut le mettre à -1 et interdire ainsi tout parallélisme, mais au moins il est sûr.
- Tous les opcodes concernant l'audio doivent prendre en compte le code de précision à l'échantillon.
- Certaines fonctions de la précédente API sont supprimées ; *OpenFile* et *OpenFile2*, toutes deux remplacées par la nouvelle *OpenFile2* avec argument supplémentaire.
- Des ajouts ont été faits aux spécifications du type d'argument pour les opcodes.
  - Les types quelconques ont été ajoutés comme suit :
    - '.' signifie un argument requis de n'importe quel type
    - '?' signifie un argument facultatif de n'importe quel type

- '\*' signifie une liste variable d'arguments de n'importe quel type
- Les tableaux sont maintenant spécifiés par "[x]" où x est une spécification de type. Une spécification de type peut-être n'importe quelle spécification courante, y compris celle des types quelconques. Voir *Opcodes/arrays.c* pour un exemple d'utilisation.
- Nouveau système de type

Un nouveau système de type a été ajouté à Csound6, et des modifications significatives ont été apportées au compilateur. Le système précédent de gestion des types dépendait de la première lettre du nom de variable à chaque détermination de son type. Cela impliquait de retester de nombreuses fois les types. De plus, l'ajout de nouveaux types était difficile, car il fallait modifier beaucoup de code particulier pour tester les nouvelles lettres de type.

Dans Csound6, un système de type séparé a été ajouté. Les types sont définis comme CS\_TYPE's. La création de variables d'un type et l'initialisation de mémoire sont encapsulées dans les CS\_TYPE's. Ce changement facilite l'ajout de nouveaux types ainsi que les calculs génériques de réserve de mémoire, parmi d'autres choses.

Le compilateur a été modifié depuis Csound5 pour utiliser désormais le système de type comme partie intégrale de sa phase de vérification sémantique. Les variables sont maintenant enregistrées dans un CS\_VAR\_POOL à leur première définition, la CS\_VARIABLE ayant une référence à son CS\_TYPE. Après cette première définition dans le bloc, l'information de type est consultée lors des accès ultérieurs à la variable plutôt que recalculée d'après le nom de la variable. Cela ouvre des possibilités pour de nouvelles stratégies de nommage et de typage des variables, à savoir l'utilisation de "myVar:K" pour dénoter un argument de taux-k. Cela ouvre aussi des possibilités pour les types définis par l'utilisateur, tels que "data myType kval, aval", avec l'utilisation ensuite de "myVar:myType" pour définir une variable de ce type. (Ceci est spéculatif, et ce n'est pas une proposition active pour le moment.)

L'ajout du système de type a formalisé le système de type statique qui existait précédemment dans Csound avant Csound6. Il a indubitablement simplifié la base de code en terme de gestion des types ainsi que posé les fondations d'une recherche future sur les types à intégrer dans Csound.

## Nouveautés dans la version 5.19 (7 janvier 2013)

C'est principalement une distribution de correction de bogues avec cependant de nouveaux opcodes et des améliorations.

- Nouveaux opcodes :
  - Module *ipmidi* pour le MIDI sur un réseau.
  - OPCODE *ppltrack*.
  - OPCODE *combinv*.
- GEN et macros :
  - Vérification améliorée dans *GEN28*.
  - Vérification de l'intervalle dans *outrg* et option de repliement.
- Orchestre :
  - Une instruction vide provoque un avertissement.

- Numéros de ligne ajoutés à plusieurs messages d'arguments en entrée (nouveau parseur).
  - Opcodes et GENs modifiés :
    - De meilleurs messages d'erreur et d'avertissement.
    - *loopseg* vérifie maintenant le nombre de ses arguments.
    - *harmon2/3/4* ont été améliorés.
    - *active* : une option a été ajoutée pour ignorer les instances en phase de relâchement.
    - Une nouvelle implémentation mieux testée de ChordSpace.
  - Bogues corrigés :
    - L'optimisation bâclée des filtres passe-bas a été corrigée.
    - Les opcodes *chn* ont été corrigés sous Linux.
    - Un bogue de silence a été corrigé dans *loscil*.
    - *GEN23* a été corrigé lorsqu'un commentaire ne se termine pas par une nouvelle ligne.
    - *loopseg* a été corrigé.
    - Le nombre de canaux d'entrée et de sortie a été corrigé dans le nouveau parseur.
    - Le problème de *GEN43* a été corrigé.
    - *fout* a été corrigé.
    - *centroid* avait tendance à planter.
    - Bogue mineur dans le système d'impression qui perdait les %.
    - Il y avait une valeur non initialisée dans *fold*.
    - Il y avait des valeurs non initialisées dans *dconv*.
    - L'affectation des *fsigs* fonctionne maintenant.
  - Changements système :
    - Des erreurs de segmentation sont évitées lors de certaines erreurs de l'utilisateur.
    - Les opcodes *modal4* sont plus rapides.
    - La compilation *cabbage* est possible.
    - La taille des *pchamps* a été rendue dynamique dans le message d'évènement de *csoundapi~*.
    - Le format de sortie par défaut pour les tuyaux et les réels en double précision est AU.
    - Changement pour *ircam* avec le format par défaut, '-o stdout' et les tuyaux.
- 
- Réels en double précision ajouté au format de sortie.

- Changement sur les plates-formes :
  - Linux :
    - Les spinlocks sont initialisés (bogue corrigés dans les opcodes chn).
  - OSX :
    - Sélection des périphériques améliorée dans le module rtauhal.
    - Un tampon circulaire et une opération sans verrou ont été ajoutés à rtauhal.
    - Le programme d'installation sur MacOSX a été corrigé (en créant des liens symboliques vers lib\_csnd.dylib).
  - Haiku :
    - Nouvelle plate-forme.
  - Android :
    - On peut utiliser -B pour fixer la taille de tampon circulaire.
    - On a ajouté les opcodes fluidsynth.
    - On a ajouté la méthode inputMessage à CsoundObj.
    - Il est possible d'installer CSDPlayer sur une carte SD.
  - iOS :
    - Routage audio amélioré.
    - Le haut-parleur inférieur est choisi par défaut.
  - API :
    - Nouvelle fonction *csoundCompileFromStrings()*.

## Nouveautés dans la version 5.18 (29 août 2012)

C'est principalement une distribution de correction de bogues avec cependant de nouveaux opcodes et des améliorations.

- Nouveaux opcodes :
    - *centroid* semblable à *pvscent* mais agissant sur des signaux audio.
    - *cosseg* comme *linseg* mais avec interpolation cosinusoidale.
    - *cossegb* comme *linsegb* mais avec interpolation cosinusoidale.
    - *cossegr* comme *linsegr* mais avec interpolation cosinusoidale.
    - *joystick* pour lire des valeurs venant d'un joystick externe (seulement sous Linux).
-

- *plater* modélisant une plaque réverbérante carrée.
- *pwd* pour déterminer le répertoire courant
- *readf* pour lire des chaînes de caractères d'un fichier
- *readfi* pour lire des chaînes de caractères d'un fichier à l'initialisation
- *vbap* de la famille *vbap*, flexible sur le nombre de haut-parleurs et sur le choix des dispositions.
- *vbapg* semblable à *vbap* mais ne calculant que les gains sur les canaux.
- Nouvelles fonctionnalités
  - Changements à `<CsOptions>` permettant les espaces entre les mots et les caractères d'échappement.
  - *fout* et *fin* utilisent une meilleure stratégie de mémoire tampon et sont ainsi plus rapides.
  - Il est possible de ne spécifier qu'un fichier d'orchestre avec l'option `--orc`. Utile lorsqu'on a pas besoin de partition.
  - Une nouvelle option de ligne de commande, `--ogg`, a été ajoutée pour avoir aisément une sortie au format ogg/vorbis.
  - On a ajouté *alsaseq* au midi en temps réel.
- Bogues corrigés et améliorations :
  - Les opcodes *dates* pouvaient planter sur les architectures 64 bit ; c'est corrigé.
  - Certains inter-verrous multicoeur étaient erronés. On pense que ce n'était pas encore un problème, mais que ça le serait devenu dans le futur.
  - Il y avait des cas de double fermeture d'un fichier conduisant au plantage en fin de programme.
  - Deux nouvelles fonctionnalités dans *partikkel*. La loi de panoramique pour *channelmasks* peut maintenant être fixée par une table de fonction (second argument facultatif de *partikkel*), et les nouveaux opcodes de support *partikkelget* et *partikkelset*, pour atteindre et modifier les indices de masquage interne de *partikkel*.
  - *follow2* a été retravaillé pour que les calculs au taux-i et au taux-k soient les mêmes.
  - *pvscent* a été corrigé car il retournait la moitié de la valeur correcte.
  - *vbaplsinit* peut créer plus d'une disposition de haut-parleurs utilisables par *vbap/vbapg*. Egalement des diagnostics bien meilleurs sur les dispositions incorrectes.
- Changements internes :
  - Changement du code pour une possible utilisation de bison 2.6.
  - Il est supposé que la version 1.0.19 ou une version ultérieure de *libsndfile* est disponible.
  - Si la partition est omise, une boucle d'attente quasi infinie est générée.

## Nouveautés dans la version 5.17 (mars 2012)

C'est principalement une distribution sans changement majeur mais avec de nombreuses corrections.

- Nouvel opcode :
    - opcode *cell* d'automate cellulaire.
  - Opcodes et GENs modifiés :
    - *active* rapporte maintenant le nombre total d'instruments actifs ou alloués si son argument vaut zéro.
    - Les opcodes de socket TCP *stsend* et *strecv* ont été remodelés selon un schéma logique.
    - Le système DSSI accepte maintenant jusqu'à 9 canaux.
    - *FLsavesnap* fonctionne avec d'autres widgets pour lesquels *imin* > *imax*.
  - Utilitaires :
    - *csbeats* est mieux documenté et il est compilé par défaut ; il propose également plus de durées de notes.
    - Quelques failles de sécurité dans les utilitaires ont été corrigées.
  - Bogues corrigés :
    - L'opcode *unirand* au taux-a a été corrigé.
    - Correction des paramètres régionaux pour les constantes virgule flottante dans l'orchestre.
    - *transegr* a été corrigé.
  - Changements système :
    - La partition peut maintenant durer plus longtemps (changement de la taille de variable temporelle).
    - Une partition vide donne une durée d'exécution très longue (de nombreuses années).
    - Le code Android est distribué.
    - Changement dans l'utilisation des fichiers tmp ; ils sont maintenant tous effacés à la fin de l'exécution (auparavant certains d'entre eux restaient) et la variable d'environnement TMPDIR est utilisée.
    - L'interaction entre les commentaires, les fins de ligne et la fin de fichier a été fixée.
    - Les nombres hexadécimaux sont maintenant autorisés dans l'orchestre.
    - Un orchestre vide ne provoque plus de plantage.
    - L'expansion de macro dans une chaîne a changé.
    - Lorsque qu'il y a un eof dans une macro de partition mal formée, la boucle infinie est évité.
    - Les diagnostics de noms de macro avec arguments et de fuites de mémoire ont été corrigés.
- 
- Changement du préprocesseur : `{{ { }}` à l'intérieur de "..." et de meilleurs diagnostics.

- L'installateur pour Windows a été corrigé afin de supprimer complètement \$INSTDIR\bin de PATH durant la désinstallation : cela nettoie la variable d'environnement PATH lors d'une désinstallation sous Windows. Auparavant, il restait un "\bin" résiduel dans le PATH.
- La classe CsoundAC de MusicModel est plus utilisable par les programmes C++.
- *ftcps* manquait en tant que fonction.
- Changement internes :
  - Beaucoup ! Certains messages modérés, amélioration du code, etc.

## Nouveautés dans la version 5.16 (février 2012)

Le changement principal est que le nouveau parseur est maintenant employé par défaut. L'ancien parseur est toujours disponible en cas de difficultés, mais le nouveau parseur a été testé en profondeur depuis le début de l'année, et il a reçu une restructuration complète de l'expansion des macros. Une des conséquences est l'exécution plus rapide de la plupart des orchestres bien que leur analyse soit plus lente. Quelques optimisations sont implémentées sous la forme de réduction en constante dans des cas simples. Les numéros de ligne et les noms de fichier sont mieux tracés qu'auparavant.

Quelques fuites de mémoire ont été aussi corrigées.

- Nouveaux opcodes :
  - Opcodes adaptés de SuperCollider par Tito Latini: *dust*, *dust2*, *gausstrig*, *gendy*, *gendyc* et *gendyx*.
  - Un générateur de bruit fractal par Tito Latini: *fractalnoise*.
  - Opcodes pour accéder aux valeurs dans les tables par indexation directe, par John ffitich: *ptable*, *ptablei*, *ptable3* et *ptablew*. Ces opcodes sont semblables respectivement à *table*, *tablei*, *table3* et *tablew*, mais ils ne nécessitent pas des tables ayant une longueur en puissance de deux.
- Opcodes et GENs modifiés :
  - Il y avait un problème de borne d'intervalle dans l'opcode *tab* qui pouvait induire une référence hors-limites erronée.
  - GEN15 appelait en interne GEN13 et GEN14, en leur passant des valeurs d'amplitude non initialisées. Problème corrigé.
  - *fmbell* prend maintenant un argument facultatif pour contrôler la durée d'entretien.
  - Changement de base de *pvs* pour les conversion de *tab* vers *table*.
  - *poscil* est maintenant polymorphique, ce qui permet d'avoir des amplitudes et des fréquences de taux-*k* ou de taux-*a*.
  - Le comportement de *p()* et de *i()* a changé lorsque l'argument est de taux-*k*.
  - Le fonctionnement différé GEN49 est corrigé.
  - GEN23 offre maintenant un fonctionnement différé.
- Utilitaires :
  - Indicateur pour utiliser avec le nouveau parseur les fichiers en mémoire.



- Frontaux :
  - Un accès aux tables a été ajouté à csoundapi~ via de nouvelles méthodes get/set.
- Bogues corrigés et améliorations :
  - Un grand nombre dans le nouveau parseur en relation avec la précédence et le multicoeur.
  - De meilleurs diagnostics lorsqu'un fichier ou un csd est manquant.
  - fichier csd : CsFileB et CsSampleB corrigés.
  - L'instruction de partition 'n' a été corrigée.
  - Un bogue dans disk2 produisant une boucle infinie a été corrigé.
  - Un bogue dans hrtfmove causant un fondu-enchaîné bruiteux a été corrigé.
  - Des dépassement de tampon aléatoires ont été corrigés dans certains utilitaires.
  - Une erreur de segmentation dans midicN est maintenant évitée.
  - Bogue dans mp3in lorsque skip=0, corrigé.
  - L'instruction de partition 'r' a été corrigée en ce qui concerne les macros.
  - sndwarp pouvait engendrer une erreur de segmentation.
- Changements système :
  - Les commandes de préprocesseur #if #else #endin fonctionnent.
  - La profondeur d'#includes est maintenant limitée à la place d'une récursion infinie.
  - Si --nodisplays ou -d est utilisé, tous les affichages sont réellement supprimés ; cela corrige le bogue qui, lorsque l'on utilisait -d ou --nodisplays, permettait à la fonction csoundModuleInit de winFLTK.c de fixer des procédures de rappel pour l'affichage ; ce bogue était causé par l'appel d'applications python TK et de CsoundYield\_FLTK.
  - Les fuites de mémoire dans mp3in et dans mp3len ont été corrigées.
- Changement internes :
  - Très, très, très nombreux ! Et le nouveau parseur ...

## Nouveautés dans la version 5.15 (décembre 2011)

- Nouveaux opcodes:
  - opcode *ftab2tab*.
  - opcode *tab2pvs*.
  - opcode *pvs2tab*.
  - opcode *cpumeter*, (pas vraiment nouveau mais désormais disponible sous OSX)
  - opcode *minmax*.

- *opcode ftresize* (EXPERIMENTAL).
- *opcode ftresizei* (EXPERIMENTAL).
- *opcode hrtfearly*.
- *opcode hrtfneverb*.

Nouveau GEN et macros

- Code permettant de différer GEN49 [NB semble ne pas fonctionner]
  - Opcodes et GENS modifiés
    - *socksend* et *sockrecv* n'effectuent plus de test MTFU et fonctionnent sous Windows.
    - *mpulse* a été modifié de façon à ce que si l'évènement suivant a une date négative, sa valeur absolue est utilisée.
    - l'opcode *serial* fonctionne maintenant sous Windows comme sous Un\*x.
    - *out*, *out2*, *outq*, *outh*, *outo*, *outx* et *out32* sont maintenant identiques et prennent autant d'arguments que *nchnls*. Cela remplace le remappage d'opcodes actuel.
    - *turnoff2* est maintenant polymorphe quant aux type S et k (il accepte les noms d'instrument).
  - Bogues corrigés :
    - *GEN42* a été corrigé.
    - *jacko* : une erreur de segmentation a été corrigée en supprimant l'option inutilisée de JackSessionID.
    - la fuite de mémoire de *doppler* a été corrigée.
    - *transegr* a été corrigé en mode release, lorsque la majeure partie de l'enveloppe est ignorée.
    - *FLPack* est maintenant conforme au manuel.
    - *max\_k* est maintenant conforme au manuel.
    - *hrtfneverb* a été corrigé.
    - le code *atsa* fonctionne maintenant sous Windows dans la plupart des cas.
    - bogue de *tabmorph* corrigé.
    - le problème des opcodes définis par l'utilisateur n'ayant pas de sortie est réglé.
    - Différentes corrections aux commentaires de type /\* ... \*/.
  - Changements systèmes :
    - Différents questions de licence ont été réglées.
    - Loris ne fait plus partie de l'arbre de Csound.
- 
- Des fuites de mémoire ont été corrigées

- Si aucune partition n'est fournie, une partition factice avec une durée de 100 ans est créée.
- Tout le traitement de partition se fait en mémoire sans fichier temporaire.
- La mémoire des chaînes de caractère est maintenant extensible et sans limitation de taille.
- `#if #else #end` sont maintenant dans le nouveau parseur.
- Ajustements de la précision des fichier MIDI en sortie.
- Sous OSX, passage de Coreaudio à AuHAL
- Le multicoeur est maintenant sûr avec ZAK, les canaux d'E/S et la modification de tables.
- Nouveau module coremidi.
- Amélioration du clavier virtuel : 1) Menu déroulant pour choisir l'octave de base (celle qui commence avec la touche virtuelle correspondant à la lettre Z du clavier de l'ordinateur). La valeur par défaut est 5, ce qui préserve la compatibilité descendante. 2) Correspondance Maj-X ce qui ajoute deux octaves à la correspondance X pour un total de quatre octaves jouables depuis le clavier de l'ordinateur (en partant de l'octave de base choisie). 3) Correspondances Ctrl-N / Ctrl-Maj-N pour incrémenter / décrémenter la réglette du contrôleur N. 4) La roulette de la souris contrôle maintenant les réglettes.
- type `tsig` pour les vecteurs.
- `tsigs` et `fsigs` autorisés comme arguments des UDOs
- API : version mineure augmentée.
- Changements internes :
  - Très, très, très nombreux !
  - 
  -

## Nouveautés dans la version 5.14 (octobre 2011)

- Nouveaux opcodes :
  - opcode *mp3len*.
  - opcode *qnan*.
  - opcode *qinf*.
  - opcode *exprandi*.
  - opcode *cauchy*.
  - opcode *gaussi*.
  - opcode *cpumeter*.
- opcode *linsegb*.

- opcode *expsegb*.
- opcode *transegb*.
- opcode *expsegba*.
- opcode *pvsgain*.
- opcode *pvsbufread2*.
- opcodes *serial*.
- opcodes *lua*.
- opcode *plustab*.
- opcode *multtab*.
- opcode *maxarray*.
- opcode *minarray*.
- opcode *sumarray*.
- opcode *scalearray*.
- Nouvelles fonctionnalités :
  - le processeur beats est renommé csbeats et distribué
  - l'utilité mkdb qui fournit un catalogue des greffons opcodes/bibliothèques
  - la bibliothèque ladspa construite dans le système par défaut
  - les macros sont maintenant déroulées en chaînes dans la partition
  - il y a une syntaxe de boucle until .. do .. od (seulement avec le nouveau parseur)
  - les signaux SIGPIPE sont ignorés plutôt que de provoquer la sortie de Csound
  - il est possible d'utiliser des vecteurs de valeurs de taux-k, appelés variables-t. Ils sont initialisés à une taille donnée et peuvent être lus avec la syntaxe simple []. L'affectation des composantes ne se fait qu'avec =. Il y a aussi quelques nouveaux opcodes qui fournissent des fonctionnalités plus larges.
- Bogues corrigés et améliorations :
  - la lecture de valeurs pour remplir des tables était défectueuse à cause de commentaires
  - l'erreur interne dans wii\_data a été corrigée
  - pvsshift a été corrigé
  - jacko a été corrigé
  - petite correction dans gen23

- atsaadd corrigé
  - compress corrigé pour fonctionner avec Odbfs
  - le compteur de position de pvsubread a été corrigé
  - l'opcode tempo a été corrigé
  - la section CsFileB des fichiers .csd était défectueuse, c'est corrigé
  - les tables différées de gen01 pouvait avoir une taille erronée
  - vbap\_zak rendu fonctionnel (!)
  - le problème de mémoire dans ATSSinoi est corrigé
  - différentes corrections dans cscore
  - différentes corrections dans partials et dans tradsyn
  - transegr pouvait planter dans certains cas
  - les opcodes loris proviennent de la dernière version
  - l'opcode date a une nouvelle origine sur certaines plates-formes pour éviter les dépassements
  - pvsublur fonctionne maintenant après un reinit
  - disk1, disk2 et soundin peuvent lire maintenant jusqu'à 40 canaux
  - prints se comporte mieux avec les arrondis
  - fmpcrl a maintenant un vibrato qui fonctionne
  - atreson a maintenant un paramètre de gain au taux-k
  - l'opcode comb est consolidé pour pouvoir accepter le même argument en entrée et en sortie
  - précision accrue dans line et expon
  - OSCsend retrouve l'espace préalablement perdu
  - OSCsend peut envoyer une table sous forme d'objet binaire (blob) avec la balise T -- expérimental et non testé
  - lpf18 a maintenant un argument facultatif iskip
  - i() accepte aussi une valeur de taux-i. Dans ce cas, c'est un no-op (opcode neutre)
  - makecsd a été révisé et étendu pour avoir des options pour le traitement du MIDI et des partitions et les licences
  - lpanal revu pour supprimer les boques et des étrangetés
  - un problème de bruit dans alsa et un clic dans portaudio ont été corrigés
- 
- le pilote portaudio a été modifié pour être plus robuste lors du stop/exit

- Changements internes :
  - De nombreuses modifications dans le nouveau parseur pour le rendre opérationnel, mais à utiliser avec précaution
  - Le système multi-coeurs est distribué en mode expérimental et doit être utilisé avec beaucoup de précautions.

## Nouveautés dans la version 5.13 (janvier 2011)

- Nouveaux opcodes :
    - opcode *median*.
    - opcode *filevalid*.
    - opcodes de traitement spectral *pvstanal*, *pvs warp*, *temposcal* et *pvslock*.
    - opcode *mincer*.
    - opcode de suite de *fareylen*.
  - Nouvelles fonctionnalités :
    - Générateurs de nombres aléatoires réels utilisant `/dev/random` (seulement sur Linux).
    - macro INF ajoutée aux orchestres ; z se lit infini dans les partitions
    - *init* a été changé pour permettre plusieurs initialisations en une seule instruction
    - GEN pour supporter les suites de Farey
    - *maxalloc*, *cpuprc* et *active* acceptent maintenant les instruments nommés.
    - Si la normalisation dans les opcodes *pow* vaut 0, elle est traitée comme si elle valait 1
    - *inch* peut prendre jusqu'à 20 entrées et sorties.
    - *pvscale*, *pvsvoc* et *pvmix* ont maintenant de très bons modes de préservation d'enveloppe spectrale (1 = cepstrum filtré, 2 = enveloppe véritable).
    - *oscill* pouvait être statique si la durée était trop longue ; maintenant il y a un incrément positive minimum.
    - GEN49 utilise maintenant les chemins de recherche.
  - Bogues corrigés et améliorations :
    - Le compte des lignes a été fixé dans les orchestres, ainsi que le caractère \ dans les chaînes.
    - Les opcodes rapides *tab* sont protégés contre les plantages
    - % dans les impressions formatées pouvait planter
    - La double libération de mémoire dans *fgen* a été fixée
- 
- *sndwarp* est plus discret (il donnait trop de messages)

- gen41 utilise des probabilités positives
- *adsynt* a été retravaillé pour supprimer de nombreux bogues
- L'erreur de phase *deadsynt2* a été fixée
- Le bogue du nombre maximum de gens a été fixé
- Meilleur test dans *grain4*
- Meilleur test dans *adsyn*
- Le module était erroné dans le nouveau parseur
- *atonex/tonex* faisaient de fausses opérations
- *mp3in* pouvait répéter le son en fin de fichier
- l'opcode *changed* s'initialise à zéro
- Un bogue sérieux a été corrigé dans *tabmorph*
- Un bogue sérieux a été supprimé de GEN49, ce qui fait qu'il n'y a plus de longs silences incorrects.
- Opcode *partikkel* : un bogue de placement de grain dans un sous-échantillon lorsque l'on utilise la MF du taux de grains a été fixé
- Changements internes :
  - On trouve dans le nouveau parseur seulement les opérateurs @ et @@ pour arrondir l'entier suivant à une puissance de 2 ou à une puissance de 2 + 1
  - Le tri de la partition a été rendu beaucoup plus rapide
  - *lineto* a été amélioré
  - Les gens nommés sont autorisés
  - Les divers affichages comprennent le nom d'instrument si celui-ci est disponible
  - Une option de ligne de commande pour omettre le chargement d'une bibliothèque
  - Le nombre de canaux de sortie n'est plus contraint à égaler celui des canaux d'entrée
  - Plusieurs corrections au nouveau parseur
  - Les avertissements sont plus utilisés que les messages (ce qui permet de les désactiver)
  - *csoundSetMessageCallback* est réinitialisé si *callback* vaut null

## Nouveautés dans la version 5.12 (janvier 2010)

- Nouveaux opcodes :
  - *transegr* est une version de l'opcode *transeg* qui a une section de relâchement déclenchée par midi, par un opcode *turnoff2* ou par un évènement de partition *i* dont le numéro d'instrument est négatif.

- *ftgenonce* génère une table de fonction depuis la définition d'un instrument, sans duplication de données.
- *passign* permet une initialisation rapide de variables de taux-i à partir de p-champs.
- *crossfm* implémente la synthèse par modulation de fréquence croisée.
- *loopxseg* est comme *loopseg* mais avec une enveloppe exponentielle.
- *looptseg* est comme *loopseg* mais avec une enveloppe flexible comme *transeg*.
- Bogues corrigés et améliorations :
  - *pvshift* écrasait les données en mode double.
  - Le cas 3 de *pan2* a été fixé.
  - 
  - *clockon* et *clockoff* fonctionnent à nouveau.
  - *cross2* et *interp* pouvaient avoir des divisions par zéro.
  - Le numéro de ligne dans les messages d'erreur n'inclut plus de texte de *.csoundrc*.
  - *p5gconnect* a été modifié pour utiliser un processus léger séparé afin d'éviter les problèmes de temps mort.
  - *transeg* vérifie le nombre de ses arguments.
  - *sfload* se limitait à 10 soundfonts et n'était pas sécurisé. Il n'est plus limité.
- Changements Internes :
  - `\` peut être utilisé comme caractère d'échappement dans les chaînes de l'orchestre.
  - Le nouveau parseur a été corrigé pour les arguments facultatifs.
  - Meilleure vérification de l'instruction *f* avec un nombre négatif.
  - Les soundfonts n'initialisent le tableau des hauteurs qu'une seule fois dans les opcodes de soundfont.
  - La collection usuelle de changements mineurs, de mises en forme et de commentaires.

## Nouveautés dans la version 5.11 (juin 2009)

- Nouveaux opcodes :
  - *mp3in* : permet la lecture des fichiers mp3 directement dans l'orchestre.
  - *wiiconnect*, *wiidata*, *wiisend*, *wiirange* opcodes par john ffitch pour recevoir et envoyer des données de ou vers un contrôleur wiimote.
  - Nouveaux opcodes pour recevoir des données directement d'un P5 Glove (gant de données) par john ffitch *p5gdata*
  - *tabsum* additionne des sections de ftables



- *MixerSetLevel\_i* une version du taux d'initialisation seulement de *MixerSetLevel*
- *doppler* implémente une simulation d'effet Doppler.
- *filebit* retourne la profondeur binaire d'un fichier.
- Les nouveaux *opcodes de Graphe de Fluence* permettent l'utilisation de graphes de fluences (graphes de flots de données asynchrones) dans Csound.
- Nouvelles fonctionnalités :
  - Nouveau type de panning pour l'opcode pan2
  - Nouvelle balise <CsExScore> de partition csd.
  - Nouvelle option -Ma pour le module ALSA RT MIDI qui écoute tous les périphérique.
  - Il y a un GEN49 pour lire des fichiers mp3.
  - Un code d'arrondi de bin a été ajouté à *pvscale*
  - Le support de taille de table non puissance de 2 a été ajouté à *ftload* et à *ftsave*
  - GEN23 a été totalement réécrit pour être plus consistant dans ce qui constitue un séparateur et des commentaires. (Il n'y a toujours pas de commentaires /\* \*/)
- Bogues corrigés et améliorations :
  - Nouveaux exemples pour les opcodes pvs par by Joachim Heintz : pvsarp, pvscent, pvsbandp, pvsbandr, pvsbufread, pvsadsyn, pvsynth, pvsblur, pvscale, pvscross, pvsfilter, pvsfreeze, pvshift, pvsmaska, pvsmorph
  - L'utilisation de la numérotation automatique des ftables réutilise les numéros de table
  - seed avec un argument positif était défectueux
  - sprintf avec une chaîne vide imprimait des données erronées
  - mute fonctionne maintenant à la fois avec les instruments numérotés et nommés
  - Petites corrections dans diskini et dans tablexkt
- Changements Internes :
  - SConstruct construit maintenant des bibliothèques partagées complètement indépendantes pour les adaptateurs de Python, Lua et Java.
  - Le nouveau Parseur est presque fonctionnel
  - Le retraçage des graphiques a été modifié de façon à ne redessiner que ceux qui sont sélectionnés.
  - Alsa-TR est plus tolérant sur les taux d'échantillonnage approchés.
  - Il est possible d'avoir une partition générée par un programme externe plutôt que d'utiliser le format de partition standard en spécifiant <CScore bin="translator"> pour appeler le programme de traduction des données de la partition.

- `lpc_export` corrigé.
- La limite sur la longueur des noms de macro a été supprimée.
- `PMAX`, le nombre d'arguments d'un évènement de partition a été réduit de 2 unités, et un système de dépassement a été introduit pour que les GENs puissent avoir un nombre arbitraire d'arguments.
- La version de l'API a été incrémentée à 2.1.
- Nouvelle fonction de l'API `ldmemfile2withCB()` qui est une version de `ldmemfile()` qui permet de spécifier un callback appelé exactement une seule fois pour traiter le tampon `MEMFIL` après son chargement.
- `csound->floatsize` corrigé; valait zéro dans les versions précédentes.
- `GetChannelLock` ajouté.

## Nouveautés dans la version 5.10 (décembre 2008)

- Nouvelles fonctionnalités :
  - Nouvelle option pour écouter tous les périphériques MIDI avec le module temps réel de `portmidi`. Pour activer l'écoute de tous les périphériques utiliser "`-+rtmidi=portmidi -Ma`".
  - Implémentation du dithering sur la sortie ; le dithering rectangulaire et triangulaire est disponible dans certains cas.
  - Le type 6 de *GEN20* a maintenant une option pour fixer la variance.
- Bogues corrigés et améliorations :
  - La variable d'environnement `Locale` a été fixée à "`C numeric`" pour éviter les problèmes de point décimal (, vs .).
  - Bogue corrigé dans *diskin*
  - *outo* était défectueux pour le canal 6
  - Bogue corrigé dans *pitchamdf*
  - L'initialisation de *zfilter2* a été corrigée
  - Bogue corrigé dans *s32b14*
  - D'autres bogues qui n'avaient pas été publiés ont été corrigés.
- Changements Internes :
  - La version majeure de l'API de Csound est incrémentée à 2, ceci affectant aussi `csound.so`. Ceci signifie que Csound 5.10 est incompatible avec les applications (frontaux, clients ou hôtes) construites pour Csound 5.08 et pour les versions antérieures qui utilisent la version 1.x de l'API. Il faudra reconstruire ces applications pour qu'elles fonctionnent avec les versions courante ou futures de Csound. Les frontaux pour Csound écrits dans des langages interprétés tels que Python ou Java pourront continuer à fonctionner sans modification. Il est également possible de garder une version antérieure de la bibliothèque de Csound et une version de l'API 2.0 sur la même machine afin que les applications basées sur l'ancienne ou sur la nouvelle version de Csound puissent coexister. Ces changements n'a-

fectent en rien la compatibilité des orchestres et des partitions de Csound : tous les anciens documents devraient continuer à fonctionner comme par le passé.

- Le temps est maintenant mesuré en interne en échantillons, ce qui résoud un ancien bogue concernant l'arrondi du temps au taux-k.
- Plusieurs changements internes concernant les branchements. Certains opcodes sont significativement plus rapides.
- 

## Nouveautés dans la version 5.09 (octobre 2008)

- Nouveaux opcodes :
  - Nouvel opcode *vosim* par Rasmus Ekman qui recrée la technique historique VOSIM (VOcal SIMulator).
  - Nouvel opcode *dcblock2* par Victor Lazzarini.
  - Nouveau modèle de l'oscillateur de Chua : *chuap* par Michael Gogins.
  - Nouveaux opcodes d'*Algèbre Linéaire* par Michael Gogins. Algèbre linéaire standard sur vecteurs et matrices réels et complexes : arithmétique composante à composante, normes, transposition et conjugaison, produits scalaires, matrice inverse, décomposition LU, décomposition QR et décomposition QR en valeurs propres. Inclut la copie de vecteur de et vers des signaux de taux-a, des tables de fonction et des signaux-f.
  - Nouveaux opcodes ambisonic : *bformdec1* et *bformenc1*. Ces opcodes rendent obsolètes les anciens *bformdec* et *bformenc*.
  - Nouveaux opcodes de contrôle de la partition par Victor Lazzarini : *rewindscoreet setscorepos*.
- Nouvelles fonctionnalités :
  - Les opcodes de la famille *vbap* (*vbap4*, *vbap8*, *vbap16* et *vbapz*) acceptent maintenant des variables de taux-k pour tous leurs arguments en entrée.
  - Nouveau module d'E/S pulseaudio sous Linux.
  - Nouveau paramètre facultatif *ienv* pour générer des enveloppes pour les opcodes de soundfont : *sfplay*, *sfplay3*, *sfplaym* et *sfplay3m*.
  - Ajout d'un 'argument de non-normalisation' à la routine GEN nommée "tanh". (Voir *Routines GEN Nommées*)
  - Ajout d'une option d'ordonnanceur de priorité sur alsa.
- Bogues corrigés et améliorations :
  - Notation scientifique permise dans *GEN23* (comme c'était le cas dans *csound4* !).
  - Bogue fixé dans l'initialisation de FLTK. L'utilisation de FLTK devrait être plus stable.
  - L'erreur sur les commentaires */\* \*/* dans l'orchestre a été corrigée.
  - *poscil* n'écrase plus la fréquence si la variable est partagée.

- *printk* et *printks* vérifient que l'opcode est initialisé.
- *soundout* et *soundouts* sont déclarés obsolètes en faveur de *fout*.
- L'opcode *space* a été modifié pour accepter des tables dont la taille n'est pas une puissance de 2 (taille différée).
- Un bogue de *pvsmorph* a été corrigé.
- Changements Internes :
  - Le nouveau parseur reconnaît *#include* et les macros sans argument.
  - Moins de forçage de type entre floats et doubles dans la version float.
  - Un support expérimental multicore a été inclut.
  - L'opcode *buzz* a été réécrit.
  - Plusieurs autres changements internes et quelques corrections de bogues.

## Nouveautés dans la version 5.08 (février 2008)

- Nouveaux opcodes :
  - *imagecreate*, *imagesize*, *imagegetpixel*, *imagesetpixel*, *imagesave*, *imageload* et *imagefree*: nouveaux opcodes de traitement d'image par Cesare Marilungo pour lire/écrire des images png depuis Csound.
  - *pvsbandp* et *pvsbandr* par John ffitich, qui réalisent le filtrage passe-bande et réjection de bande dans le domaine spectral sur un signal pvs.
  - Nouveaux opcodes HRTF par Brian Carty : *hrtfmove*, *hrtfmove2* et *hrtfstat*.
  - Nouveau opcodes de distorsion non-linéaire : *powershape*, *polynomial*, *chebyshevpoly*, *pdclip*, *pdhalf*, *pdhalfy*, et *syncphasor*
  - Nouvel opcode de contrôle de transport jack : *jacktransport*
- Nouvelles fonctionnalités :
  - Ajout de l'option de ligne de commande *--csd-line-nums=* pour sélectionner la façon de rapporter la ligne d'une erreur.
  - Nouvel opérateur de "non-report" (!) du langage de partition qui empêche le report implicite des p-champs dans les instructions i.
  - Ajout de l'option de ligne de commande *--syntax-check-only* (mutuellement exclusive avec *--i-only*)
  - Balise *<CsLicence>* pour les *CSDs*. *<CsLicense>* est acceptée comme une alternative à *<CsLicence>*.
- Bogues corrigés et améliorations :
  - L'ordre des sorties pour *hilbert* a été changé. Ce changement brise la compatibilité avec les versions précédentes, mais il fixe l'opcode qui travaille maintenant comme c'est décrit dans la documentation.

- Les messages sur le chargement de greffons d'opcode ont été modifiés pour pouvoir être supprimés avec une option de niveau de message.
- Changements majeurs sur les rapport d'erreur de partition ; les numéros de ligne dans la chaîne des entrées sont rapportés précisément pour la plupart des erreurs.
- *pan2* a été corrigé afin d'être conforme à la documentation.
- La balise *<CsVersion>* fonctionne à nouveau en conformité avec le manuel.
- Les instructions de boucle { et } ont été fixées. Leur documentation ainsi que celles des opérateurs des expressions de partition ~, &, |, et # ont été ajoutées.
- *hilbert* avait ses sorties permutées, c'est corrigé. L'exemple de manual a été mis à jour.
- Changements Internes :
  - Changement de la localisation pour gettext ; les traductions en français et en espagnol (Colombie) sont disponibles.
  - Changements internes dans *partikkel*, interpolation de la lecture de forme d'onde et du fenêtrage, ce qui permet une synthèse granulaire synchrone de hauteur plus précise. Exemples mis à jour pour *partikkel*.
  - *pvscale* : algorithme amélioré pour la SDFT, supprimant la variation d'amplitude.

## Nouveautés dans la version 5.07 (octobre 2007)

- Nouveaux opcodes :
  - *pan2* : un opcode de spatialisation stéréo
  - *cpsmidinn*, *pchmidinn*, *octmidinn* : des convertisseurs de numéros de note MIDI
  - *fluidSetInterpMethod* : interpolation dans les SoundFonts de fluid
  - *sflooper* : une version SoundFont de *flooper2*
  - *pvsbuffer* et *pvsbufread* : mise en tampon/lecture de fsigs pour des changements de retards/échelle temporelle.
- Nouvelles fonctionnalités :
  - SDFT - la Transformée de Fourier Discrète à fenêtre Glissante -- intégrée aux opcodes *pvsanal*, etc si le recouvrement est inférieur à *ksmps* ou inférieur à 10. Certains opcodes *pvsXXX* sont étendus pour prendre des paramètres de taux-a dans cette situation.
  - Nouvelle option (*-O null* / *--logfile=null*) qui désactive tous les messages et toutes les impressions sur la console.
- Bogues corrigés et améliorations :
  - *partikkel* -- la synthèse par particule avait un bogue accidentel, corrigé.
  - La fermeture de l'entrée MIDI sur Windows(MM) échouait ; corrigé.

- L'opcode *fluidEngine* prend maintenant comme paramètres facultatifs le nombre de canaux (compris entre 16 et 256) et le nombre de voix à jouer en polyphonie (compris entre 16 et 4096, la valeur par défaut étant 4096).
- L'utilitaire *atsa* se comporte de manière plus sûre lorsqu'il reçoit du silence.
- *ATSaddnz* : vérifications améliorées.
- Les ambisonics (*bformdec*, *bformenc*) ont plus d'options pour le contrôle des opposés.
- Le bogue dans *turnoff2* est corrigé.
- *het\_export* : une vérification erronée plantait l'export.
- Changements Internes :
  - L'installateur sous Windows a été amélioré.
  - CsoundVST a été remplacé par CsoundAC, qui ne dépend pas des en-têtes du SDK de VST.
  - Moins de messages lors du démarrage sous Windows(MM).
  - Le type d'argument p (le taux-k vaut alors 1 par défaut) a été ajouté dans les types des paramètres d'entrée et de sortie des opcodes.

## Nouveautés dans la version 5.06 (juin 2007)

- Nouveaux opcodes granulaires : *partikkel*, *partikkelsync* et *diskgrain*.
  - Nouvel opcode pour distribuer des événements : *scoreline*.
  - Plusieurs nouveaux opcodes en provenance de CsoundAV de Grabriel Maldonado : *hvs1*, *hvs2*, *hvs3*, *vphaseseg*, *inrg*, *outrg*, *lposcila*, *lposcilsa*, *lposcilsa2*, *tabmorph*, *tabmorpha*, *tabmorphi*, *tabmorphak*, *trandom*, *vtable1k*, *slider8table*, *slider16table*, *slider32table*, *slider64table*, *slider8tablef*, *slider16tablef*, *slider32tablef*, *slider64tablef*, *sliderKawai* et la version au taux-a de *ctrl7*.
  - Egalement depuis CsoundAV, plusieurs nouveaux widgets FLTK : *FLkeyIn*, *FLslidBnk2*, *FLvs-lidBnk*, *FLvs-lidBnk2*, *FLmouse*, *FLxyin*, *FLhvsBox*, *FLslidBnkSet*, *FLslidBnkSetk*, *FLslidBnk2Set*, *FLslidBnk2Setk*, *FLslidBnkGetHandle*,
  - De nouveaux opcodes pvs : *pvsdiskin*, *pvs-morph*,
  - *eqfil*
  - De nouvelles options de ligne de commande (*--m-warnings*) pour contrôler les messages
  - *csladspa* : un kit de greffon CSD vers LADSPA.
  - Et plusieurs corrections de bogues parmi lesquelles : version au taux-k de *system* ; problèmes de changement d'échelle de *vrandh* et de *vrandi* ; plantage occasionnel de *turnoff* ; bogue OS X ; *ATScross* et *mod*.
- Csound5GUI fonctionne maintenant correctement sur toutes les plates-formes et csoundapi~ (objet pd) a été mis à jour.

---

# Partie I. Vue d'ensemble

---

---

# Table des matières

Introduction .....	4
La commande Csound .....	5
Ordre de priorité .....	5
Description de la syntaxe de la commande .....	5
Ligne de commande de Csound .....	7
Options de ligne de commande (par catégorie) .....	19
Variables d'environnement de Csound .....	32
Format de fichier unifié pour les orchestres et les partitions .....	34
Description .....	34
Exemple .....	37
Fichier de paramètres de ligne de commande (.csound6rc) .....	38
Prétraitement du fichier de partition .....	38
La fonction Extract .....	38
Prétraitement indépendant avec Scsort .....	38
Utiliser Csound .....	40
Sortie console de Csound .....	40
Comment Csound fonctionne .....	42
Valeurs d'amplitude dans Csound .....	43
Audio en temps réel .....	44
Entrées/sorties en temps réel sous Linux .....	45
Mac OSX .....	51
Windows .....	52
Entrées/sorties en temps réel avec le kit de connexion de JACK .....	53
Optimisation de la latence audio en E/S .....	54
Configuration .....	56
Syntaxe de l'orchestre .....	57
Instructions de l'en-tête de l'orchestre .....	58
Instructions de bloc d'instrument et d'opcode .....	58
Instructions ordinaires .....	59
Types, constantes et variables .....	59
Initialisation de variable .....	60
Expressions .....	61
Répertoires et fichiers .....	61
Nomenclature .....	62
Macros .....	62
Instruments nommés .....	63
Opcodes définis par l'utilisateur (UDO) .....	65
Vecteurs et tableaux .....	66
Syntaxe fonctionnelle dans Csound6 .....	66
Serveur UDP .....	67
La partition numérique standard .....	70
Prétraitement des partitions standard .....	70
Carry .....	70
Tempo .....	71
Sort .....	71
Instructions de partition .....	72
Symboles next-P et previous-P .....	73
Ramping .....	73
Macros de partition .....	74
Partition dans plusieurs fichiers .....	76
Evaluation des expressions .....	77



Chaînes de caractères dans les p-champs .....	78
Frontaux .....	80
CsoundAC .....	81
Construire Csound .....	84
Liens Csound .....	85

---

# Introduction

Csound est un système de musique par ordinateur basé sur des générateurs unitaires et programmable par l'utilisateur. Il fut écrit à l'origine par Barry Vercoe au Massachusetts Institute of Technology en 1984 comme la première version en langage C de ce type de logiciel. Depuis, Csound a reçu de nombreuses contributions de la part de chercheurs, de programmeurs et de musiciens du monde entier.

Vers 1991, John Fitch porta Csound sur Microsoft DOS. De nos jours, Csound tourne sur plusieurs variétés de UNIX et de Linux, sur Microsoft DOS et Windows, sur toutes les versions du système d'exploitation du Macintosh y compris Mac OS X, et sur d'autres systèmes.

Il y a des systèmes de musique par ordinateur plus récents qui ont des éditeurs graphiques de patch (par exemple Max/MSP, PD, jMax, ou Open Sound World), ou qui utilisent des techniques d'ingénierie logicielle plus avancées (par exemple Nyquist ou SuperCollider). Cependant Csound possède toujours l'ensemble le plus important et le plus varié de générateurs unitaires, est le mieux documenté, s'exécute sur le plus grand nombre de plates-formes, et il est très facilement extensible. Il est possible de compiler Csound en utilisant l'arithmétique double précision pour obtenir une qualité sonore supérieure. Bref, on peut considérer Csound comme l'un des instruments de musique les plus puissants jamais créés.

En plus de cette version "canonique" de Csound et de CsoundAC, il existe d'autres versions de Csound et d'autres frontaux pour Csound, dont la plupart se trouvent sur <http://csound.github.io>.

---

# La commande Csound

La commande *csound* est un frontal de base au système que l'on peut utiliser pour générer une sortie son à partir d'un fichier *orchestre* et d'un fichier *partition* (ou d'un *fichier csd* unifié). Elle est conçue pour être appelée depuis un terminal ou une fenêtre DOS. En plus de celle-ci, il y a d'autres *frontaux*, peut-être plus faciles à utiliser. Le fichier partition peut être codé dans différents formats, au choix de l'utilisateur. La traduction, le tri et le formatage de la partition dans un texte numérique lisible par l'orchestre sont effectués par différents préprocesseurs ; tout ou partie de la partition est ensuite envoyé à l'orchestre. L'exécution de l'orchestre est influencée par des *options de commande*, qui fixent le niveau des comptes-rendus graphiques et de console, spécifient les noms des fichiers d'E/S et les formats d'échantillonnage, et déclarent la nature de la détection et du contrôle en temps réel.

## Ordre de priorité

On peut fixer les options d'exécution de Csound en cinq endroits. Elles sont traitées dans l'ordre suivant :

1. Les valeurs par défaut de Csound
2. Le fichier défini par la *variable d'environnement* CSOUND6RC, ou le fichier .csound6rc dans le répertoire HOME
3. Le fichier .csound6rc dans le répertoire courant
4. La balise <CsOptions> dans un fichier .csd
5. En les passant sur la ligne de *commande* de Csound

Les dernières options dans la liste vont écraser les éventuelles options précédentes. A partir de la version 5.01 de Csound, les taux d'échantillonnage et de contrôle (options *-r* et *-k*) spécifiés n'importe où prévalent sur les valeurs sr, kr et ksmps définis dans l'en-tête de l'orchestre.

## Description de la syntaxe de la commande

La commande *csound* est suivie par un ensemble d'*Options de Ligne de Commande* et par les noms des fichiers de l'orchestre (*.orc*) et de la partition (*.sco*) ou du *Fichier Unifié csd* (contenant à la fois l'orchestre et la partition) à traiter. Les *Options de Ligne de Commande* pour contrôler la configuration d'entrée et de sortie peuvent apparaître n'importe où dans la ligne de commande, séparées ou collées ensemble. Un drapeau nécessitant un Nom ou un Nombre le trouvera dans l'argument lui-même ou dans celui qui le suit immédiatement. Les commandes suivantes sont donc équivalentes :

```
csound -nm3 nomorchestre -Sxxnomfichier nompartition  
csound -n -m 3 nomorchestre -x xnomfichier -S nompartition
```

Tous les drapeaux et les noms sont optionnels. Les valeurs par défaut sont :

```
csound -s -otest -b1024 -B1024 -m7 -P128 nomorchestre nompartition
```

où *nomorchestre* est un fichier contenant le code de l'orchestre Csound, et *nompartition* est un fichier de données de partition en format de partition numérique standard, facultativement pré-trié et réajusté en temps. Si *nompartition* est omis, il y a deux options par défaut :

1. si l'on attend une entrée en temps réel (par exemple *-L*, *-M*, *-iadc* ou *-F*), un fichier partition factice est utilisé, constitué de la seule instruction 'f 0 3600' (c'est-à-dire écouter sur l'entrée TR pendant une heure)

2. sinon Csound utilise le dernier *score.srt* produit dans le répertoire courant.

Csound rend compte des différentes étapes de traitement de la partition et de l'orchestre lors de l'exécution, effectuant différents tests de syntaxe et d'erreurs. Une fois l'exécution commencée, les messages d'erreur proviennent soit du chargeur d'instrument soit des générateurs unitaires eux-mêmes. Une commande Csound peut inclure toute combinaison d'options bien formée.

## Exécuter les exemples du manuel à partir de la ligne de commande

La plupart des exemples du manuel sont prêts à l'emploi sans avoir besoin d'ajouter des options de ligne de commande, car ces options sont fixées dans la balise <CsOptions> du fichier csd, si bien qu'il suffit de taper une commande telle que :

```
csound oscil.csd
```

depuis le répertoire des exemples, et une sortie audio en temps réel sera générée.

# Ligne de commande de Csound

csound

## Description

La commande *csound* exécute Csound.

## Syntaxe

```
csound [options] [nomorch] [nompartition]
```

```
csound [options] [nomfichiercsd]
```

## Options de la ligne de commande de Csound

Ci-dessous la liste par ordre alphabétique des options de ligne de commande disponibles dans Csound 6. Les implémentations sur différentes plates-formes peuvent ne pas réagir de la même façon à certaines options ! On peut consulter les options de ligne de commande par catégorie dans la section *Options de la Ligne de Commande (par Catégorie)*.

Les arguments de la ligne de commande sont de 2 types : arguments *options* (commençant par « - », « -- » ou « ++ »), et arguments *nom* (tels que noms de fichier). Certains arguments option sont suivis d'un nom ou d'un argument numérique. Les options qui commencent par « -- » et « ++ » prennent habituellement elles-mêmes un argument précédé du signe « = ».

### Options de la ligne de commande

-@ FICHIER	Une ligne de commande étendue est fournie par le fichier « FICHIER »
-3, --format=24bit	Utiliser des échantillons audio de 24 bit.
-8, --format=uchar	Utiliser des échantillons audio en caractères non-signés sur 8 bit.
--format=type	Choisir le format du fichier de sortie audio parmi les formats disponibles dans libsndfile. Actuellement la liste est aiff, au, avr, caf, flac, htk, ircam, mat4, mat5, MPC, nist, ogg, paf, pvf, raw, sd2, sds, svx, voc, w64, W64, wav, wavex, WVE, xi. On peut aussi écrire --format=type:format ou --format=format:type pour fixer le type du fichier (wav, aiff, etc.) et le format d'échantillonnage (short, long, float, etc.) en même temps.
-A, --aiff, --format=aiff	Ecrire un fichier son au format AIFF. A utiliser avec les options -c, -s, -l, ou -f.
-a, --format=alaw	Utiliser des échantillons audio a-law.
--aft-zero	Utilise zéro comme valeur initiale de l'aftertouch.
-B NUM, --hardwarebuf-samps=NUM	Nombre de trames d'échantillonnage audio maintenues dans le tampon du <i>circuit</i> CNA. C'est une limite au-dessus de laquelle l'E/S audio <i>logicielle</i> va attendre avant de retourner. Une faible valeur

réduit le délai audio d'E/S ; mais la valeur est souvent limitée par le matériel, et l'on risque des retards dans les données avec de petites valeurs. Dans le cas de la sortie portaudio (la sortie par défaut en temps réel), le paramètre -B (plus précisément -B / sr) est passé comme valeur de "latence suggérée". En dehors de cela, Csound n'a aucun contrôle sur la manière dont PortAudio interprète le paramètre. La valeur par défaut est 1024 sur Linux, 4096 sur Mac OS X et 16384 sur Windows.

-b NUM, --iobufsamps=NUM

Nombre de trames d'échantillonnage audio dans chaque tampon *logiciel* d'E/S. De grandes valeurs conviennent, mais les petites valeurs réduiront le délai d'E/S audio et amélioreront la précision temporelle des événements en temps réel. La valeur par défaut est 256 sur Linux, 1024 sur Mac OS X, et 4096 sur Windows. Lors d'une exécution en temps réel, Csound attend les E/S audio toutes les *NUM* divisions. Il effectue aussi le traitement audio (et interroge d'autres entrées comme le MIDI) toutes les *ksmps* divisions de l'orchestre. On peut synchroniser les deux. Par commodité, si NUM est négatif, la valeur effective est *ksmps* \* -NUM (audio synchrone avec les divisions de période k). Avec de petites valeurs de NUM (par exemple 1) l'interrogation devient fréquente et calée sur les divisions fixes d'échantillonnage du CNA.

Note : si l'on utilise en même temps -iadc et -odac (audio temps réel en mode duplex complet), il faut fixer l'option -b à un multiple entier de *ksmps*.

-C, --cscore

Utiliser le traitement par Cscore du fichier partition.

-c, --format=schar

Utiliser des échantillons audio en caractères signés sur 8 bit.

--csd-line-nums=NUM

Détermine comment les numéros de ligne sont comptés et affichés pour les messages d'erreur lors du traitement d'un fichier Csound Unified Document (.csd). Cette option n'a aucun effet si des fichiers d'orchestre et de partition séparés sont utilisés. (Csound 5.08 et versions ultérieures).

- 0 = les numéros de ligne sont relatifs au début des sections de l'orchestre ou de la partition du CSD.
- 1 = les numéros sont relatifs au début du fichier CSD. C'est le comportement par défaut dans Csound 5.08.

-D, --defer-gen1

Différer le chargement des fichiers sons de GEN01 jusqu'au moment de l'exécution.

-d, --nodisplays

Supprimer tous les affichages. Voir -O si vous souhaitez enregistrer le compte-rendu dans un fichier.



## Note

Cette option ne fonctionne que depuis la ligne de commande et pas depuis .csound6rc ou depuis une section *CsOptions* d'un fichier csd. Elle nécessite une prise en compte avant que le son proprement dit ne commence. Elle est ignorée dans .csound6rc et dans *CsOptions*.

<code>--daemon</code>	Exécution en mode démon : ne se termine pas si aucun CSD/orchestra n'est donné, ou s'il est vide ou s'il ne compile pas.
<code>--devices[=x]</code>	Donne une liste des périphériques audio (x=out, seulement les périphériques en sorties ; x=in, en entrée; sinon en entrée et en sortie), puis termine l'application.
<code>--displays</code>	Autoriser les affichages, inversant l'effet d'une éventuelle option <code>-d</code> précédente.
<code>--default-paths</code>	Autoriser à nouveau l'addition de répertoire de CSD/ORC/SCO aux chemins de recherche, si cette possibilité avait été désactivée par une option <code>--no-default-paths</code> précédente (par exemple dans <code>.csound6rc</code> ).
<code>--env:NOM=VALEUR</code>	Positionner la variable d'environnement NOM à VALEUR. Note : on ne peut pas positionner toutes les variables d'environnement de cette manière, car certaines d'entre elles sont lues avant l'analyse de la ligne de commande. Cette option fonctionne entre autres avec INCDIR, SADIR, <i>SFDIR</i> et <i>SSDIR</i> .
<code>--env:NOM+=VALEUR</code>	Ajouter VALEUR à la liste des chemins de recherche dont le séparateur est ';' dans la variable d'environnement NOM (ça peut-être INCDIR, SADIR, <i>SFDIR</i> ou <i>SSDIR</i> ). Si un fichier est trouvé dans plusieurs répertoires, c'est le dernier qui est utilisé.
<code>--expression-opt</code>	<p><i>Noter que cette option n'a aucun effet dans Csound 6. Seulement dans Csound 5. Activer certaines optimisations dans les expressions :</i></p> <ul style="list-style-type: none"> <li>Les affectations redondantes sont éliminées chaque fois que c'est possible. Par exemple la ligne <code>a1 = a2 + a3</code> sera compilée en <code>a1 Add a2, a3</code> au lieu de <code>#a0 Add a2, a3 a1 = #a0</code> évitant une variable temporaire et un appel d'opcode. Moins d'appels d'opcode induisent une utilisation moindre du CPU (un orchestre moyen peut être compilé 10% plus vite avec <code>--expression-opt</code>, mais cela dépend aussi largement du nombre d'expressions utilisées, du taux de contrôle (voir également ci-dessous), etc ; ainsi, la différence peut être moindre, mais aussi beaucoup plus).</li> <li>le nombre de variables temporaires de taux <code>a</code> et de taux <code>k</code> est réduit significativement. L'expression <div style="text-align: center;"> <math display="block">(a1 + a2 + a3 + a4)</math> </div> sera compilée en <div style="margin-left: 40px;"> <pre>#a0 Add a1, a2 #a0 Add #a0, a3 #a0 Add #a0, a4</pre> </div> ; (le résultat se trouve dans #a0)   au lieu de <div style="margin-left: 40px;"> <pre>#a0 Add a1, a2 #a1 Add #a0, a3</pre> </div> </li> </ul>

#a2 Add #a1, a4 ; (le résultat se trouve dans #a2)

Les avantages d'avoir moins de variables temporaires sont :

- moins de mémoire cache utilisée, ce qui peut améliorer les performances des orchestres avec beaucoup d'expressions de taux a et un faible taux de contrôle (par exemple ksmpls = 100)
- les grands orchestres sont chargés plus vite grâce au nombre moins important d'identifiants différents
- les erreurs de dépassement d'indice (par exemple quand des messages comme Case2: indx=-56004 (ffff253c); (short)indx = 9532 (253c) sont imprimés et que Csound a un comportement bizarre ou plante) peuvent être corrigées, car de telles erreurs sont provoquées par trop de noms de variable différents (spécialement au taux a) dans un seul instrument.

Noter que l'optimisation (pour des raisons techniques) n'est pas exécutée sur les i-variables temporaires.



## Avertissement

Lorsque --expression-opt est activé, il est interdit d'utiliser la fonction i() avec un argument expression, et il n'est pas prudent de compter au temps i sur la valeur de k-expressions.

-F FICHER, --midifile=FICHER	Lire les événements MIDI à partir du fichier <i>FICHER</i> . Le fichier ne doit avoir qu'une seule piste dans les versions 4.xx et antérieures de Csound ; cette limitation est levée à partir de Csound 5.00.
-f, --format=float	Utiliser des échantillons audio en format réel simple précision (non jouables sur certains systèmes, mais lisibles avec <i>-i</i> , <i>soundin</i> et <i>GENOI</i> ).
-G, --postscriptdisplay	Supprimer les graphiques, une sortie graphique PostScript est produite à la place.
-g, --asciisplay	Supprimer les graphiques, une sortie pseudo-graphique ASCII est produite à la place.
--get-system-sr	Affiche le sr du système et termine l'application, nécessite au préalable -o dac. Si l'audio ne supporte pas cette requête, -1 est retourné.
--set-system-sr	Fixe sr au sr du système, nécessite au préalable -o dac. Si l'audio ne supporte pas cette demande -1 est retourné.
-H#, --heartbeat=NUM	Imprimer un battement de cœur après chaque écriture de tampon dans le fichier son : <ul style="list-style-type: none"> <li>• pas de NUM, une barre tournante.</li> <li>• NUM = 1, une barre tournante.</li> <li>• NUM = 2, un point (.)</li> </ul>



	<ul style="list-style-type: none"> <li>• NUM = 3, la taille du fichier en secondes.</li> <li>• NUM = 4, un beep sonore.</li> </ul>
-h, --noheader	Pas d'en-tête dans le fichier son de sortie. N'écrit pas d'en-tête de fichier, seulement les échantillons binaires.
--help	Afficher un message d'aide en ligne.
-I, --i-only	<i>seulement au temps i</i> . Allouer et initialiser tous les instruments selon la partition, mais en ignorant tous les traitement de temps p (pas de k-signaux ni de a-signaux, et donc aucune amplitude et aucun son). Fournit un moyen rapide de tester la validité des p-champs de la partition et des i-variables de l'orchestre. Cette option est mutuellement exclusive avec l'option --syntax-check-only.
-i FICHIER, --input=FICHIER	Nom d'un fichier son en entrée. S'il ne s'agit pas d'un nom de chemin complet, le fichier sera d'abord cherché dans le répertoire courant, ensuite dans celui qui est donné par la variable d'environnement <i>SSDIR</i> (si elle définie), enfin par <i>SFDIR</i> . Si le nom est <i>stdin</i> , la lecture audio se fera à partir de l'entrée standard.  Les noms <i>devaudio</i> ou <i>adc</i> provoqueront l'écoute du son sur le périphérique d'entrée audio de l'hôte. Il est possible de choisir un numéro de périphérique en ajoutant un entier compris entre 0 et 1023, ou un nom de périphérique séparé par un caractère : (par exemple -iadc3, -iadc:hw:1,1). L'utilisation d'un numéro ou d'un nom de périphérique dépend de l'interface audio de l'hôte. Dans le premier cas, un nombre en-dehors de l'intervalle autorisé provoque habituellement une erreur et un affichage de la liste des numéros de périphériques valides.  Les données audio entrant grâce à -i peuvent être reçues au moyen d'opcodes tels que <i>inch</i> .
--id_artist=chaîne	(longueur max. = 200 caractères) Champ artiste dans le fichier son de sortie (pas d'espaces)
--id_comment=chaîne	(longueur max. = 200 caractères) Champ commentaire dans le fichier son de sortie (pas d'espaces)
--id_copyright=chaîne	(longueur max. = 200 caractères) Champ copyright dans le fichier son de sortie (pas d'espaces)
--id_scopyright=integer	(Depuis la version 6.05) Copyright/licence encodé par un entier dont la signification est :  0 : "Tous droits réservés" (valeur par défaut) 1 : "Creative Commons Attribution-NonCommercial-NoDerivatives (CC BY-NC-ND)" 2 : "Creative Commons Attribution-NonCommercial-ShareAlike (CC BY-NC-SA)" 3 : "Creative Commons Attribution-NonCommercial (CC BY-NC)" 4 : "Creative Commons Attribution-NoDerivatives (CC BY-ND)"

	5 : "Creative Commons Attribution-ShareAlike (CC BY-SA)" 6 : "Creative Commons Attribution-ShareAlike (CC BY)" 7 : "Sous licence BSD"
--id_date=chaîne	(longueur max. = 200 caractères) Champ date dans le fichier son de sortie (pas d'espaces)
--id_software=chaîne	(longueur max. = 200 caractères) Champ logiciel dans le fichier son de sortie (pas d'espaces)
--id_title=chaîne	(longueur max. = 200 caractères) Champ titre dans le fichier son de sortie (pas d'espaces)
--ignore_csopts=entier	S'il vaut 1, Csound ignorera toutes les options spécifiées dans la section CsOptions du fichier csd. Voir <i>Format de Fichier Unifié pour les Orchestres et les Partitions</i> .
--input_stream=chaîne	Nom du flot d'entrée pulseaudio.
-J, --ircam, --format=ircam	Ecrire un fichier son dans le format de l'IRCAM.
-j NUM	NUM processus sont rendus disponibles pour l'exécution. Ce n'est avantageux que si le nombre de processeurs de l'ordinateur est supérieur ou égal au nombre de processus demandés. Ca peut aussi ralentir l'exécution si <i>ksmps</i> est trop petit.
--jack_client=[nom_client]	Le nom de client utilisé par Csound, par défaut 'csound5'. Si plusieurs instances de Csound se connectent au serveur JACK, il faut utiliser différents noms de client pour éviter les conflits. (Linux et Mac OS X seulement)
--jack_inportname=[préfixe du nom du port d'entrée], --jack_outportname=[préfixe du nom du port de sortie]	Préfixe du nom des ports JACK d'entrée/sortie de Csound ; la valeur par défaut est 'input' et 'output'. Le nom de port réel est le numéro de canal ajouté au préfixe du nom. (Linux et Mac OS X seulement)  Exemple : avec les réglages par défaut ci-dessus, un orchestre stéréo créera ces ports dans une opération en full duplex :
	<pre> csound5:input1      (enregistrement gauche) csound5:input2      (enregistrement droite) csound5:output1     (reproduction gauche) csound5:output2     (reproduction droite) </pre>
-K, --nopeaks	Ne générer aucun bloc PEAK.
-k NUM, --control-rate=NUM	Remplacer le taux de contrôle ( <i>kr</i> ) fourni par l'orchestre.
-L PERIPHERIQUE, --score-in=PERIPHERIQUE	Lire en temps réel des événements de partition en ligne de texte à partir du périphérique PERIPHERIQUE. Le nom <i>stdin</i> permettra de recevoir les événements de partition de votre terminal, ou d'un autre processus via un tube de communication (pipe). Chaque ligne d'évènement est terminée par un retour chariot. Les événements sont codés de la même manière que ceux de la <i>partition numérique standard</i> , sauf qu'un événement avec p2=0 sera exécuté immédiatement, et qu'un événement avec p2=T sera exécuté T secondes après son arrivée. Les événements peuvent arriver n'importe quand et dans n'importe quel ordre. La fonction <i>carry</i> ( <i>report de</i>

*valeur*) de la partition est autorisée ici, ainsi que les notes liées (p3 négatif) at les arguments chaîne, mais les pentes d'interpolation et les références *pp* ou *np* ne le sont pas.



## Note

L'option -L n'est valide que sur les système \*NIX qui ont des tuyaux. Elle ne fonctionne pas sous Windows.

-l, --format=long

Utiliser des échantillons audio codés en entiers longs.

-M PERIPHERIQUE, --midi-device=PERIPHERIQUE

Lire les événements MIDI à partir du périphérique *PERIPHERIQUE*. Si l'on utilise ALSA MIDI (-+rtmidi=alsa), les périphériques sont sélectionnés par leur nom et pas par un numéro. Ainsi, il faut utiliser une option comme -M hw:CARTE,PERIPHERIQUE où CARTE et PERIPHERIQUE sont les numéros de la carte et du périphérique (par exemple -M hw:1,0). Dans le cas de CoreMidi et de MME, PERIPHERIQUE doit être un nombre, et s'il est en-dehors de l'intervalle permis, une erreur est levée et les numéros de périphérique valides sont imprimés. Avec PortMidi, PERIPHERIQUE est soit un nombre pour une entrée sur un seul port, soit 'a' pour écouter sur tous les ports, ou bien 'm' pour associer les ports à des canaux MIDI au-delà de 16. Dans ce cas, le périphérique 0 utilise 1-16, le périphérique 1 utilise 17-32, ... Les options 'a' et 'm' conviennent aussi lorsqu'il n'y a pas de périphérique car aucune erreur n'est générée.

-m NUM, --messagelevel=NUM

Niveau des messages pour la sortie standard (terminal). Prend la *somme* de n'importe lesquelles de ces valeurs :

- 1 = messages d'amplitude de note
  - 2 = message d'échantillons hors intervalle
  - 4 = messages d'avertissement
  - 128 = impression d'information de tests de référence
  - 1024 = suppression des messages d'obsolescence
- Et exactement un de ceux-ci pour choisir le format de l'amplitude des notes :
- 0 = amplitudes brutes, pas de couleur
  - 32 = dB, pas de couleur
  - 64 = dB, hors intervalle colorées en rouge
  - 96 = dB, toutes colorées
  - 256 = brutes, hors intervalle colorées en rouge
  - 512 = brutes, toutes colorées

La valeur par défaut est 135 (128+4+2+1), ce qui signifie tous les messages, valeurs d'amplitude brutes, et impression du temps écoulé à la fin de l'exécution. La mise en couleur des amplitudes brutes

	<p>fut introduite dans la version 5.04. La suppression des messages d'obsolescence a été ajoutée dans la version 6.14.</p>
--m-amps=NUM	<p>Niveau des messages d'amplitudes sur la sortie standard (terminal).</p> <ul style="list-style-type: none"> <li>• 0 = pas de messages d'amplitude de note</li> <li>• 1 = messages d'amplitude de note</li> </ul>
--m-range=NUM	<p>Niveau des messages de dépassement de limite sur la sortie standard (terminal).</p> <ul style="list-style-type: none"> <li>• 0 = aucun message d'échantillon hors limites</li> <li>• 1 = messages d'échantillons hors limites</li> </ul>
--m-warnings=NUM	<p>Niveau des messages d'avertissement sur la sortie standard (terminal).</p> <ul style="list-style-type: none"> <li>• 0 = pas de messages d'avertissement</li> <li>• 1 = messages d'avertissement</li> </ul>
--m-dB=NUM	<p>Niveau des messages pour le format d'amplitude sur la sortie standard (terminal).</p> <ul style="list-style-type: none"> <li>• 0 = messages d'amplitude absolue</li> <li>• 1 = messages d'amplitude en dB</li> </ul>
--m-colours=NUM	<p>Niveau des messages pour le format d'amplitude sur la sortie standard (terminal).</p> <ul style="list-style-type: none"> <li>• 0 = pas de coloration des messages d'amplitude</li> <li>• 1 = coloration des messages d'amplitude</li> </ul>
--m-benchmarks=NUM	<p>Niveau des messages d'information de test de performance sur la sortie standard (terminal).</p> <ul style="list-style-type: none"> <li>• 0 = pas de nombres de test de performance</li> <li>• 1 = nombres de test de performance</li> </ul>
++max_str_len=entier	<p>(min: 10, max: 10000) Longueur maximale des variables chaîne + 1 ; la valeur par défaut est 256 autorisant une longueur de 255 caractères. La longueur des constantes chaîne n'est pas limitée par ce paramètre.</p>
--midi-devices[=x]	<p>Donne une liste des périphériques midi (x=out, seulement les périphériques en sortie ; x=in, en entrée ; sinon en entrée et en sortie), puis termine l'application.</p>
--midi-key=N	<p>Transmettre le numéro de touche d'un message MIDI note on au p-champ N en valeur MIDI [0-127].</p>
--midi-key-cps=N	<p>Transmettre le numéro de touche d'un message MIDI note on au p-champ N en cycles par seconde.</p>

<code>--midi-key-oct=N</code>	Transmettre le numéro de touche d'un message MIDI note on au p-champ N en octave linéaire.
<code>--midi-key-pch=N</code>	Transmettre le numéro de touche d'un message MIDI note on au p-champ N en oct.pch (classe de hauteur).
<code>--midi-velocity=N</code>	Transmettre la vélocité d'un message MIDI note on au p-champ N en valeur MIDI [0-127].
<code>--midi-velocity-amp=N</code>	Transmettre la vélocité d'un message MIDI note on au p-champ N en amplitude [0-0dbfs].
<code>--midioutfile=NOMFICHIER</code>	Sauvegarder la sortie MIDI dans un fichier (seulement à partir de Csound 5.00).
<code>++msg_color=booléen</code>	Activer les attributs de message (couleurs etc.) ; il peut être nécessaire de les désactiver sur certains terminaux qui impriment des caractères étranges au lieu de modifier les attributs du texte. Par défaut : true.
<code>++mute_tracks=chaîne</code>	(longueur max. = 255 caractères) Ignorer les évènements (autres que les changements de tempo) dans les pistes de fichier MIDI, définies par un motif binaire (par exemple, <code>++mute_tracks=00101</code> désactivera la troisième et la cinquième pistes).
<code>-N, --notify</code>	Avertir (par un beep) quand la partition ou la piste MIDI est terminée.
<code>-n, --nosound</code>	Pas de son. Faire tous les traitements, mais ne pas écrire de son sur le disque. Cette option ne change rien d'autre dans l'exécution.
<code>--num-threads=NUM</code>	<i>NUM</i> processus sont rendus disponibles pour l'exécution. Ce n'est avantageux que si le nombre de processeurs de l'ordinateur est supérieur au nombre de processus demandés. Ca peut aussi ralentir l'exécution si <i>ksmps</i> est trop petit.
<code>--no-default-paths</code>	Désactiver l'addition de répertoire de CSD/ORC/SCO au chemin de recherche.
<code>--no-expression-opt</code>	Désactiver l'optimisation des expressions.
<code>-O FICHIER, --logfile=FICHIER</code>	Compte-rendu dans le fichier <i>FICHIER</i> . Si <i>FICHIER</i> est null (c-à-d <code>-O null</code> ou <code>--logfile=null</code> ) toutes les impressions de message sur la console sont désactivées.



### Note

Cette option ne fonctionne que depuis la ligne de commande et pas depuis `.csound6rc` ou depuis une section *CsOptions* d'un fichier `csd`. Elle nécessite une prise en compte avant que le son proprement dit ne commence. Elle est ignorée dans `.csound6rc` et dans *CsOptions*.

<code>-o FICHIER, --output=FICHIER</code>	Nom du fichier son de sortie. Si ce n'est pas un nom de chemin complet, le fichier son sera placé dans le répertoire donné par la variable d'environnement <i>SFDIR</i> (si elle est définie), sinon dans le répertoire courant. Le nom <i>stdout</i> provoque l'écriture audio sur la sortie stan-
---	---

dard, tandis qu'avec *null* il n'y a aucun son en sortie comme pour l'option -n. Si aucun nom n'est donné, le nom par défaut sera *test*.

Les noms *devaudio* ou *dac* (on peut utiliser *-odac* ou *-o dac*) provoquent l'écriture du son sur le périphérique de sortie son de l'hôte. Il est possible de choisir un numéro de périphérique en ajoutant une valeur entière dans l'intervalle 0 à 1023, ou un nom de périphérique séparé par un caractère : (par exemple *-odac3*, *-odac:hw:1,1*). Selon l'interface audio de l'hôte on emploiera un numéro de périphérique ou un nom. Dans le premier cas, un nombre hors de l'intervalle lève habituellement une erreur et affiche la liste des numéros de périphérique valides.

<code>--ogg</code>	Fixe le format du fichier de sortie à ogg. (Csound 5.18 et ultérieur)
<code>--omacro:XXX=YYY</code>	Donner la valeur YYY à la macro d'orchestre XXX
<code>--opcode-lib=NOMBIB</code>	Charge la bibliothèque de greffon <i>NOMBIB</i> .
<code>--orc nomorc</code>	L'argument est le fichier d'orchestre. Utilisé lorsque l'on a pas besoin de partition. (Csound 5.18 et ultérieur).
<code>--ksmps=N</code>	Force ksmps à N (versions 6.05 et supérieures).
<code>--output_stream=chaîne</code>	Nom du flot de sortie pulseaudio.
<code>--port=N</code>	Détermine un port UDP pour recevoir des commandes et/ou du code d'instruments ou d'orchestre (implique <code>--daemon</code> ). Voir <i>serveur UDP</i> .
<code>--udp-echo</code>	Active l'affichage des commandes UDP sur le terminal. Tout message reçu par le serveur UDP est affiché en écho (que ce soit une commande valide ou non).
<code>--udp-console=address:port</code>	Redirige les messages de la console vers une adresse:port distante via UDP.
<code>--udp-mirror-console=address:port</code>	Recopie les messages de la console vers une adresse:port distante via UDP.
<code>-Q PERIPHERIQUE</code>	<p>Activer les opérations MIDI OUT vers le périphérique d'id <i>PERIPHERIQUE</i>. Cette option permet l'exécution en parallèle sur MIDI OUT et CNA. Malheureusement le séquençement temps réel implémenté dans Csound est complètement géré par le flot d'échantillons du tampon du CNA. C'est pourquoi les opérations MIDI OUT peuvent présenter quelques irrégularités dans le temps. On peut réduire ces irrégularités en utilisant une valeur plus faible pour l'option <i>-b</i>.</p> <p>Si l'on utilise ALSA MIDI (<code>--rtmidi=alsa</code>), les périphériques sont sélectionnés par leur nom et non par un numéro. Il faut alors utiliser une option comme <code>-Q hw:CARTE,PERIPHERIQUE</code> où CARTE et PERIPHERIQUE sont les numéros de la carte et du périphérique (par exemple <code>-Q hw:1,0</code>). Dans le cas de PortMidi et de MME, PERIPHERIQUE doit être un nombre, et s'il est hors intervalle, une erreur est levée et les numéros de périphérique valides sont imprimés.</p>

-R, --rewrite	Réécrire continuellement l'en-tête pendant l'écriture du fichier son (WAV/AIFF).
-r NUM, --sample-rate=NUM	Remplacer le taux d'échantillonnage ( <i>sr</i> ) fourni par l'orchestre.
--raw_controller_mode=booléen	Désactiver le traitement spécial des contrôleurs MIDI tels que sustain, pédale, all notes off, etc., autorisant l'utilisation des 128 contrôleurs pour n'importe quelle fonction. Cela initialise également la valeur de tous les contrôleurs à zéro. Valeur par défaut : no.
--realtime	Le mode temps réel prioritaire est activé avec les effets suivants : <ol style="list-style-type: none"> <li>1. tout opcode de lecture/écriture audio dans un fichier est géré de manière asynchrone par un fil d'exécution séparé.</li> <li>2. toutes les opérations d'initialisation sont aussi exécutées de manière asynchrone.</li> </ol>
--rtaudio=chaîne	(longueur max. = 20 caractères) Nom du module audio temps réel. La valeur par défaut est PortAudio. Sont disponibles selon la plateforme et les options de construction : Linux : alsa, jack; Windows : mme; Mac OS X : CoreAudio. De plus, on peut utiliser null sur toutes les plates-formes, afin d'interdire l'utilisation de tout greffon audio temps réel.
--rtmidi=chaîne	(longueur max. = 20 caractères) Nom du module MIDI temps réel. La valeur par défaut est PortMidi ; autres options (en fonction des options de construction) : Linux : alsa; Windows : mme, winmm. De plus, on peut utiliser null sur toutes les plates-formes, afin d'interdire l'utilisation de tout greffon MIDI temps réel.  Les périphériques ALSA MIDI sont sélectionnés par leur nom au lieu d'un numéro. Aussi, il faut utiliser une option comme -M hw:CARTE,PERIPHERIQUE où CARTE et PERIPHERIQUE sont les numéros de la carte et du périphérique (par exemple -M hw:1,0).
-s, --format=short	Utiliser des échantillons audio codés par des entiers courts.
--sample-accurate	Démarré et arrêté les instances d'instruments à l'échantillon le plus proche de l'instant demandé. Ceci diffère du comportement traditionnel de Csound qui arrondi le temps au cycle-k le plus proche. Noter que cela ne fonctionne pas avec les notes liées.
--sched	<i>Seulement sur linux.</i> Utiliser pour le temps réel le temps-partagé et le verrouillage de la mémoire. (nécessite également -d et -o dac ou -o devaudio). Voir aussi --sched=N ci-dessous.
--sched=N	<i>Seulement sur linux.</i> Identique à --sched, mais permet de spécifier une valeur de priorité: si N est positif (dans l'intervalle 1 à 99) la politique de temps-partagé SCHED_RR sera utilisée avec une priorité de N ; autrement, SCHED_OTHER est utilisée avec le niveau "de gentillesse" (nice) à N. On peut aussi l'utiliser avec le format --sched=N,MAXCPU,TEMPS pour autoriser l'utilisation d'un processus léger (thread) de contrôle qui terminera Csound si le temps moyen d'utilisation de CPU dépasse MAXCPU pourcents sur une durée de TEMPS secondes (à partir de Csound 5.00).

<code>--server=chaîne</code>	Nom du serveur pulseaudio.
<code>--simple-sorted-score</code>	Conserve le fichier de partition trié <i>score.srt</i> dans un format simple et lisible lors de la sortie. (Nouveau dans la version 6.14).
<code>--skip_seconds=float</code>	(min: 0) Commencer la reproduction au temps indiqué (en secondes), en ignorant les événements antérieurs de la partition ou du fichier MIDI.
<code>--smacro:XXX=YYY</code>	Donner la valeur <i>YYY</i> à la macro de partition <i>XXX</i>
<code>--strset</code>	<i>Csound 5.</i> L'option <code>--strset</code> permet de passer des chaînes à <code>strset</code> pour les lier à des valeurs numériques depuis la ligne de commande, dans le format ' <code>--strsetN=VALEUR</code> '. Utile pour passer des paramètres à l'orchestre (par exemple des noms de fichier).
<code>--syntax-check-only</code>	Provoque l'arrêt de Csound immédiatement après que les parseurs de l'orchestre et de la partition ont fini la vérification de la syntaxe des fichiers d'entrée et avant que l'orchestre n'exécute la partition. Cette option est mutuellement exclusive avec l'option <code>--i-only</code> . (Csound 5.08 et versions ultérieures).
<code>-T, --terminate-on-midi</code>	Terminer l'exécution quand la fin du fichier MIDI est atteinte.
<code>-t0, --keep-sorted-score</code>	Empêcher Csound d'effacer le fichier de la partition triée, <i>score.srt</i> , lors de la sortie.
<code>-t NUM, --tempo=NUM</code>	Utiliser les pulsations non interprétées de <i>score.srt</i> pour cette exécution, et fixer le tempo initial à <i>NUM</i> pulsations par minute. Quand ce drapeau est positionné, le tempo de l'exécution de la partition est aussi contrôlable depuis l'orchestre. ATTENTION : ce mode d'opération est expérimental et n'est pas forcément fiable.
<code>-U UTILITE, --utility=UTILITE</code>	Invoquer le programme utilitaire <i>UTILITE</i> . En donnant un nom invalide on obtient une liste des utilitaires.
<code>-u, --format=ulaw</code>	Utiliser des échantillons audio u-law.
<code>--vbr-quality=X</code>	Fixe une qualité de débit binaire variable pour la sortie ogg. (Csound 6.03 et ultérieur).
<code>-v, --verbose</code>	Traduction et exécution détaillées. Imprime les détails de la traduction de l'orchestre et de son exécution, permettant une localisation plus précise des erreurs.
<code>--version</code>	Afficher les informations de version.
<code>-W, --wave, --format=wave</code>	Ecrire un fichier son au format WAV.
<code>-x FICHIER, --extract-score=FICHIER</code>	Extraire un morceau de la partition triée, <i>score.srt</i> , en utilisant le fichier d'extraction <i>FICHIER</i> (voir <i>Extract</i> ).
<code>-Z, --dither</code>	Activer le dithering pour la conversion audio du format interne en virgule flottante vers un format 32, 16 ou 8 bit. La forme d'excitation par défaut est triangulaire.
<code>-Z, --dither--triangular, --dither--uniform</code>	Activer le dithering pour la conversion audio du format interne en virgule flottante vers un format 32, 16 ou 8 bit. Dans le cas de <code>-Z</code> le



chiffre qui suit doit être 1 (pour triangulaire) ou 2 (pour uniforme). L'interprétation exacte dépend du système de sortie.

`-z NUM, --list-opcodesNUM`

Lister les opcodes de cette version :

- pas de NUM, montrer seulement les noms
- NUM = 0, montrer seulement les noms
- NUM = 1, montrer les arguments de chaque opcode dans le format `<nomop> <argssortie> <argsentrée>`
- NUM = 2, montrer les noms y compris ceux qui sont obsolètes
- NUM = 3, montrer les arguments pour chaque opcode, y compris pour ceux qui sont obsolètes, en utilisant le format `<opname> <outargs> <inargs>`

## Options de ligne de commande (par catégorie)

Ci-dessous la liste par catégorie des options de ligne de commande disponibles dans Csound 5. Les implémentations sur différentes plates-formes peuvent ne pas réagir de la même façon à certaines options !

On peut consulter les options de ligne de commande par ordre alphabétique dans la section *Options de Ligne de Commande (par Ordre Alphabétique)*.

Le format d'une commande est soit :

```
csound [options] [nomorchestre] [nompartition]
soit
```

```
csound [options] [nomfichiercsd]
```

où les arguments sont de 2 sortes : arguments *options* (commençant par « - », « -- » ou « + »), et arguments *nom* (tels que noms de fichier). Certains arguments options sont suivis d'un nom ou d'un argument numérique. Les options qui commencent par « -- » et « + » prennent habituellement un argument précédé du signe « = ».

### Sortie dans un fichier audio

`-3, --format=24bit`

Utiliser des échantillons audio de 24 bit.

`-8, --format=uchar`

Utiliser des échantillons audio en caractères non-signés sur 8 bit.

`-A, --aiff, --format=aiff`

Ecrire un fichier son au format AIFF. A utiliser avec les options `-c`, `-s`, `-l`, ou `-f`.

`-a, --format=alaw`

Utiliser des échantillons audio a-law.

`-c, --format=schar`

Utiliser des échantillons audio en caractères signés sur 8 bit.

`-f, --format=float`

Utiliser des échantillons audio en format réel simple précision (non jouables sur certains systèmes, mais lisibles avec `-i`, *soundin* et *GEN01*).

`--format=type`

Choisir le format du fichier de sortie audio parmi les formats disponibles dans libsndfile. Actuellement la liste est aiff, au, avr, caf,

	<p>flac, htk, ircam, mat4, mat5, MPC, nist, ogg, paf, pvf, raw, sd2, sds, svx, voc, w64, W64, wav, wavex, WVE, xi. On peut aussi écrire --format=type:format ou --format=format:type pour fixer le type du fichier (wav, aiff, etc.) et le format d'échantillonnage (short, long, float, etc.) en même temps.</p>
-h, --noheader	<p>Pas d'en-tête dans le fichier son de sortie. N'écrit pas d'en-tête de fichier, seulement les échantillons binaires.</p>
-i FICHIER, --input=FICHIER	<p>Nom d'un fichier son en entrée. S'il ne s'agit pas d'un nom de chemin complet, le fichier sera d'abord cherché dans le répertoire courant, ensuite dans celui qui est donné par la variable d'environnement <i>SSDIR</i> (si elle définie), enfin par <i>SFDIR</i>. Si le nom est <i>stdin</i>, la lecture audio se fera à partir de l'entrée standard.</p> <p>Les noms <i>devaudio</i> ou <i>adc</i> provoqueront l'écoute du son sur le périphérique d'entrée audio de l'hôte. Il est possible de choisir un numéro de périphérique en ajoutant un entier compris entre 0 et 1023, ou un nom de périphérique séparé par un caractère : . L'utilisation d'un numéro ou d'un nom de périphérique dépend de l'interface audio de l'hôte. Dans le premier cas, un nombre en-dehors de l'intervalle autorisé provoque habituellement une erreur et un affichage de la liste des numéros de périphérique valides.</p> <p>Les données audio entrant grâce à -i peuvent être reçues au moyen d'opcodes tels que <i>inch</i>.</p>
-J, --ircam, --format=ircam	<p>Ecrire un fichier son dans le format de l'IRCAM.</p>
-K, --nopeaks	<p>Ne générer aucun bloc PEAK.</p>
-l, --format=long	<p>Utiliser des échantillons audio codés en entiers longs.</p>
-n, --nosound	<p>Pas de son. Faire tous les traitements, mais ne pas écrire de son sur le disque. Cette option ne change rien d'autre dans l'exécution.</p>
-o FICHIER, --output=FICHIER	<p>Nom du fichier son de sortie. Si ce n'est pas un nom de chemin complet, le fichier son sera placé dans le répertoire donné par la variable d'environnement <i>SFDIR</i> (si elle est définie), sinon dans le répertoire courant. Le nom <i>stdout</i> provoque l'écriture audio sur la sortie standard, tandis qu'avec <i>null</i> il n'y a aucun son en sortie comme pour l'option -n. Si aucun nom n'est donné, le nom par défaut sera <i>test</i>.</p> <p>Les noms <i>dac</i> ou <i>devaudio</i> (on peut utiliser <i>odac</i> ou <i>-o dac</i>) provoquent l'écriture du son sur le périphérique de sortie son de l'hôte. Il est possible de choisir un numéro de périphérique en ajoutant une valeur entière dans l'intervalle 0 à 1023, ou un nom de périphérique séparé par un caractère : . Selon l'interface audio de l'hôte on emploiera un numéro de périphérique ou un nom. Dans le premier cas, un nombre hors de l'intervalle lève habituellement une erreur et affiche la liste des numéros de périphérique valides.</p>
--ogg	<p>Fixe le format du fichier de sortie à ogg. (Csound 5.18 et ultérieur)</p>
--vbr-quality=X	<p>Fixe une qualité de débit binaire variable pour la sortie ogg. (Csound 6.03 et ultérieur).</p>

-R, --rewrite	Réécrire continuellement l'en-tête pendant l'écriture du fichier son (WAV/AIFF).
-s, --format=short	Utiliser des échantillons audio codés par des entiers courts.
-u, --format=ulaw	Utiliser des échantillons audio u-law.
-W, --wave, --format=wave	Ecrire un fichier son au format WAV.
-Z, --dither	Activer le dithering pour la conversion audio du format interne en virgule flottante vers un format 32, 16 ou 8 bit. La forme d'excitation par défaut est triangulaire.
-Z, --dither--triangular, --dither--uniform	Activer le dithering pour la conversion audio du format interne en virgule flottante vers un format 32, 16 ou 8 bit. Dans le cas de -Z le chiffre qui suit doit être 1 (pour triangulaire) ou 2 (pour uniforme). L'interprétation exacte dépend du système de sortie.

### Champs du fichier de sortie

--id_artist=chaîne	(longueur max. = 200 caractères) Champ artiste dans le fichier son de sortie (pas d'espaces)
--id_comment=chaîne	(longueur max. = 200 caractères) Champ commentaire dans le fichier son de sortie (pas d'espaces)
--id_copyright=chaîne	(longueur max. = 200 caractères) Champ copyright dans le fichier son de sortie (pas d'espaces)
--id_scopyright=integer	(Depuis la version 6.05) Copyright/licence encodé par un entier dont la signification est :  0 : "Tous droits réservés" (valeur par défaut) 1 : "Creative Commons Attribution-NonCommercial-NoDerivatives (CC BY-NC-ND)" 2 : "Creative Commons Attribution-NonCommercial-ShareAlike (CC BY-NC-SA)" 3 : "Creative Commons Attribution-NonCommercial (CC BY-NC)" 4 : "Creative Commons Attribution-NoDerivatives (CC BY-ND)" 5 : "Creative Commons Attribution-ShareAlike (CC BY-SA)" 6 : "Creative Commons Attribution-ShareAlike (CC BY)" 7 : "Sous licence BSD"
--id_date=chaîne	(longueur max. = 200 caractères) Champ date dans le fichier son de sortie (pas d'espaces)
--id_software=chaîne	(longueur max. = 200 caractères) Champ logiciel dans le fichier son de sortie (pas d'espaces)
--id_title=chaîne	(longueur max. = 200 caractères) Champ titre dans le fichier son de sortie (pas d'espaces)

### Entrée/sortie audio en temps réel

-i adc[PERIPHERIQUE], --input=adc[PERIPHERIQUE]	Les noms <i>devaudio</i> ou <i>adc</i> provoqueront l'écoute du son sur le périphérique d'entrée audio de l'hôte. Il est possible de choisir un numéro de périphérique en ajoutant un entier compris entre 0 et 1023,
---	---

	ou un nom de périphérique séparé par un caractère : (par exemple -iadc3, -iadc:hw:1,1). L'utilisation d'un numéro ou d'un nom de périphérique dépend de l'interface audio de l'hôte. Dans le premier cas, un nombre en-dehors de l'intervalle autorisé provoque habituellement une erreur et un affichage de la liste des numéros de périphérique valides.
-o dac[PERIPHERIQUE], --out-put=dac[PERIPHERIQUE]	Les noms <i>dac</i> ou <i>devaudio</i> (on peut utiliser <i>-odac</i> ou <i>-o dac</i> ) provoquent l'écriture du son sur le périphérique de sortie son de l'hôte. Il est possible de choisir un numéro de périphérique en ajoutant une valeur entière dans l'intervalle 0 à 1023, ou un nom de périphérique séparé par un caractère : (par exemple -odac3, -odac:hw:1,1). Selon l'interface audio de l'hôte on emploiera un numéro de périphérique ou un nom. Dans le premier cas, un nombre hors de l'intervalle lève habituellement une erreur et affiche la liste des numéros de périphérique valides.
--rtaudio=chaîne	(longueur max. = 20 caractères) Nom du module audio temps réel. La valeur par défaut est PortAudio (sur toutes les plates-formes). Sont également disponibles selon la plate-forme et les options de construction : Linux : alsa, jack; Windows : mme; Mac OS X : CoreAudio. De plus, on peut utiliser null sur toutes les plates-formes, afin d'interdire l'utilisation de tout greffon audio temps réel.
--set-system-sr	Fixe sr au sr du système, nécessite au préalable -o dac. Si l'audio ne supporte pas cette demande -l est retourné.
--realtime	Le mode temps réel prioritaire est activé avec les effets suivants : <ol style="list-style-type: none"> <li>1. tout opcode de lecture/écriture audio dans un fichier est géré de manière asynchrone par un fil d'exécution séparé.</li> <li>2. toutes les opérations d'initialisation sont aussi exécutées de manière asynchrone.</li> </ol>
++server=chaîne	Nom du serveur pulseaudio.
++output_stream=chaîne	Nom du flot de sortie pulseaudio.
++input_stream=chaîne	Nom du flot d'entrée pulseaudio.
++jack_client=[nom_client]	Le nom de client utilisé par Csound, par défaut 'csound5'. Si plusieurs instances de Csound se connectent au serveur JACK, il faut utiliser différents noms de client pour éviter les conflits. (Linux et Mac OS X seulement)
++jack_inportname=[préfixe du nom du port d'entrée], ++jack_outportname=[préfixe du nom du port de sortie]	Préfixe du nom des ports JACK d'entrée/sortie de Csound ; la valeur par défaut est 'input' et 'output'. Le nom de port réel est le numéro de canal ajouté au préfixe du nom. (Linux et Mac OS X seulement)  Exemple : avec les réglages par défaut ci-dessus, un orchestre stéréo créera ces ports dans une opération en full duplex :
	<div> csound5:input1 csound5:input2 csound5:output1 csound5:output2 </div> <div> (enregistrement gauche) (enregistrement droite) (reproduction gauche) (reproduction droite) </div>

## Entrée/sortie par fichier MIDI

<code>--aft-zero</code>	Utilise zéro comme valeur initiale de l'aftertouch.
<code>-F FICHIER, --midifile=FICHIER</code>	Lire les événements MIDI à partir du fichier <i>FICHIER</i> . Le fichier ne doit avoir qu'une seule piste dans les versions 4.xx et antérieures de Csound ; cette limitation est levée à partir de Csound 5.00.
<code>--midioutfile=NOMFICHIER</code>	Sauvegarder la sortie MIDI dans un fichier (seulement à partir de Csound 5.00).
<code>++mute_tracks=chaîne</code>	(longueur max. = 255 caractères) Ignorer les événements (autres que les changements de tempo) dans les pistes de fichier MIDI, définies par un motif binaire (par exemple, <code>++mute_tracks=00101</code> désactivera la troisième et la cinquième pistes).
<code>++raw_controller_mode=booléen</code>	Désactiver le traitement spécial des contrôleurs MIDI tels que pédale de sustain, all notes off, etc., autorisant l'utilisation des 128 contrôleurs pour n'importe quelle fonction. Cela initialise également la valeur de tous les contrôleurs à zéro. Valeur par défaut : no.
<code>++skip_seconds=float</code>	(min: 0) Commencer la reproduction au temps indiqué (en secondes), en ignorant les événements antérieurs de la partition ou du fichier MIDI.
<code>-T, --terminate-on-midi</code>	Terminer l'exécution quand la fin du fichier MIDI est atteinte.

## Entrée/sortie MIDI en temps réel

<code>--midi-devices[=x]</code>	Donne une liste des périphériques midi (x=out, seulement les périphériques en sortie ; x=in, en entrée ; sinon en entrée et en sortie), puis termine l'application.
<code>-M PERIPHERIQUE, --midi-device=PERIPHERIQUE</code>	Lire les événements MIDI à partir du périphérique <i>PERIPHERIQUE</i> . Si l'on utilise ALSA MIDI ( <code>++rtmidi=alsa</code> ), les périphériques sont sélectionnés par leur nom et pas par un numéro. Ainsi, il faut utiliser une option comme <code>-M hw:CARTE,PERIPHERIQUE</code> où CARTE et PERIPHERIQUE sont les numéros de la carte et du périphérique (par exemple <code>-M hw:1,0</code> ). Dans le cas de CoreMidi et de MME, PERIPHERIQUE doit être un nombre, et s'il est en-dehors de l'intervalle permis, une erreur est levée et les numéros de périphérique valides sont imprimés. Avec PortMidi, PERIPHERIQUE est soit un nombre pour une entrée sur un seul port, soit 'a' pour écouter sur tous les ports, ou bien 'm' pour associer les ports à des canaux MIDI au-delà de 16. Dans ce cas, le périphérique 0 utilise 1-16, le périphérique 1 utilise 17-32, ... Les options 'a' et 'm' conviennent aussi lorsqu'il n'y a pas de périphérique car aucune erreur n'est générée.
<code>--midi-key=N</code>	Transmettre le numéro de touche d'un message MIDI note on au p-champ N en valeur MIDI [0-127].
<code>--midi-key-cps=N</code>	Transmettre le numéro de touche d'un message MIDI note on au p-champ N en cycles par seconde.
<code>--midi-key-oct=N</code>	Transmettre le numéro de touche d'un message MIDI note on au p-champ N en octave linéaire.

<code>--midi-key-pch=N</code>	Transmettre le numéro de touche d'un message MIDI note on au p-champ N en oct.pch (classe de hauteur).
<code>--midi-velocity=N</code>	Transmettre la vélocité d'un message MIDI note on au p-champ N en valeur MIDI [0-127].
<code>--midi-velocity-amp=N</code>	Transmettre la vélocité d'un message MIDI note on au p-champ N en amplitude [0-0dbfs].
<code>--midioutfile=NOMFICHIER</code>	Sauvegarder la sortie MIDI dans un fichier (seulement à partir de Csound 5.00).
<code>--rtmidi=chaîne</code>	(longueur max. = 20 caractères) Nom du module MIDI temps réel. La valeur par défaut est PortMidi ; autres options (en fonction des options de construction) : Linux : alsa; Windows : mme, winmm. De plus, on peut utiliser null sur toutes les plates-formes, afin d'interdire l'utilisation de tout greffon MIDI temps réel.  Les périphériques ALSA MIDI sont sélectionnés par leur nom au lieu d'un numéro. Aussi, il faut utiliser une option comme -M hw:CARTE,PERIPHERIQUE où CARTE et PERIPHERIQUE sont les numéros de la carte et du périphérique (par exemple -M hw:1,0).
<code>-Q PERIPHERIQUE</code>	Activer les opérations MIDI OUT vers le périphérique d'id <i>PERIPHERIQUE</i> . Cette option permet l'exécution en parallèle sur MIDI OUT et CNA. Malheureusement le séquençement temps réel implémenté dans Csound est complètement géré par le flot d'échantillons du tampon du CNA. C'est pourquoi les opérations MIDI OUT peuvent présenter quelques irrégularités dans le temps. On peut réduire ces irrégularités en utilisant une valeur plus faible pour l'option <i>-b</i> .  Si l'on utilise ALSA MIDI ( <code>--rtmidi=alsa</code> ), les périphériques sont sélectionnés par leur nom et non par un numéro. Il faut alors utiliser une option comme <code>-Q hw:CARTE,PERIPHERIQUE</code> où CARTE et PERIPHERIQUE sont les numéros de la carte et du périphérique (par exemple <code>-Q hw:1,0</code> ). Dans le cas de PortMidi et de MME, PERIPHERIQUE doit être un nombre, et s'il est hors intervalle, une erreur est levée et les numéros de périphérique valides sont imprimés.

## Affichage

<code>--csd-line-nums=NUM</code>	Détermine comment les numéro de ligne sont comptés et affichés pour les messages d'erreur lors du traitement d'un fichier Csound Unified Document (.csd). Cette option n'a aucun effet si des fichiers d'orchestre et de partition séparés sont utilisés. (Csound 5.08 et versions ultérieures). <ul style="list-style-type: none"> <li>• 0 = les numéros de ligne sont relatifs au début des sections de l'orchestre ou de la partition du CSD.</li> <li>• 1 = les numéros sont relatifs au début du fichier CSD. C'est le comportement par défaut dans Csound 5.08.</li> </ul>
----------------------------------	--

**-d, --nodisplays** Supprimer tous les affichages. Voir *-O* si vous souhaitez enregistrer le compte-rendu dans un fichier.



### Note

Cette option ne fonctionne que depuis la ligne de commande et pas depuis *.csound6rc* ou depuis une section *CsOptions* d'un fichier *csd*. Elle nécessite une prise en compte avant que le son proprement dit ne commence. Elle est ignorée dans *.csound6rc* et dans *CsOptions*.

**--displays** Autoriser les affichages, inversant l'effet d'une éventuelle option *-d* précédente.

**-G, --postscriptdisplay** Supprimer les graphiques, une sortie graphique PostScript est produite à la place.

**-g, --asciisplay** Supprimer les graphiques, une sortie pseudo-graphique ASCII étant produite à la place.

**-H#, --heartbeat=NUM** Imprimer un battement de cœur après chaque écriture de tampon dans le fichier son :

- pas de NUM, une barre tournante.
- NUM = 1, une barre tournante.
- NUM = 2, un point (.)
- NUM = 3, la taille du fichier en secondes.
- NUM = 4, un beep sonore.

**-m NUM, --messagelevel=NUM** Niveau des messages pour la sortie standard (terminal). Prend la *somme* de n'importe lesquelles de ces valeurs :

- 1 = messages d'amplitude de note
  - 2 = message d'échantillons hors intervalle
  - 4 = messages d'avertissement
  - 128 = impression d'information de tests de référence
  - 1024 = suppression des messages d'obsolescence
- Et exactement un de ceux-ci pour choisir le format de l'amplitude des notes :
- 0 = amplitudes brutes, pas de couleur
  - 32 = dB, pas de couleur
  - 64 = dB, hors intervalle colorées en rouge
  - 96 = dB, toutes colorées
  - 256 = brutes, hors intervalle colorées en rouge
  - 512 = brutes, toutes colorées

La valeur par défaut est 135 (128+4+2+1), ce qui signifie tous les messages, valeurs d'amplitude brutes, et impression du temps écoulé à la fin de l'exécution. La coloration des amplitudes brutes fut introduite dans la version 5.04. La suppression des messages d'obsolescence a été ajoutée dans la version 6.14.

<code>--m-amps=NUM</code>	<p>Niveau des messages d'amplitudes sur la sortie standard (terminal).</p> <ul style="list-style-type: none"> <li>• 0 = pas de messages d'amplitude de note</li> <li>• 1 = messages d'amplitude de note</li> </ul>
<code>--m-range=NUM</code>	<p>Niveau des messages de dépassement de limite sur la sortie standard (terminal).</p> <ul style="list-style-type: none"> <li>• 0 = aucun message d'échantillon hors limites</li> <li>• 1 = messages d'échantillons hors limites</li> </ul>
<code>--m-warnings=NUM</code>	<p>Niveau des messages d'avertissement sur la sortie standard (terminal).</p> <ul style="list-style-type: none"> <li>• 0 = pas de messages d'avertissement</li> <li>• 1 = messages d'avertissement</li> </ul>
<code>--m-dB=NUM</code>	<p>Niveau des messages pour le format d'amplitude sur la sortie standard (terminal).</p> <ul style="list-style-type: none"> <li>• 0 = messages d'amplitude absolue</li> <li>• 1 = messages d'amplitude en dB</li> </ul>
<code>--m-colours=NUM</code>	<p>Niveau des messages pour le format d'amplitude sur la sortie standard (terminal).</p> <ul style="list-style-type: none"> <li>• 0 = pas de coloration des messages d'amplitude</li> <li>• 1 = coloration des messages d'amplitude</li> </ul>
<code>--m-benchmarks=NUM</code>	<p>Niveau des messages d'information de test de performance sur la sortie standard (terminal).</p> <ul style="list-style-type: none"> <li>• 0 = pas de nombres de test de performance</li> <li>• 1 = nombres de test de performance</li> </ul>
<code>++msg_color=booléen</code>	<p>Activer les attributs de message (couleurs etc.) ; il peut être nécessaire de les désactiver sur certains terminaux qui impriment des caractères étranges au lieu de modifier les attributs du texte. Par défaut : true.</p>
<code>-v, --verbose</code>	<p>Traduction et exécution détaillées. Imprime les détails de la traduction de l'orchestre et de son exécution, permettant une localisation plus précise des erreurs.</p>
<code>-z NUM, --list-opcodesNUM</code>	<p>Lister les opcodes de cette version :</p>



- pas de NUM, montrer seulement les noms
- NUM = 0, montrer seulement les noms
- NUM = 1, montrer les arguments de chaque opcode dans le format <nomop> <argssortie> <argsentrée>
- NUM = 2, montrer les noms y compris ceux qui sont obsolètes
- NUM = 3, montrer les arguments pour chaque opcode, y compris pour ceux qui sont obsolètes, en utilisant le format <opname> <outargs> <inargs>

## Configuration et contrôle de l'exécution

-B NUM, --hardwarebuf-samps=NUM

Nombre de trames d'échantillonnage audio maintenues dans le tampon du *circuit* CNA. C'est une limite au-dessus de laquelle l'E/S audio *logicielle* va attendre avant de retourner. Une faible valeur réduit le délai audio d'E/S ; mais la valeur est souvent limitée par le matériel, et l'on risque des retards dans les données avec de petites valeurs. Dans le cas de la sortie portaudio (la sortie par défaut en temps réel), le paramètre -B (plus précisément -B / sr) est passé comme valeur de "latence suggérée". En dehors de cela, Csound n'a aucun contrôle sur la manière dont PortAudio interprète le paramètre. La valeur par défaut est 1024 dans Linux, 4096 dans Mac OS X et 16384 dans Windows.

-b NUM, --iobufsamps=NUM

Nombre de trames d'échantillonnage audio dans chaque tampon *logiciel* d'E/S. De grandes valeurs fonctionnent bien, mais les petites valeurs réduiront le délai d'E/S audio et amélioreront la précision temporelle des événements en temps réel. La valeur par défaut est 256 dans Linux, 1024 dans Mac OS X, et 4096 dans Windows. Lors d'une exécution en temps réel, Csound attend les E/S audio toutes les NUM divisions. Il effectue aussi le traitement audio (et interroge d'autres entrées comme le MIDI) toutes les *ksmps* divisions de l'orchestre. On peut synchroniser les deux. Par commodité, si NUM est négatif, la valeur effective est *ksmps* \* -NUM (audio synchrone avec les divisions de période k). Avec de petites valeurs de NUM (par exemple 1) l'interrogation devient fréquente et calée sur les divisions fixes d'échantillonnage du CNA.

Note : si l'on utilise en même temps -iadc et -odac (audio temps réel en mode duplex complet), il faut fixer l'option -b à un multiple entier de *ksmps*.

-daemon

Exécution en mode démon : ne se termine pas si aucun CSD/orchestra n'est donné, ou s'il est vide ou s'il ne compile pas.

-k NUM, --control-rate=NUM

Remplacer le taux de contrôle (*kr*) fourni par l'orchestre.

-L PERIPHERIQUE, --score-in=PERIPHERIQUE

Lire en temps réel des événements de partition en ligne de texte à partir du périphérique *PERIPHERIQUE*. Le nom *stdin* permettra de recevoir les événements de partition de votre terminal, ou d'un autre processus via un tube de communication (pipe). Chaque ligne d'évènement est terminée par un retour chariot. Les évène-

ments sont codés de la même manière que ceux de la *partition numérique standard*, sauf qu'un évènement avec  $p2=0$  sera exécuté immédiatement, et qu'un évènement avec  $p2=T$  sera exécuté T secondes après son arrivée. Les évènements peuvent arriver n'importe quand et dans n'importe quel ordre. La fonction *carry* (*report de valeur*) de la partition est autorisée ici, ainsi que les notes liées ( $p3$  négatif) at les arguments chaîne, mais les pentes d'interpolation et les références *pp* ou *np* ne le sont pas.



## Note

L'option -L n'est valide que sur les système \*NIX qui ont des tuyaux. Elle ne fonctionne pas sous Windows.

--omacro:XXX=YYY	Donner la valeur YYY à la macro d'orchestre XXX
--port=N	Détermine un port UDP pour recevoir des commandes et/ou du code d'instruments ou d'orchestre (implique --daemon) Voir <i>serveur UDP</i> .
--udp-echo	Active l'affichage des commandes UDP sur le terminal. Tout message reçu par le serveur UDP est affiché en écho (que ce soit une commande valide ou non).
--udp-console=address:port	Redirige les messages de la console vers une adresse:port distante via UDP.
--udp-mirror-console=address:port	Recopie les messages de la console vers une adresse:port distante via UDP.
-r NUM, --sample-rate=NUM	Remplacer le taux d'échantillonnage ( <i>sr</i> ) fourni par l'orchestre.
--sample-accurate	Démarre et arrête les instances d'instruments à l'échantillon le plus proche de l'instant demandé. Ceci diffère du comportement traditionnel de Csound qui arrondi le temps au cycle-k le plus proche. Noter que cela ne fonctionne pas avec les notes liées.
--sched	<i>Seulement dans linux.</i> Utiliser la planification du temps réel et le verrouillage de la mémoire. (nécessite également -d et -o <i>dac</i> ou -o <i>devaudio</i> ). Voir aussi --sched=N ci-dessous.
--sched=N	<i>Seulement sur linux.</i> Identique à --sched, mais permet de spécifier une valeur de priorité: si N est positif (dans l'intervalle 1 à 99) la politique de planification SCHED_RR sera utilisée avec une priorité de N ; autrement, SCHED_OTHER est utilisée avec le niveau "de gentillesse" (nice) à N. On peut aussi l'utiliser avec le format --sched=N,MAXCPU,TEMPS pour autoriser l'utilisation d'un processus léger (thread) de surveillance qui terminera Csound si le temps moyen d'utilisation de CPU dépasse MAXCPU pourcents sur une durée de TEMPS secondes (à partir de Csound 5.00).
--smacro:XXX=YYY	Donner la valeur YYY à la macro de partition XXX
--strset	<i>Csound 5.</i> L'option --strset permet de passer des chaînes de caractères à strset depuis la ligne de commande, dans le format '--str-

	setN=VALEUR'. Utile pour passer des paramètres à l'orchestre (par exemple des noms de fichier).
--skip_seconds=float	(min: 0) Commencer la reproduction au temps indiqué (en secondes), en ignorant les événements antérieurs de la partition ou du fichier MIDI.
-t NUM, --tempo=NUM	Utiliser les pulsations non interprétées de <i>score.srt</i> pour cette exécution, et fixer le tempo initial à <i>NUM</i> pulsations par minute. Quand cette options est positionnée, le tempo de l'exécution de la partition est également contrôlable depuis l'orchestre. ATTENTION : ce mode d'opération est expérimental et n'est pas forcément fiable.
-j NUM, --num-threads=NUM	<i>NUM</i> processus sont rendus disponibles pour l'exécution. Ce n'est avantageux que si le nombre de processeurs de l'ordinateur est supérieur ou égal au nombre de processus demandés. Ca peut aussi ralentir l'exécution si <i>ksmps</i> est trop petit.

## Divers

-@ FICHIER	Une ligne de commande étendue est fournie dans le fichier « FICHIER »
-C, --cscore	Utiliser le traitement par Cscore du fichier partition.
--default-paths	Autoriser à nouveau l'addition de répertoire de CSD/ORC/SCO aux chemins de recherche, si cette possibilité avait été désactivée par une option <i>--no-default-paths</i> précédente (par exemple dans <i>.csound6rc</i> ).
-D, --defer-gen1	Différer le chargement des fichiers sons de GEN01 jusqu'au moment de l'exécution.
--env:NOM=VALEUR	Positionner la variable d'environnement NOM à VALEUR. Note : on ne peut pas positionner toutes les variables d'environnement de cette manière, car certaines d'entre elles sont lues avant l'analyse de la ligne de commande. Cette option fonctionne entre autres avec INCDIR, SADIR, <i>SFDIR</i> et <i>SSDIR</i> .
--env:NOM+=VALEUR	Ajouter VALEUR à la liste des chemins de recherche dont le séparateur est ';' dans la variable d'environnement NOM (ça peut-être INCDIR, SADIR, <i>SFDIR</i> ou <i>SSDIR</i> ). Si un fichier est trouvé dans plusieurs répertoires, c'est le dernier qui est utilisé.
--expression-opt	<p>Noter que cette option n'a aucun effet dans Csound 6. Seulement dans Csound 5. Activer certaines optimisations dans les expressions :</p> <ul style="list-style-type: none"> <li>• Les affectations redondantes sont éliminées chaque fois que c'est possible. Par exemple la ligne <i>a1 = a2 + a3</i> sera compilée en <i>a1 Add a2, a3</i> au lieu de <i>#a0 Add a2, a3 a1 = #a0</i> évitant une variable temporaire et un appel d'opcode. Moins d'appels d'opcode induisent une utilisation moindre du CPU (un orchestre moyen peut être compilé 10% plus vite avec <i>--expression-opt</i>, mais cela dépend aussi largement du nombre d'expressions utilisées, du</li> </ul>

taux de contrôle (voir également ci-dessous), etc ; ainsi, la différence peut être moindre, mais aussi beaucoup plus).

- le nombre de variables temporaires de taux a et de taux k est réduit significativement. L'expression

```
(a1 + a2 + a3 + a4)
```

sera compilée en

```
#a0 Add a1, a2
#a0 Add #a0, a3
#a0 Add #a0, a4      ; (le résultat se trouve dans #a0)
```

au lieu de

```
#a0 Add a1, a2
#a1 Add #a0, a3
#a2 Add #a1, a4      ; (le résultat se trouve dans #a2)
```

Les avantages d'avoir moins de variables temporaires sont :

- moins de mémoire cache utilisée, ce qui peut améliorer les performances des orchestres avec beaucoup d'expressions de taux a et un faible taux de contrôle (par exemple ksmps = 100)
- les grands orchestres sont chargés plus vite grâce au nombre moins important d'identifiants différents
- les erreurs de dépassement d'indice (par exemple quand des messages comme Case2: indx=-56004 (ffff253c); (short)indx = 9532 (253c) sont imprimés et que Csound a un comportement bizarre ou plante) peuvent être corrigées, car de telles erreurs sont provoquées par trop de noms de variable différents (spécialement au taux a) dans un seul instrument.

Noter que l'optimisation (pour des raisons techniques) n'est pas exécutée sur les i-variables temporaires.



## Avertissement

Lorsque --expression-opt est activé, il est interdit d'utiliser la fonction i() avec un argument expression, et il n'est pas prudent de compter au temps i sur la valeur de k-expressions.

--version

Afficher les informations de version.

--get-system-sr


Affiche le sr du système et termine l'application, nécessite au préalable -o dac. Si l'audio ne supporte pas cette requête, -1 est retourné.

--help

Afficher un message d'aide en ligne.

--devices[=x]

Donne une liste des périphériques audio (x=out, seulement les périphériques en sorties ; x=in, en entrée; sinon en entrée et en sortie), puis termine l'application.

-I, --i-only	<i>seulement au temps i</i> . Allouer et initialiser tous les instruments selon la partition, mais en ignorant tous les traitement de temps p (pas de k-signaux ni de a-signaux, et donc aucune amplitude et aucun son). Fournit un moyen rapide de tester la validité des p-champs de la partition et des i-variables de l'orchestre. Cette option est mutuellement exclusive avec l'option --syntax-check-only.
--ignore_csopts=entier	S'il vaut 1, Csound ignorera toutes les options spécifiées dans la section CsOptions du fichier csd. Voir <i>Format de Fichier Unifié pour les Orchestres et les Partitions</i> .
--ksmps=N	Force ksmps à N (versions 6.05 et supérieures).
--max_str_len=entier	(min: 10, max: 10000) Longueur maximale des variables chaîne + 1 ; la valeur par défaut est 256 autorisant une longueur de 255 caractères. La longueur des constantes chaîne n'est pas limitée par ce paramètre.
-N, --notify	Avertir (par un beep) quand la partition ou la piste MIDI est terminée.
--no-default-paths	Désactiver l'addition de répertoire de CSD/ORC/SCO au chemin de recherche.
--no-expression-opt	Désactiver l'optimisation des expressions.
-O FICHIER, --logfile=FICHIER	Compte-rendu dans le fichier <i>FICHIER</i> . Si <i>FICHIER</i> est null (c-à-d. -O null ou --logfile=null) toutes les impressions de message sur la console sont désactivées.
<div style="display: flex; align-items: center;">  <div> <p><b>Note</b></p> <p>Cette option ne fonctionne que depuis la ligne de commande et pas depuis .csound6rc ou depuis une section CsOptions d'un fichier csd. Elle nécessite une prise en compte avant que le son proprement dit ne commence. Elle est ignorée dans .csound6rc et dans CsOptions.</p> </div> </div>	
--opcode-lib=NOMBIB	Charge la bibliothèque de greffon <i>NOMBIB</i> .
--orc nomorc	L'argument est le fichier d'orchestre. Utilisé lorsque l'on a pas besoin de partition. (Csound 5.18 et ultérieur).
--syntax-check-only	Provoque l'arrêt de Csound immédiatement après que les parseurs de l'orchestre et de la partition ont fini la vérification de la syntaxe des fichiers d'entrée et avant que l'orchestre n'exécute la partition. Cette option est mutuellement exclusive avec l'option --i-only. (Csound 5.08 et versions ultérieures).
--simple-sorted-score	Conserve le fichier de partition trié score.srt dans un format simple et lisible lors de la sortie. (Nouveau dans la version 6.14).
-U UTILITE, --utility=UTILITE	Invoquer le programme utilitaire <i>UTILITE</i> . En donnant un nom invalide on obtient une liste des utilitaires.
-x FICHIER, --extract-score=FICHIER	Extraire un morceau de la partition triée, <i>score.srt</i> , en utilisant le fichier d'extraction <i>FICHIER</i> (voir <i>Extract</i> ).

## Variables d'environnement de Csound

Csound peut utiliser les variables d'environnement suivantes :

- **SFDIR** : Répertoire par défaut pour les fichiers son. Utilisé si aucun chemin complet n'est fourni pour les fichiers son.
- **SSDIR** : Répertoire par défaut pour les fichiers audio et MIDI en entrée (source). Utilisé si aucun chemin complet n'est fourni pour les fichiers son. On peut l'utiliser conjointement avec **SFDIR** pour fixer des répertoire d'entrée et de sortie séparés. Prière de noter qu'aussi bien les fichiers MIDI que les fichiers audio sont recherchés aussi dans **SSDIR**.
- **SADIR** : Répertoire par défaut pour les fichiers d'analyse. Utilisé si aucun chemin complet n'est donné pour les fichiers d'analyse.
- **SFOUTYP** : Fixe le type par défaut du fichier de sortie. Actuellement ne sont valides que 'WAV', 'AIFF' et 'IRCAM'. Cette variable est testée par l'exécutable de csound et par les utilitaires et elle est utilisée si aucun type de fichier de sortie n'a été spécifié.
- **INCDIR** : Répertoire des fichiers à inclure. Spécifie l'endroit où se trouvent les fichiers utilisés par les instructions *#include*.
- **OPCODE6DIR** : Définit l'endroit où se trouvent les greffons d'opcode en version simple précision (32 bit).
- **OPCODE6DIR64** : Définit l'endroit où se trouvent les greffons d'opcode en version double précision (64 bit).
- **SNAPDIR** : Utilisée par les opcodes de contrôle graphique FLTK pour charger et sauvegarder les instantanés.
- **CSOUND6RC** : Définit le fichier de ressource (ou de configuration) de csound. Un chemin complet avec le nom d'un fichier contenant des options de csound doit être donné. Cette variable vaut `.csound6rc` par défaut.
- **CSSTRNGS** : A partir de Csound 5.00, la localisation des messages est contrôlée par les deux variables d'environnement **CSSTRNGS** et **CS\_LANG**, qui sont toutes deux optionnelles. **CSSTRNGS** pointe vers un répertoire contenant des fichiers `.xmg`.
- **CS\_LANG** : Sélectionne une langue pour les messages de csound.
- **RAWWAVE\_PATH** : Utilisée par les opcodes STK pour trouver les fichiers son bruts. Ne sert que si vous utilisez des opcodes de sur-couche STK comme `STKBowed` ou `STKBrass`.
- **CSNOSTOP** : Si cette variable d'environnement a pour valeur "yes", alors tous les affichages graphiques sont fermés à la fin de l'exécution (ce qui veut dire que vous n'en verrez peut-être pas grand chose dans le cas d'une exécution courte en temps différé). Dans le cas contraire, il faut cliquer sur "Quit" dans la fenêtre d'affichage FLTK pour sortir, ce qui permet de voir les graphiques même après que la fin de la partition soit atteinte.
- **MFDIR** : Répertoire par défaut pour les fichiers MIDI. Utilisé si aucun chemin complet n'est donné pour les fichiers MIDI. Prière de noter que les fichiers MIDI sont également recherchés dans **SSDIR** et **SFDIR**.
- **CS\_OMIT\_LIBS** : Permet de définir une liste de bibliothèques de greffons à ignorer. Les noms des bibliothèques peuvent être séparés par des virgules et le préfixe "lib" n'est pas nécessaire.

Pour plus d'information sur SFDIR, SSDIR, SADIR, MFDIR et INCDIR voir *Répertoires et Fichiers*.

Les seules variables d'environnement obligatoires sont OPCODE6DIR et OPCODE6DIR64. Il est très important de les remplir correctement, sinon la plupart des opcodes ne seront pas disponibles. Assurez-vous de fixer le chemin correctement en fonction de la précision de votre exécutable. Si vous lancez csound en ligne de commande sans aucun argument vous devriez voir un texte ressemblant à : Csound version 6.03.1 (double samples) May 10 2014. Ce texte fait référence à la version double précision.

CSSTRNGS et CS\_LANG sont actuellement peu utiles car Csound n'a pas encore été complètement traduit dans d'autres langues.

Voici d'autres variables d'environnement qui ne sont pas propres à Csound mais qui peuvent être importantes :

- **PATH** : Le répertoire contenant les exécutables de csound devrait être listé dans cette variable.
- **LADSPA\_PATH** et **DSSI\_PATH** : Ces variables d'environnement sont nécessaires si vous utilisez les opcodes du greffon *dssi4cs* (hôtes LADSPA et DSSI).
- **CSDOCDIR** : Spécifie le répertoire dans lequel se trouvent les fichiers d'aide html. Bien qu'elle ne soit pas utilisée directement par Csound, cette variable d'environnement peut aider les frontaux et les éditeurs (qui la mettent en œuvre) à trouver le manuel de csound.

## Fixer les variables d'environnement

### Sur la ligne de commande

On peut fixer les variables d'environnement sur la ligne de commande ou dans le fichier de configuration .csoundrc en utilisant l'option de ligne de commande --env:NOM=VALEUR ou --env:NOM+=VALEUR, où NOM est le nom de la variable d'environnement, et VALEUR est sa valeur. Voir *Options de Ligne de Commande*.



#### Note

Prière de noter que cette méthode ne fonctionnera pas pour les variables d'environnement qui sont lues avant les arguments de la ligne de commande. Pour SADIR, SSDIR, SFDIR, INCDIR, SNAPDIR, RAWWAVE\_PATH, CSNOSTOP, SFOUTYP cela devrait marcher, mais les variables d'environnement suivantes doivent être fixées dans le système avant de lancer csound : OPCODE6DIR, OPCODE6DIR64, CSSTRINGS, et CS\_LANG. Actuellement (v. 5.02) CSOUNDRC peut être fixée par --env, mais cette possibilité n'est pas garantie dans les versions futures.

## Windows

Pour fixer une variable d'environnement dans Windows XP et 2000 aller dans Panneau de Contrôle->Système->Avancé et cliquer sur le bouton 'Variables d'environnement'. Dans les autres versions de Windows antérieures à Windows XP et Windows 2000 on fixe les variables d'environnement dans le fichier autoexec.bat. Aller dans 'Poste de travail', choisir le lecteur C:, cliquer avec le bouton droit sur autoexec.bat, et choisir 'Edition'. Le format de l'instruction est : SET NOM=VALEUR.

## Linux

Il y a plusieurs manières de fixer les variables d'environnement sur linux. On peut les initialiser avec la commande de shell *export*, dans le fichier .bashrc ou des fichiers similaires ou en les ajoutant au fichier /etc/profile.

## Mac

```
~$ export OPCODE6DIR64=/Users/you/your/Csound6/build
```

De plus si l'on a un bash shell par défaut, alors il est plus facile de modifier le fichier .bashrc ou le fichier /etc/profile.

A noter que si l'on choisit l'une des méthodes ci-dessus, par exemple modifier le fichier .bashrc, alors les variables d'environnement sont allouées quand un nouveau shell est créé. Ceci peut poser un problème lorsque votre application implémente une interface Quartz ou Aqua et n'utilise pas la ligne de commande.

Si c'est le cas, la solution standard (jusqu'à OS 10.3.9 et à moins que l'application utilise l'API de csound et fixe directement les variables d'environnement) consiste à créer un fichier contenant une liste de propriétés XML (un fichier nommé .plist par l'OS). Ce fichier devrait se trouver dans ~/.MacOSX/Environment.plist. Cette solution a été utilisée spécifiquement pour l'objet [csoundapi~] pour Pd sur OS X. Comme Pd utilise un style de paquetage .app natif OS X, et s'exécute en dehors de l'interface Aqua, les moyens standard de fournir les variables d'environnement à Csound ne fonctionnent pas. La solution consiste à fixer les variables d'environnement de Csound pour l'environnement Aqua.

Il est probable que la plupart des utilisateurs n'auront pas de répertoire caché .MacOSX dans leur répertoire \$HOME (alias ~/). Il faut d'abord créer ce répertoire et y ajouter Environment.plist. Le contenu du fichier Environment.plist ressemblera à ceci :

```
<?xml version="1.0" encoding='UTF-8'?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>OPCODEDIR</key>
<string>/Library/Frameworks/CsoundLib.framework/Versions/5.1/Resources/Opcodes</string>
<key>OPCODEDIR64</key>
<string>/Volumes/ExternalHD/devel/csound5/lib64</string>
<key>INCDIR</key>
<string>/Volumes/ExternalHD/CSOUND/include</string>
<key>SFDIR</key>
<string>/Volumes/ExternalHD/iTunes/csoundaudio</string>
</dict>
</plist>
```

et ainsi de suite, en utilisant la balise XML <key> pour chaque variable d'environnement requise par l'API et la balise <string> pour le chemin correspondant dans le système.

Prière de noter qu'il faut se déconnecter et se reconnecter (login) pour que ces changements prennent effet.

## Format de fichier unifié pour les orchestres et les partitions

### Description

Le Format de fichier unifié, introduit à partir de la version 3.50 de Csound, permet de combiner dans le même fichier l'orchestre et la partition, ainsi que les options de ligne de commande. Le fichier a pour extension .csd. Ce format fut introduit à l'origine par Michael Gogins dans AXCSound.

Le fichier est un fichier de données structurées qui utilise un langage de balises, de la famille SGML comme HTML. Une balise ouvrante (<balise>) et une balise fermante (</balise>) servent à délimiter les différents éléments. Ce fichier est sauvegardé comme un fichier texte.



## Format du fichier de données structurées

### Éléments obligatoires

la première balise du fichier doit être la balise ouvrante `<CsoundSynthesizer>`. La dernière balise du fichier doit être la balise fermante `</CsoundSynthesizer>`. Cet élément sert à avertir le compilateur csound du format `.csd`. Tout texte situé avant la balise de début et après la balise de fin est ignoré par Csound. Cette balise peut aussi s'écrire `<CsoundSynthesiser>`.

### Options (<CsOptions>)

Les *options de ligne de commande* de Csound sont insérées dans l'Élément Options. La section est délimitée par la balise ouvrante `<CsOptions>` et par la balise fermante `</CsOptions>`. Les lignes commençant par `#` ou par `;` sont traitées comme des commentaires.

### Orchestre (<CsInstruments>)

Les définitions d'instruments (orchestre) sont mises dans l'Élément Instruments. Les instructions et la syntaxe de cette section sont identiques à celles du *fichier orchestre* de Csound, et répondent aux mêmes besoins, y compris les instructions d'en-tête (*sr*, *kr*, etc). Cet Élément Instruments est délimité par la balise ouvrante `<CsInstruments>` et par la balise fermante `</CsInstruments>`.

### Partition (<CsScore>)

Les instructions de la partition Csound sont mises dans l'Élément Score. Les instructions et la syntaxe de cette section sont identiques à celles du *fichier partition* de Csound, et répondent aux mêmes exigences. L'Élément Score est délimité par la balise ouvrante `<CsScore>` et par la balise fermante `</CsScore>`.

Les instructions de partition peuvent alternativement être générées par un programme externe en utilisant la balise `CsScore` avec un exécutable en attribut. Les lignes allant jusqu'à la balise fermante `</CsScore>` sont copiées dans un fichier et le programme externe nommé est appelé avec ce nom de fichier et le fichier de partition destinataire. Le programme externe doit créer une partition Csound standard.

## Éléments optionnels

### Inclusion de fichiers Base64 (<CsFileB>)

On peut inclure des fichiers encodés en Base64 avec la balise `<CsFileB filename= nomfichier>`, où *nomfichier* est le nom du fichier à inclure. Les données encodées en Base64 doivent se terminer par une balise `</CsFileB>`. Pour encoder les fichiers, on peut se servir des utilitaires *csb64enc* et *makecsd* (inclus dans Csound à partir de la version 5.00). Le fichier sera extrait dans le répertoire courant, et effacé à la fin de l'exécution. S'il existe déjà un fichier du même nom, il n'est pas écrasé, mais au contraire, une erreur est levée.

On peut inclure des fichiers MIDI encodés en Base64 avec la balise `<CsMidifileB filename= nomfichier>`, où *nomfichier* est le nom du fichier qui contient l'information MIDI. Il n'y a pas de balise fermante associée. Ceci a été ajouté dans la version 4.07 de Csound. Note : il n'est pas recommandé d'utiliser cette balise ; il vaut mieux utiliser `<CsFileB>`.

On peut inclure des fichiers d'échantillons encodés en Base64 avec la balise `<CsSampleB filename= nomfichier>`, où *nomfichier* est le nom du fichier qui contient les échantillons. Il n'y a pas de balise fermante associée. Ceci a été ajouté dans la version 4.07 de Csound. Note : il n'est pas recommandé d'utiliser cette balise ; il vaut mieux utiliser `<CsFileB>`.

## Inclusion de fichiers non encodés (<CsFile>)

On peut inclure des fichiers non encodés avec la balise `<CsFile filename=nomfichier>`, où *nomfichier* est le nom du fichier à inclure. Les données doivent être suivies de la balise fermante `</CsFile>` seule sur ligne. Le fichier est extrait dans le répertoire courant et effacé à la fin de l'exécution. Si un fichier du même nom existe déjà, il n'est pas écrasé et il se produit une erreur.

## Limitation de version (<CsVersion>)

On peut se limiter à certaines versions de Csound en plaçant l'une de ces instructions entre la balise ouvrante `<CsVersion>` et la balise fermante `</CsVersion>` :

Before #.#

ou

After #.#

où #.# est le numéro de version de Csound requis. La deuxième instruction peut s'écrire simplement comme :

#.#

Ceci a été ajouté dans le version 4.09 de Csound.

## Information de licence (<CsLicence> ou <CsLicense>)

Des détails de licence peuvent être inclus entre la balise ouvrante `<CsLicence>` et la balise fermante `</CsLicence>`. Il n'y a pas de format pour cette information, n'importe quel texte est acceptable. Ce texte sera imprimé par Csound sur la console lorsque le CSD sera exécuté.

## Information de licence (<CsShortLicence> ou <CsShortLicense>)

Depuis la version 6.05 on peut aussi inclure des détails de licence entre la balise ouvrante `<CsShortLicence>` et la balise fermante `</CsShortLicence>`. Cet option concerne sept licences bien connues, encodée par un entier.

- 0 : "Tous droits réservés" (valeur par défaut)
- 1 : "Creative Commons Attribution-NonCommercial-NoDerivatives (CC BY-NC-ND)"
- 2 : "Creative Commons Attribution-NonCommercial-ShareAlike (CC BY-NC-SA)"
- 3 : "Creative Commons Attribution-NonCommercial (CC BY-NC)"
- 4 : "Creative Commons Attribution-NoDerivatives (CC BY-ND)"
- 5 : "Creative Commons Attribution-ShareAlike (CC BY-SA)"
- 6 : "Creative Commons Attribution-ShareAlike (CC BY)"
- 7 : "Sous licence BSD"

## HTML imbriqué (<html>)

Tout code HTML valide peut être imbriqué dans un fichier CSD. Ce code doit être structuré exactement comme dans une page Web ordinaire. Il peut contenir tout code HTML, Javascript, Cascading Style Sheet, WebGL, etc., valide.

Dans certains frontaux et environnements de programmation, comprenant au moins CsoundQt ou Csound pour Android, cette page sera analysée, exécutée et affichée dans un navigateur Web embarqué dans l'environnement. Le code Javascript de cette page aura accès à un objet *csound* global qui implémente les fonctions suivantes qui forment un sous-ensemble de l'API de Csound. Les noms, les types de données

et l'utilisation de ces fonctions sont exactement les mêmes que ceux qui sont détaillés dans le manuel de référence de l'API de Csound.

```
[int] getVersion ();
compileOrc (orchestra_text);
[double] evalCode (orchestra_expression);
readScore (score_text);
setControlChannel (channel_name, numeric_value);
[double] getControlChannel (channel_name);
message (message_string);
[int] getSr ();
[int] getKsmps ();
[int] getNchnls ();
// Ne fait pas partie de l'API de Csound -- appelé par l'environnement pour
// détecter si Csound est en cours d'exécution.
[int] isPlaying ();
```

Les éléments HTML du fichier CSD peuvent être utilisés pour créer des interfaces utilisateur personnalisées pour la pièce, pour générer des événements de partition et même du code d'orchestre en utilisant Javascript, pour stocker des préréglages des contrôleurs graphiques et pour bien d'autres fonctions. Les exemples *GameOfLife3D.csd* [examples/GameOfLife3D.csd] et *LindenmayerCanvas.csd* [examples/LindenmayerCanvas.csd] démontrent ces utilisations (testés dans CsoundQt ; l'exécution de ces exemples nécessitent des ressources supplémentaires que l'on trouve dans le répertoire des exemples de Csound dans GIT).

## Exemple

Ci-dessous un fichier exemple, test.csd, qui produit un fichier .wav échantillonné à 44,1 kHz contenant une seconde d'une onde sinus à 1 kHz. L'affichage est supprimé. test.csd a été créé à partir de deux fichiers, tone.orc et tone.sco, avec l'addition des options de ligne de commande.

```
<CsoundSynthesizer>
; test.csd - un fichier Csound de données structurées

<CsOptions>
-W -d -o tone.wav
</CsOptions>

<CsVersion>      ; section facultative
Before 4.10      ; ces deux instructions testent si
After 4.08       ; la version de Csound est la 4.09
</CsVersion>

<CsInstruments>
; à l'origine, tone.orc
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
instr 1
    al oscil p4, p5, 1 ; simple oscillateur
    out al
endin
</CsInstruments>

<CsScore>
; à l'origine, tone.sco
f1 0 8192 10 1
i1 0 1 20000 1000 ; joue un son pur à un kHz pendant une seconde
e
```

```
</CsScore>
```

```
</CsoundSynthesizer>
```

## Fichier de paramètres de ligne de commande (.csound6rc)

Si le fichier *.csound6rc* existe, il sera utilisé pour fixer les paramètres de la ligne de commande. Ceux-ci peuvent être redéfinis. Csound 5.00 et les versions ultérieures lisent ce fichier d'abord depuis le répertoire HOME (ou le chemin complet défini par la *variable d'environnement* CSOUND6RC), et ensuite depuis le répertoire courant. Si les deux existent, les options de *.csound6rc* du répertoire courant seront prioritaires. Ce fichier a la même forme qu'un fichier *.csd*, mais sans les balises. Les lignes commençant par # ou ; sont traitées comme des commentaires.

Un fichier *.csound6rc* peut contenir des éléments comme ceux-ci :

```
-+rtaudio=portaudio -odac2 -iadc2 -+rtmidi=winmme -M1 -Q1 -m0
```

Dans ce cas, csound générera sa sortie en temps réel et recevra son entrée en temps réel depuis le périphérique 2, en utilisant l'interface du pilote portaudio. Il utilisera les entrées et les sorties MIDI en temps réel sur l'interface 1. Il imprimera très peu de messages (-m0). Ces options seront utilisées par défaut en l'absence d'autres options spécifiées dans la balise <CsOptions> du fichier *.csd* ou dans la ligne de commande (voir *Ordre de priorité*).

## Prétraitement du fichier de partition

### La fonction Extract

Cette fonction va extraire une partie d'un fichier de partition numérique triée en suivant les instructions venant d'un fichier de contrôle. Le fichier de contrôle contient une liste d'instruments et deux points dans le temps depuis (from) et à (to), de la forme :

```
instruments 1 2 from 1:27.5 to 2:2
```

Les étiquettes des composants peuvent être abrégés en i, f et t. Les points dans le temps marquent le début et la fin de l'extraction en termes de :

```
[no de section] : [no de pulsation].
```

chacune des trois parties de l'argument est optionnelle. Les valeurs par défaut lorsque i, f ou t sont manquants sont :

```
tous les instruments, début de la partition, fin de la partition.
```

### Prétraitement indépendant avec Scsort

Bien que le résultat de tout le prétraitement de la partition se trouvent dans le fichier *score.srt* après l'exécution de l'orchestre (il existe dès que le prétraitement de la partition est fini), l'utilisateur peut vouloir parfois lancer ces phases indépendamment. La commande

```
scot nomfichier
```

va traiter le fichier au format Scot *nomfichier*, et produira comme résultat une *partition numérique standard* dans un fichier appelé *score* pour consultation ou traitement ultérieur.

### La commande

```
scsort < fichierentrée > fichiersortie
```

effectuera les prétraitements de Report de Valeur (Carry), Tempo et Tri sur une partition numérique dans fichierentrée, déposant le résultat dans fichiersortie.

De même *extract*, lui aussi invoqué normalement comme élément de la *commande Csound*, peut être invoqué comme programme autonome :

```
extract xfile < partition.triée > extrait.partition
```

Cette commande attend une partition déjà triée. Une partition non triée doit d'abord passer par Scsort pour ensuite enchaîner avec le programme extract :

```
scsort < fichierpartition > | extract xfile > extrait.partition
```

---

# Utiliser Csound

On peut faire fonctionner Csound dans divers modes et configurations. La méthode originale pour lancer Csound était un programme de console (invite DOS pour Windows, Terminal pour Mac OS X). Bien sûr, ceci fonctionne toujours. Lancer `csound` sans argument retourne une liste d'options de commande en ligne, qui sont expliquées plus en détail dans la section *options de ligne de commande (par catégorie)*. Normalement, l'utilisateur exécute quelque chose comme :

```
csound monfichier.csd
```

ou si l'on utilise des fichiers d'orchestre (`orc`) et de partition (`sco`) séparés :

```
csound monorchestre.orc mapartition.sco
```

On peut trouver plusieurs fichiers `.csd` dans le répertoire des exemples. La plupart des articles de ce manuel sur les opcodes incluent également des fichiers `.csd` simples montrant l'utilisation de l'opcode.

Il y a aussi plusieurs *frontaux* que l'on peut utiliser pour lancer `csound`. Un *frontal* est un programme graphique qui facilite la tâche de lancer `csound`, et qui fournit parfois des fonctionnalités d'édition et de composition.

Csound a aussi plusieurs moyens de produire une sortie. Il peut :

- Lire et écrire dans des fichiers son (restitution différée) - En utilisant les options `-o` et `-i` pour spécifier un fichier de sortie.
- Lire et écrire des données audio-numériques en utilisant une carte son (restitution en ) - En utilisant les options `-odac` et `-iadc`
- Lire et écrire dans des fichiers MIDI (temps différé) - En utilisant les options `-F` et `--midioutfile`.
- Lire et écrire des données MIDI en utilisant une interface et un contrôleur MIDI (contrôle en temps réel) - En utilisant les options `-M` et `-Q`.

## Sortie console de Csound



### Note

Cette sortie vient de `csound5` ; les versions ultérieures peuvent différer légèrement.

Pendant son exécution, Csound écrit une sortie texte sur la console, qui montre des données sur cette exécution. Une sortie console ressemble à ceci :

```
time resolution is 0.455 ns
PortMIDI real time MIDI greffon for Csound
virtual_keyboard real time MIDI greffon for Csound
PortAudio real-time audio module for Csound
0dBFS level = 32768.0
Csound version 5.10 beta (float samples) Apr 19 2009
libsndfile-1.0.17
Reading options from $HOME/.csoundrc
UnifiedCSD:  oscil.csd
STARTING FILE
Creating options
Creating orchestra
```

```

Creating score
orchname: /tmp/csound-XYACV6.orc
scorename: /tmp/csound-IYtLAJ.sco
rtaudio: ALSA module enabled
rtmidi: PortMIDI module enabled
orch compiler:
17 lines read
    instr 1
Elapsed time at end of orchestra compile: real: 0.129s, CPU: 0.020s
sorting score ...
    ... done
Elapsed time at end of score sort: real: 0.130s, CPU: 0.020s
Csound version 5.10 beta (float samples) Apr 19 2009
displays suppressed
0dBFS level = 32768.0
orch now loaded
audio buffered in 256 sample-frame blocks
ALSA input: total buffer size: 1024, period size: 256
reading 1024-byte blks of shorts from adc (RAW)
ALSA output: total buffer size: 1024, period size: 256
writing 1024-byte blks of shorts to dac
SECTION 1:
ftable 1:
new alloc for instr 1:
B 0.000 .. 2.000 T 2.000 TT 2.000 M: 10000.0 10000.0
Score finished in csoundPerform().
inactive allocs returned to freespace
end of score.          overall amps: 10000.0 10000.0
          overall samples out of range: 0 0
0 errors in performance
Elapsed time at end of performance: real: 2.341s, CPU: 0.050s
345 1024-byte soundblks of shorts written to dac
Removing temporary file /tmp/csound-CoVcrm.srt ...
Removing temporary file /tmp/csound-IYtLAJ.sco ...
Removing temporary file /tmp/csound-XYACV6.orc ...

```

La sortie console de Csound est assez fournie, particulièrement avant le début de l'exécution proprement dite (version, greffons chargés, etc.). L'exécution commence lorsqu'apparaît sur la console :

```
SECTION 1:
```

Dans cette exécution particulière, les lignes :

```

new alloc for instr 1:
B 0.000 .. 2.000 T 2.000 TT 2.000 M: 10000.0 10000.0

```

montrent qu'une note pour l'instrument 1, durant 2 secondes et commençant à la date 0.000, a été produite avec une amplitude de 10000 sur les canaux 1 et 2. Une section importante de la sortie console est :

```

end of score.          overall amps: 10000.0 10000.0
          overall samples out of range: 0 0

```

qui montre l'amplitude globale et le nombre d'échantillons qui ont été rognés parce qu'ils étaient hors limites.

La ligne :

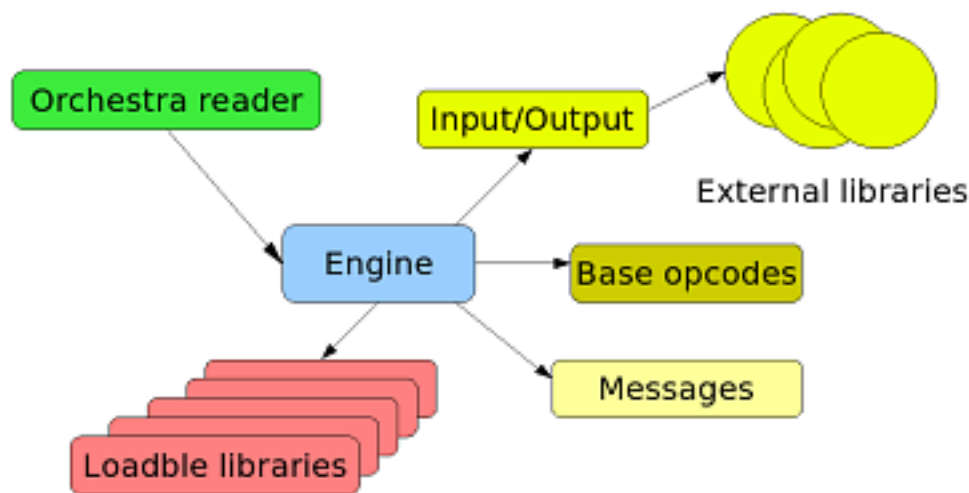
```
Elapsed time at end of performance: real: 2.341s, CPU: 0.050s
```

montre le temps d'horloge et le temps CPU utilisés par le processeur pour compléter le travail. Si le temps CPU est inférieur au temps d'horloge, cela veut dire que le csd peut être exécuté en temps réel (à moins qu'il ne contienne certaines sections très gourmandes en ressources CPU). La valeur "real time" est le temps total de traitement et il est supérieur car il comprend les accès disque, le chargement de modules, etc. (le temps CPU ne comptabilise que les calculs numériques). Si vous avez un son qui dure 100 secondes et

que sa génération hors ligne ne dure que 5 secondes, cela veut dire que vous n'utilisez qu'environ 5% du CPU, et que son exécution ne nécessite que 0.05 du temps réel.

## Comment Csound fonctionne

Csound calcule et génère sa sortie en utilisant des "générateurs unitaires" (ugens) appelés *opcodes*. Ces opcodes sont utilisés pour définir des *instruments* dans l'*orchestre*. Quand vous lancez Csound, le moteur charge les Opcodes de base, et les opcodes contenus dans des "bibliothèques d'opcodes" séparées et chargeable. Il interprète ensuite l'orchestre (au moyen du chargeur d'orchestre). Le moteur met en place une chaîne de traitement des instruments, qui reçoit ensuite des événements depuis la partition ou en temps réel. La chaîne de traitement utilise les modules d'entrée/sortie pour générer la sortie. Il y a des modules qui peuvent écrire dans un fichier, ou générer une *sortie audio en temps réel*.



La Structure Modulaire de Csound.

## Les tampons de traitement de Csound

Csound traite les données audio par blocs d'échantillons appelés tampons. Il y a trois couches de tampons séparées :

1. *spout* = tampon logiciel de bas niveau de Csound, contient *ksmps* trames d'échantillon. Csound traite les événements de contrôle en temps réel toutes les *ksmps* trames d'échantillon.
2. *-b* = Tampon logiciel intermédiaire de Csound (le tampon "logiciel"), en trames d'échantillon. Devrait être (mais ce n'est pas nécessaire) un multiple entier de *ksmps* (peut également être égal à *ksmps*). Une fois toutes les *ksmps* trames d'échantillon, Csound copie *spout* dans le tampon *-b*. Une fois toutes les *-b* trames d'échantillon, Csound copie le tampon *-b* dans le tampon "matériel" *-B*.
3. *-B* = tampon interne de la carte son (le tampon "matériel"), en trames d'échantillon. Devrait être (et cela peut être nécessaire) un multiple entier de *-b*. Si Csound n'arrive pas à délivrer un des *-b*, les trames d'échantillon *-b* en plus dans *-b* sont toujours là pour que la carte son continue de jouer tandis que



Csound se rattrape. Mais ils peuvent être de la même taille si vous escomptez que Csound sera toujours en continuité avec la carte son.

## Valeurs d'amplitude dans Csound

Les valeurs d'amplitude dans Csound sont toujours relatives à une valeur "*0dbfs*" représentant l'amplitude de crête avant écrêtement, soit dans un codec AN/NA, soit dans un fichier son avec une étendue définie (ce qui est le cas de WAVE et de AIFF). A l'origine, dans Csound, cette valeur était toujours 32767, correspondant à l'étendue bipolaire d'un fichier son 16 bit ou d'un codec AN/NA 16 bit, les seules sorties possibles de Csound à l'époque. Ceci reste l'amplitude de crête *par défaut* dans Csound, pour une compatibilité descendante et vous verrez que certains des exemples de ce manuel utilisent toujours cette valeur (c'est pourquoi l'on trouve de grandes valeurs d'amplitude comme 10000).

La valeur *0dbfs* permet à Csound de produire des valeurs convenablement calibrées quelque soit le format utilisé, entiers sur 24 bit, nombres en virgule flottante sur 32 bit, ou même entiers sur 32 bit. Autrement dit, les valeurs d'amplitude littérales écrites dans un instrument de Csound ne concordent avec celles qui sont écrites *littéralement* dans le fichier que si la valeur *0dbfs* dans Csound correspond exactement à celle du format d'échantillonnage de la sortie. La conséquence de cette approche est que l'on peut écrire une pièce avec une certaine amplitude et en avoir une restitution correcte et identique (sans tenir compte bien sûr de la gamme dynamique meilleure des formats en haute résolution) qu'elle soit écrite dans un fichier de nombres entiers ou en virgule flottante, ou rendue en temps réel.



### Note

La seule exception à ceci se produit si l'on choisit d'écrire dans un format de fichier "brut" (sans en-tête). Dans de tels cas la valeur interne *0dbfs* est sans signification, et quelques soient les valeurs utilisées, elles sont écrites inchangées. Cela permet de faire générer ou traiter par Csound des données arbitraires. C'est une chose relativement exotique à faire, mais certains utilisateurs en ont besoin.

Vous pouvez choisir de redéfinir la valeur *0dbfs* dans l'en-tête de l'orchestre, par pure commodité ou selon vos préférences. Beaucoup de personnes choisiront 1,0 (le standard pour SAOL, d'autres logiciels comme Pure Date, et pour beaucoup de greffons standard comme VST, LADSPA, CoreAudio AudioUnits, etc), mais n'importe quelle valeur est possible.

Le facteur commun dans la définition des amplitudes est l'échelle en décibel (dB), avec  $0\text{dB}_{\text{FS}}$  toujours compris comme la crête numérique ; ainsi "*0dbfs*" veut dire valeur de "*0dB Full-Scale*" (sur l'étendue complète). Cette mesure est différentes des valeurs d'amplitude réelles, puisque celles-ci sont sur une échelle linéaire qui montre l'oscillation réelle autour de 0, et peuvent ainsi être positives ou négatives. Les valeurs en décibel forment une échelle logarithmique absolue, mais peuvent être également utiles pour la plupart des opcodes. On peut convertir les amplitudes de et en décibel en utilisant les fonctions *ampdb*, *ampdbfs*, *dbamp* et *dbfsamp*. De cette manière, Csound permet au programmeur d'exprimer toutes les amplitudes en dB - les amplitudes plus faibles seront alors représentées par des valeurs de décibel négatives. Cela reflète les pratiques de l'industrie (par exemple sur les indicateurs de niveau des tables de mixage, etc).

Par exemple le même niveau de -6dB (la moitié de l'amplitude) ou de -20dB représentent une amplitude linéaire par rapport à *0dbfs* comme ceci :

**Tableau 2.  $\text{dB}_{\text{FS}}$  en relation avec l'amplitude**

$\text{dB}_{\text{FS}}$	<i>0dbfs</i> = 32767 (par défaut)	<i>0dbfs</i> = 1	<i>0dbfs</i> = 1000 (inhabituel)
0 dB	32767	1	1000

<b>dB<sub>FS</sub></b>	<b>0dbfs = 32767 (par défaut)</b>	<b>0dbfs = 1</b>	<b>0dbfs = 1000 (inhabituel)</b>
-6 dB	16384	0.5	500
-20 dB	3276.7	0.1	100

Certains utilisateurs de Csound peuvent ainsi avoir l'intention d'exprimer tous les niveaux en dB<sub>FS</sub>, et éviter toute confusion ou toute ambiguïté de niveau qui pourrait autrement se produire lorsque des valeurs explicites d'amplitude sont utilisées. L'échelle en décibel reflète la réponse de l'oreille assez fidèlement, et si vous voulez exprimer un niveau vraiment doux, il peut être plus facile et plus expressif d'écrire "-46dB" que "0.005" ou "163.8".

La raison d'utiliser 0dbfs est très simple : la crête numérique est égale au niveau maximum quelque soit la résolution de l'échantillonnage. Si vous définissez un signal à -110dB, il disparaîtra s'il est restitué dans un fichier 16 bit, mais il restera (audible ou non) s'il est restitué en 24 bit ou mieux. Autrement dit, il y a un plafond fixe mais un plancher mobile - vous pouvez définir des sons aussi doux que vous le voulez (par exemple des queues d'enveloppe), de manière prévisible, et les préserver ou non (sans changer le code de l'orchestre), selon la résolution de leur restitution (dans un fichier ou sur une e/s audio).



## Une note sur l'amplitude numérique, les décibels et l'étendue dynamique

Une approche commode de l'étendue dynamique pour une certaine précision numérique est de calculer l'intervalle en décibels entre la valeur minimale et la valeur maximale pour un échantillon. En général, 1 bit (doublement du niveau) vaut 6dB, donc 16 bit = 96dB.

Ceci n'est pas entièrement exact car les valeurs des échantillons audio sont représentées sur une échelle bipolaire avec des valeurs positives et négatives, et un bit est utilisé pour le signe. Ainsi, puisque les échantillons en entiers sur 16 bit utilisent 1 bit pour le signe et 15 bit pour la valeur, l'intervalle dynamique est de 90dB.

## Audio en temps réel

L'information suivante concerne en premier lieu l'utilisation de csound à partir de la ligne de commande. Les frontaux implémentent ces caractéristiques de différentes manières, mais leur connaissance est nécessaire dans certains d'entre eux.

Les options *-i* et *-o* sont utilisées pour spécifier une sortie en temps réel à la place de l'habituelle sortie différée dans un fichier. On utilise *-o dac* pour la sortie en temps réel et *-i adc* pour l'entrée en temps réel. Naturellement, on peut utiliser l'un ou les deux selon les possibilités matérielles. On peut aussi spécifier le matériel à utiliser en ajoutant un numéro ou un nom de périphérique au drapeau (voir *-i* et *-o*).

Il peut aussi être nécessaire d'utiliser l'option *-+rtaudio* pour spécifier le pilote d'interface à utiliser. Csound utilise Portaudio par défaut, qui est multi plates-formes et fiable, mais, pour obtenir de meilleures performances, on peut utiliser ALSA et JACK sur linux, et CoreAudio sur Mac. On peut utiliser ASIO sur Windows si la version de Portaudio a été compilée avec le support ASIO.

On peut voir une liste des périphériques disponibles en donnant un numéro de périphérique trop grand, par exemple *-o dac99*. Si vous utilisez Portaudio, ceci indiquera également si ASIO est disponible.

## Tailles de période et de tampon

Les tailles de période et de tampon varient beaucoup d'une machine à l'autre. Plus la taille du tampon est petite et plus la latence est courte, mais cela peut causer des interruptions et des clics dans le flux audio.

Les options Csound qui contrôlent les tailles de période et de tampon sont respectivement *-b* et *-B*. La taille de tampon dépend du matériel, et des essais peuvent être nécessaires pour trouver l'équilibre optimal entre une faible latence et un flux audio continu. Les valeurs données à *-b* et *-B* doivent être des puissances de deux, et la valeur de *-B* doit surpasser celle de *-b* d'au moins une puissance de deux.

Actuellement, avec *-B* fixé à 512, la latence de la sortie audio est d'environ 12 millisecondes, suffisamment rapide pour un jeu au clavier raisonnablement réactif. On peut même obtenir des latences plus courtes sur certains systèmes.

## Cadence de contrôle

De faibles valeurs de *ksmps* donneront en général une synthèse de meilleure qualité, mais consommeront plus de ressources système. Il n'y a pas de règle absolue pour fixer *ksmps* - différents orchestres nécessiteront différentes cadences de contrôle. Un instrument à guide d'onde nécessitera une valeur de *ksmps* de 1 (et pourra ne pas convenir au temps réel), alors qu'une simple synthèse FM pourra fonctionner avec de plus grandes valeurs de *ksmps* sans dégradation notable du son. Si cette synthèse FM doit jouer une ligne de basse monodique, on peut utiliser une très faible valeur de *ksmps*, cependant des clusters de notes plus complexes nécessiteront une valeur de *ksmps* plus grande. Un système linux bien réglé devrait même être capable de produire des synthèses polyphoniques complexes avec des valeurs de *ksmps* aussi faibles que 4 ou 8. Si l'on a besoin de capacités audio duplex complètes, *-b* doit être un multiple entier de *ksmps*. En gardant cela à l'esprit, on peut poser comme règle empirique de n'utiliser que des puissances de deux pour *ksmps*.

Certains réglages diffèrent selon la plate-forme. Voir la suite pour de l'informations sur chaque plate-forme.

## Entrées/sorties en temps réel sous Linux

Sous linux, les réglages portaudio/portmidi par défaut provoquent une latence plus longue que celle que l'on obtiendrait avec ALSA et/ou JACK (voir la section séparée de manuel sur le sujet). Les greffons portaudio/portmidi sont des serveurs audio et MIDI, qui fournissent une interface aux pilotes ALSA, tout comme le fait JACK, mais d'une manière fondamentalement différente.

## Utilisation d'ALSA

Le plus haut niveau de contrôle et la plus faible latence possible sont atteints en utilisant les greffons ALSA en combinaison avec l'option *--sched*. L'utilisation de *--sched* nécessite que Csound soit lancé par l'utilisateur root, ce qui peut être impossible ou indésirable dans certaines circonstances.

Les greffons ALSA nécessitent le nom ("*name*") d'une carte ("*card*") et d'un périphérique ("*device*"). A moins d'avoir nommé vos cartes dans *~/asoundrc* (ou */etc/asound.conf*), les noms seront en fait des nombres. Pour obtenir une liste des configurations possibles, utilisez les utilitaires en ligne de commande "*aplay*", "*arecord*" et "*amidi*". Ces utilitaires sont inclus dans la plupart des distributions Linux, ou peuvent être téléchargés et construits à partir de ces sources :

<ftp://ftp.alsa-project.org/pub/utis/>



### Note

A chaque démarrage de l'ordinateur, la carte son peut avoir un numéro d'ordre matériel différent, surtout s'il y a plusieurs cartes son dans le système. Ceci peut être gênant car il faut chaque fois redonner le bon numéro. On peut affecter un ordre fixe en ajoutant quelques lignes à */etc/modprobe.d/alsa-base-conf*, par exemple pour une carte avec le circuit intégré ice1712 :

```
# ALSA module ordering for soundcard
options snd slots=snd_ice1712
```

## Sortie audio

En tapant la commande suivante :

```
aplay -l
```

vous obtiendrez une liste des périphériques de reproduction audio disponibles sur votre système. Cette liste ressemble à ceci :

```
[...]
**** List of PLAYBACK Hardware Devices ****
card 0: A5451 [ALI 5451], device 0: ALI 5451 [ALI 5451]
[...]
```

Si vous avez plus d'une carte sur votre système, ou s'il y a plus d'un périphérique sur votre carte, la liste sera naturellement plus compliquée, cependant, dans tous les cas, l'information pertinente est le numéro/nom de la carte/périphérique. Afin d'utiliser la carte son ci-dessus pour la sortie audio, il faut ajouter l'option suivante à la ligne de commande Csound, dans ~/.csoundrc, ou dans la section <CsOptions> d'un CSD :

```
++rtaudio=alsa -o dac
```

## Sortie avec dmix

Si vous désirez utiliser Csound avec dmix et que votre carte son ne supporte pas le mixage matériel des flux audio, il faut régler les tampons logiciel (-b) et matériel (-B) avec un soin particulier. Si vous recevez un message du pilote ALSA de Csound qui ressemble à ceci :

```
ALSA: -B 8192 not allowed on this device; use 7526 instead
```

il y a de bonnes chances que vous puissiez utiliser dmix. Si vous utilisez dmix, les réglages de -b et de -B dans Csound doivent être synchronisés avec la taille de période (period\_size) et la taille de tampon (buffer\_size) de dmix respectivement, en utilisant le rapport du taux d'échantillonnage du projet Csound sur le taux d'échantillonnage sur lequel dmix est réglé. Les formules suivantes déterminent les réglages à utiliser pour Csound en fonction des réglages de dmix :

```
-b = (csound_sr/dmix_sample_rate) * dmix_period_size
-B = (csound_sr/dmix_sample_rate) * dmix_buffer_size
```

Par exemple, si dmix est fixé à 48000 échantillons par seconde, un period\_size de 1024, et un buffer\_size de 8192, si l'on exécute un projet Csound avec sr=48000, les réglages des tampons seront "-b 1024 -B8192". Si sr=24000, les réglages des tampons seront "-b 512 -B4096".

A cause de cette relation, si le taux d'échantillonnage du projet Csound ne divise pas exactement le taux d'échantillonnage utilisé par dmix, il pourra être difficile, voire impossible, de régler correctement -b et -B à cause des erreurs d'arrondi. En conséquence, si vous utilisez des taux d'échantillonnage différents que ceux que vous fixez pour dmix, nous vous suggérons de configurer dmix avec un period\_size et un buffer\_size divisibles par le rapport entre le taux d'échantillonnage de csound et celui de dmix. Par exemple, pour exécuter un projet avec sr=16000, les réglages suivants de dmix :

```
pcm.amix {
```

```
type dmix
ipc_key 50557
slave {
    pcm "hw:0,0"
    period_time 0
    #period_size 1024
    #buffer_size 8192
    period_size 1536
    buffer_size 12288
}
bindings {
    0 0
    1 1
}
}

# route ALSA software through pcm.amix
pcm.!default {
    type plug
    slave.pcm "amix"
}
```

avec `period_size=1536` et `buffer_size=12288` seront divisibles par 3 (le rapport du taux d'échantillonnage de csound par celui de dmix) pour obtenir "-b 512 -B4096" ( $(16000/48000) * 1536 = 512$ ,  $(16000/48000) * 12288 = 4096$ ).



### Note

Pour la plupart des cartes son qui sont affectées par ceci, le taux d'échantillonnage par défaut de la carte sera 48000 et ceux de dmix seront 1024 et 8192.

## Entrée audio

Normalement, la même carte étant utilisée pour les entrées et les sorties, en continuant l'exemple précédent, l'option :

```
-i adc:hw:0,0
```

sera ajouté pour l'entrée audio à partir du périphérique 0 de la carte 0. Pour utiliser la carte par défaut, on emploie l'option suivante, mais attention, ça peut ne pas fonctionner :

```
-i adc
```

Si l'on désire utiliser une autre carte ou un autre périphérique pour l'entrée, la commande suivante fournira une liste de périphériques en entrée :

```
arecord -l
```

Si, par exemple, vous désirez utiliser en sortie une interface audio USB, qui est la deuxième "carte" dans votre système, alors que vous désirez utiliser en entrée votre carte son interne, la première carte de votre installation, positionnez les options suivantes à l'endroit adéquat :

```
--rtaudio=alsa -i adc:hw:0,0 -o dac:hw:1,0
```

Si vous désirez utiliser le second périphérique sur votre interface USB, pour envoyer un flux audio à un canal particulier, vous utiliserez les options suivantes :

```
--rtaudio=alsa -i adc:hw:0,0 -o dac:hw:1,1
```

## MIDI

2 pilotes Midi sont disponibles :

- Midi brut.
- Séquenceur Alsa (depuis la version 5.18).

## Entrée MIDI Input (Pilote Midi brut)

Afin de permettre à votre orchestre de recevoir une entrée MIDI vous pouvez utiliser VirMIDI ou MIDI-Thru, selon vos préférences. La configuration de ces ports MIDI virtuels a été largement couverte ailleurs, voir le Linux MIDI how-to [<http://www.midi-howto.com/>] ou parcourez la documentation de votre distribution ou la documentation ALSA à la recherche d'instructions pour installer et configurer VirMidi ou MIDIThru (séquenceur factice). Une fois ceci réalisé, la commande :

```
amidi -l
```

retourne une liste des périphériques disponibles. Cette liste ressemble à ceci :

```
[...]  
Device  Name  
hw:1,0  Virtual Raw MIDI (16 subdevices)  
hw:1,1  Virtual Raw MIDI (16 subdevices)  
hw:1,2  Virtual Raw MIDI (16 subdevices)  
hw:1,3  Virtual Raw MIDI (16 subdevices)  
hw:2,0,0 PCR MIDI  
hw:2,0,1 PCR 1
```

Dans cet exemple, Csound peut se connecter à n'importe lequel des quatre ports virtuels MIDI directs, pour y écouter l'entrée MIDI. L'option suivante indique à Csound d'écouter sur le premier de ces ports :

```
--rtmidi=alsa -Mhw:1,0
```

Il faudra ensuite connecter votre matériel ou votre contrôleur logiciel au port qui accueille votre synthétiseur Csound. La manière la plus simple de le faire est d'employer l'utilitaire "aconect". Tapez :

```
aconect -li
```

pour une liste des périphériques d'entrée disponibles, et :

```
aconect -lo
```

pour une liste des périphériques de sortie disponibles (y compris le port auquel Csound a été connecté). Cette liste ressemble à ceci :

```
#aconect -li  
client 0: 'System' [type=kernel]  
  0 'Timer' '  
  1 'Announce' '  
    Connecting To: 15:0  
client 20: 'Virtual Raw MIDI 1-0' [type=kernel]  
  0 'VirMIDI 1-0' '
```

```
client 21: 'Virtual Raw MIDI 1-1' [type=kernel]
  0 'VirMIDI 1-1'
client 22: 'Virtual Raw MIDI 1-2' [type=kernel]
  0 'VirMIDI 1-2'
client 23: 'Virtual Raw MIDI 1-3' [type=kernel]
  0 'VirMIDI 1-3'
client 24: 'PCR' [type=kernel]
  0 'PCR MIDI'
  1 'PCR 1'
  2 'PCR 2'
```

```
#aconnect -lo
client 20: 'Virtual Raw MIDI 1-0' [type=kernel]
  0 'VirMIDI 1-0'
client 21: 'Virtual Raw MIDI 1-1' [type=kernel]
  0 'VirMIDI 1-1'
client 22: 'Virtual Raw MIDI 1-2' [type=kernel]
  0 'VirMIDI 1-2'
client 23: 'Virtual Raw MIDI 1-3' [type=kernel]
  0 'VirMIDI 1-3'
client 24: 'PCR' [type=kernel]
  0 'PCR MIDI'
  1 'PCR 1'
```

Dans l'exemple suivant, le clavier USB qui est listé ci-dessus comme le client 24 sera connecté au synthétiseur Csound qui est à l'écoute sur le premier port VirMIDI. Le clavier a trois ports de sortie. Le premier (24:0) transmet les messages reçus sur le port d'entrée MIDI, le second (24:1) transmet les messages de touches et de contrôleurs, et le troisième (24:2) transmet les messages système exclusif. La commande suivante connecte le second port du clavier au synthétiseur Csound :

```
aconnect 24:1 20:0
```

Il faut garder à l'esprit que Csound agit comme un périphérique MIDI direct et non comme un client du séquenceur ALSA. Cela signifie que Csound n'apparaîtra pas dans la liste des périphériques MIDI et ne sera pas disponible pour un usage direct avec *aconnect*, ainsi, il faut se connecter à un périphérique virtuel (comme 'virtual raw MIDI' ou 'MIDI through') pour des connexions persistantes, ou se connecter directement à la destination en utilisant les options de ligne de commande.

## Sortie MIDI (pilote MIDI brut)

On peut connecter Csound à n'importe quel périphérique qui apparaît dans la liste des ports de sortie du séquenceur ALSA, que l'on obtient par "amidi -l" comme ci-dessus. Afin de connecter un synthétiseur Csound au port MIDI out du clavier listé ci-dessus, on utilise l'option suivante :

```
-Qhw: 2, 0, 0
```

## Entrée et sortie MIDI (Pilote du séquenceur Midi)

Ce pilote est préférable au pilote MIDI brut. Il présente ces avantages :

- Accès concurrents multiples.
- Partages par files d'attente prioritaires.

- Envoi d'évènements en temps réel, le rôle du séquenceur MIDI étant de délivrer des évènements au bon moment (séquence) et au bon destinataire (périphérique).

La commande suivante appelle le séquenceur MIDI. Ici, il écoute sur le port 20. Le port de sortie MIDI est aussi le 20 :

```
--rtmidi=alsaseq -M20 -Q20
```

Csound crée automatiquement son propre port de séquenceur ALSA. La commande suivante donne une liste des périphériques disponibles :

```
aconnect -i -o
```

Elle retourne une sortie ressemblant à ceci :

```
client 0: 'System' [type=kernel]
  0 'Timer'
  1 'Announce'
client 14: 'Midi Through' [type=kernel]
  0 'Midi Through Port-0'
client 20: 'M Audio Delta 1010' [type=kernel]
  0 'M Audio Delta 1010 MIDI'
client 130: 'Csound' [type=user]
  0 'Csound'
```

La sortie de Csound contiendra des lignes comme :

```
.....
ALSASEQ: opened MIDI output sequencer
ALSASEQ: created output port 'Csound' 130:0
ALSASEQ: connected to 20:0
.....
.....
ALSASEQ: opened MIDI input sequencer
ALSASEQ: created input port 'Csound' 130:0
ALSASEQ: connected from 20:0
.....
```

## Temps-partagé

Si vous avez la possibilité d'exécuter Csound en tant qu'utilisateur root, l'option "--sched" permet d'améliorer spectaculairement les performances en temps réel avec ALSA, cependant vous pouvez bloquer le système si vous faites quelque chose de stupide. N'UTILISEZ PAS "--sched" si vous choisissez JACK pour la sortie audio. JACK contrôle le temps-partagé pour les applications audio qui l'utilisent, et il essaie également de fonctionner avec la priorité maximale. Si l'option "--sched" est utilisée, Csound et JACK vont entrer en compétition au lieu de coopérer, ce qui aura pour résultat de piètres performances.

## Utilisation de pulseaudio

Le support de Pulseaudio [<http://www.pulseaudio.org/>] a été ajouté dans Csound 5.09. Vous pouvez spécifier les réglages suivants :



1. Noms de sortie : il est possible d'utiliser un nombre à la place du nom complet, ainsi `-odac:1` sélectionne votre second périphérique (le compte commence à 0).
2. Nom du serveur : il est possible de se connecter à un serveur spécifique en utilisant `++server=<server_string>` où `server_string` est le nom d'un serveur ou une chaîne plus complexe de sélection de serveur (voir [pulseaudio.org](http://www.pulseaudio.org) [<http://www.pulseaudio.org/>] sur les chaînes de serveur). Ceci est transparent sur un réseau et permet les connexions à des machines distantes.
3. Noms de flot : il est possible d'étiqueter les flots générés par `csound`, en utilisant `++output_stream=<stream-name>` et `++input_stream=<stream-name>`

Voici un exemple de ligne de commande :

```
csound -odac:1 examples/trapped.csd ++rtaudio=pulse ++server=unix:/tmp/pulse-victor/native ++output_stream=trapped
```

## Mac OSX

### Real-time Audio

Sur OSX on peut utiliser *PortAudio* (par défaut), *auhal* (ou *coreaudio*) ou bien les modules audio temps réel de *Jack*. Le module *auhal* est un module natif d'OSX avec un faible délai de latence, mais il peut ne pas fonctionner avec certains équipements externes. Le module de *Jack* peut être interconnecté avec d'autres applications, mais il nécessite l'installation de *JackOSX* pour fonctionner. Pour activer un module temps réel, on peut utiliser l'option `++rtaudio` avec la valeur *portaudio*, *auhal*, ou *jack*. La valeur *portaudio* est activée par défaut si l'on indique pas de valeur.

Il faut aussi indiquer le périphérique son que l'on veut utiliser et spécifier que l'on veut générer une sortie audio en temps réel au lieu d'une sortie disque. Pour ce faire on doit utiliser l'option `-odac` ou `-o dac` qui dit à *Csound* d'envoyer sa sortie vers les convertisseurs numériques analogiques plutôt que dans un fichier. En ajoutant un nombre après l'option (par exemple `-odac2`), on peut choisir le numéro du périphérique que l'on veut utiliser. Pour trouver les périphériques du système disponibles, on utilise un grand nombre hors limites (par exemple `-odac99`), et *Csound* rapportera une erreur ainsi que la liste des périphériques disponibles. Cette convention de numérotation fonctionne avec *portaudio* et *auhal*, mais pour *Jack*, on doit passer le nom de la sortie désirée précédée de deux points (par exemple `-odac:system:playback_`).

L'entrée audio en temps réel est activée par `-iadc`. *Csound* écoute alors sur les entrées audio en temps réel. On peut à nouveau sélectionner le périphérique par son numéro (ou son nom), et tester les périphériques disponibles en utilisant un nombre hors limites. Noter que pour les entrées on utilise 'adc' au lieu de 'dac'. Il faut s'assurer que l'entrée correspondante est sélectionnée dans le panneau de contrôle de la carte son.

### MIDI en temps réel

Pour activer le MIDI en temps réel sur OSX, on utilise l'option `-M` pour l'entrée MIDI et l'option `-Q` pour la sortie MIDI. Il peut être nécessaire d'indiquer un numéro de périphérique après l'option (par exemple `-M2`), et là encore, on peut trouver les périphériques disponibles en donnant un nombre hors limites.

*Csound* utilise par défaut le module *PortMidi*, mais il y a aussi un module *coremidi* natif que l'on peut activer avec l'option :

```
++rtmidi=cmidi
```

Le module *coremidi* ne supporte actuellement que l'entrée MIDI.

Un ensemble d'options typique pour activer les entrée-sorties audio et MIDI en temps réel ressemble à ceci :

```
--rtmidi=cmidi -M1 --rtaudio=auhal-odac3 -iadc3
```

## Windows

### Audio en temps réel

Les utilisateurs de Windows peuvent utiliser soit le module temps réel par défaut *PortAudio*, soit le module temps réel *winmm*. Le module *winmm* est un module natif de Windows qui fournit une grande stabilité, mais une latence qui sera en général trop grande pour une interaction en temps réel. Pour activer un module temps réel on peut utiliser l'option *--rtaudio* avec la valeur *portaudio* ou *winmm*. La valeur par défaut est *portaudio*, qui est active sans avoir à être spécifiée.

On doit aussi spécifier le périphérique son que l'on veut utiliser, et indiquer que l'on veut générer une sortie audio en temps réel plutôt qu'un fichier son vers une sortie disque. Pour cela, on doit utiliser l'option *-odac* ou *-o dac*, qui indique comme sortie de *csound* les convertisseurs Numérique-Analogique plutôt qu'un fichier. En ajoutant un numéro après l'option (par exemple *-odac2*), on peut choisir le numéro du périphérique désiré. Pour trouver les périphériques disponibles dans le système, on peut utiliser un numéro trop grand (par exemple *-odac99*), et *csound* rapportera une erreur ainsi que la liste des périphériques disponibles.

Lorsque l'on choisit le numéro de périphérique sous *Portaudio*, on choisit également l'interface du pilote, car *Portaudio* supporte *WinMME*, *DirectX* et *ASIO*. Si vous avez une interface compatible *ASIO* ou un émulateur de pilote *ASIO* comme *ASIO4ALL* [<http://www.asio4all.com>], le périphérique affichera plusieurs durées, une pour chaque interface de pilote. Comme *ASIO* fournit la meilleure latence pour un système, il devrait être choisi pour une sortie audio en temps réel s'il est disponible.

On active l'entrée audio en temps réel par *-iadc*, ce qui règle *csound* sur l'écoute de l'entrée audio temps réel. On peut également choisir le périphérique par son numéro, et tester les périphériques disponibles avec un numéro trop grand. Notez que pour les entrées on utilise 'adc' au lieu de 'dac'. Assurez-vous que la bonne entrée soit sélectionnée dans le panneau de contrôle de votre carte son.

### MIDI en temps réel

Pour activer le MIDI en temps réel dans Windows on peut utiliser l'option *-M* pour l'entrée MIDI et l'option *-Q* pour la sortie MIDI. On peut spécifier le numéro du périphérique après le drapeau (par exemple *-M2*), et aussi trouver les périphériques disponibles en donnant un numéro trop grand.

*Csound* utilise par défaut le module MIDI *PortMidi*, mais il y a aussi un module natif *winmm*, que l'on peut activer avec l'option :

```
--rtmidi=winmm
```

Un ensemble d'options typique pour activer l'Audio et les E/S MIDI en temps réel ressemblera à ceci:

```
--rtmidi=winmm -M1 -Q1 --rtaudio=portaudio -odac3 -iadc3
```

# Entrées/sorties en temps réel avec le kit de connexion de JACK

On peut utiliser sur plusieurs systèmes le kit de connexion de JACK pour les entrées/sorties audio ainsi que MIDI. Pour plus de détails, voir

<http://jackaudio.org/faq>

## Utilisation de JACK

La manière la plus simple d'utiliser le greffon JACK pour l'entrée et la sortie est la suivante :

```
--rtaudio=jack -i adc -o dac
```

De plus, il y a certaines options de ligne de commande spécifiques à JACK :

### Options de ligne de commande pour JACK

<code>--jack_client=[nom_de_client]</code>	Le nom de client utilisé par Csound, par défaut "csound6". Si plusieurs instances de Csound se connectent au serveur JACK, il faut utiliser différents nom de client pour éviter les conflits de nom.
<code>--jack_inportname=[préfixe de nom de port en entrée], --jack_outportname=[préfixe de nom de port en sortie]</code>	Préfixe pour les noms des ports d'entrée/sortie JACK de Csound ; les valeurs par défaut sont "input" et "output". Le nom de port courant est constitué de numéro de canal ajouté au préfixe. Exemple : avec les réglages par défaut ci-dessus, un orchestre stéréo créera ces ports en opération full duplex :

<code>csound6:input1</code>	<code>(record left)</code>
<code>csound6:input2</code>	<code>(record right)</code>
<code>csound6:output1</code>	<code>(playback left)</code>
<code>csound6:output2</code>	<code>(playback right)</code>

## Connexion de Csound à d'autres clients JACK

Par défaut, les connexions sont faites sur les premiers ports de la liste de ports de JACK (généralement ceux-ci correspondent par défaut aux ports physiques). Le greffon peut se connecter à des ports spécifiés par des noms ou des numéros.

Par des noms : le préfixe du nom de port est utilisé, par exemple "system:playback\_", "system:capture\_", "alsa\_pcm:playback\_" ou "alsa\_pcm:capture\_" ce qui donne -odac:system:playback\_ (pour la sortie audio), -iadc:system:capture\_ (pour l'entrée audio). Le préfixe du nom de port exclut les noms de canal.

Par des numéros : le numéro du port de base est donné, la connexion étant faite vers ce port et les suivants en fonction du nombre de canaux. Pour un numéro de port de base N nous aurons des connexions en N +0, N+1, ..., N+nchnls-1. Par exemple -odac2 avec nchnls=2 connectera les sorties sur les ports 2 et 3. Les ports sont listés comme dans les autres modules d'entrée/sortie.

Noter qu'on peut empêcher la connexion automatique des ports en donnant comme options de sortie et d'entrée -odac:null et -iadc:null. On peut aussi réaliser les connexions manuellement avec les outils de connexion de JACK.

## Notes sur les tailles de tampon

Les données audio sont reçues de et envoyées vers le serveur JACK par Csound en utilisant un tampon circulaire contrôlé par les options `-b` et `-B`. `-B` est la taille totale du tampon alors que `-b` est la taille d'une seule période. Ces valeurs sont arrondies afin que la taille totale soit un multiple entier de la taille d'une période et lui soit supérieure. La différence entre la taille du tampon de Csound et la taille d'une période doit être supérieure ou égale à la taille d'une période de JACK.

Si on utilise à la fois `-iadc` et `-odac`, l'option `-b` doit être un multiple entier de `ksmps`.

Exemple de réglages du tampon pour une faible latence sur un système linux rapide :

```
jackd -d alsa -P -r 48000 -p 64 -n 4 -zt &  
csound -+rtaudio=jack -b 64 -B 256 [...]
```

avec ordonnancement temps réel (en tant que root) :

```
jackd -R -P 90 -d alsa -P -r 48000 -p 64 -n 2 -zt &  
csound --sched=80,90,10 -d -+rtaudio=jack -b 64 -B 192 [...]
```

Pour améliorer l'exécution, utiliser des valeurs de `ksmps` comme 32 et 64.

Le taux d'échantillonnage de l'orchestre doit être le même que celui du serveur JACK.

On peut aussi utiliser JACK pour les entrées/sorties MIDI. Pour cela, on indique `--rtmidi=jack`. On utilise pour l'entrée `-M` suivi du nom du port MIDI de JACK requis pour se connecter directement à un flot d'entrée. Pour la sortie on utilise `-Q` suivi du nom du port JACK. Les options `--jack_midi_inportname=` et `--jack_midi_outportname=` peuvent être utilisées pour renommer les ports d'E/S MIDI de Csound.

## Optimisation de la latence audio en E/S

Pour atteindre la latence la plus basse possible sans interruptions audio, il faut régler une combinaison de variables. Les valeurs retenues dépendront de la plate-forme et du système, et aussi de la complexité des calculs audio mis en œuvre. Il faut ajuster `ksmps` dans l'orchestre, ainsi que la taille du tampon logiciel (`-b`) et celle du tampon matériel (`-B`).

Habituellement la solution la plus simple est la suivante :

1. Fixer `ksmps` à une valeur de compromis entre qualité et performance, sans ajuster `-B` du tout.
2. Fixer `-b` à une puissance de deux négative.

Pour obtenir les valeurs optimales, commencer avec une valeur qui vous semble trop petite, c'est-à-dire `-1`, et continuer ensuite en "augmentant", `-2`, `-4`, etc., jusqu'à ne plus avoir de défauts dans le son. La valeur réelle de `-b` sera la valeur absolue de `-b * ksmps`.

3. Réduire le tampon matériel (`-B`). Partir de la valeur par défaut (1024 sur Linux, 4096 sur Mac OS X, 16384 sur Windows), et la réduire de moitié à chaque fois, jusqu'à entendre à nouveau des défauts. La remonter alors jusqu'à ce que l'exécution soit continue.

Cette procédure s'applique aux cartes 16 bit. Si vous avez une carte son 24 bit, alors `-B` doit valoir  $3/2$ , ou 3 fois `-b`, plutôt que 2 ou 4 fois. Csound travaille avec des nombres en virgule flottante en 32 bit ou 64 bit alors que la plupart des cartes son utilisent des entiers en 16 ou 24 bit. `-b` est le tampon interne, c'est pourquoi il traite de la partie 32 ou 64 bit, tandis que `-B` est le tampon matériel, et il traite ainsi de la partie 16 ou 24 bit. Le réglage par défaut de csound pour les réels est `-B = 4 * -b`. C'est une valeur sûre pour

une carte 16 bit. On peut s'en sortir avec  $-B = 2 * -b$ , mais c'est le minimum absolu. Par exemple, si votre réglage est *-b1024 -B2048*, csound vous dira ceci :

```
audio buffered in 1024 sample-frame blocks writing 4096-byte blocks to dac
```

4096 octets font 32768 bits.  $32768/32 = 1024$ , notre taille de bloc de trames d'échantillons,  $1024 * 32/16 = 2048$ , notre taille de tampon. Si nous réduisons la valeur de  $-B$ , il faudra réduire la valeur de  $-b$  d'un montant proportionnel afin de continuer à écrire des entiers en 16 bit sur le CNA. La taille minimale de  $-b$  est  $(-B * \text{bitrate})/32$ . Cela veut dire que le rapport minimum de  $-b$  à  $-B$  doit être :

- 1/2 en 16 bit
- 2/3 en 24 bit
- 1/1 en 32 bit

Bien qu'il n'y ait théoriquement pas de rapport maximum, il n'y a aucun sens à avoir un rapport très élevé ici, car le tampon logiciel doit remplir le tampon matériel avant de retourner. Si le rapport est élevé, cela prendra plus de temps, annulant le bénéfice de mettre une petite valeur pour  $-b$ .

Il faudra varier la valeur de  $-b$  en fonction de la complexité de l'instrument sur lequel vous travaillez, mais comme elle est intimement liée à celle de *ksmps*, il vaut mieux la synchroniser avec *ksmps* et partir de là. Une manière de faire est de décider quelle sera la longueur optimale de la chute de vos enveloppes (pour l'effet désiré), de fixer toutes les enveloppes au maximum, de donner vous-même une valeur généreuse à  $-b$ , et de jouer. S'il y a des interruptions, doubler *ksmps*, et répéter le processus jusqu'à obtenir la fluidité, descendre ensuite la valeur de  $-b$  aussi bas que possible.

La valeur de  $-B$  est d'abord déterminée par le système d'exploitation et la carte son. Essayez de trouver (par la méthode ci-dessus) jusqu'où vous pouvez descendre, et utilisez cette valeur (ou une valeur supérieure par sécurité). Si vous rencontrez des problèmes ce sera probablement à cause d'une valeur de *ksmps* inappropriée, d'une valeur de  $-b$  trop faible, ou de nombres hors-norme (voir *denorm*).

---

# Configuration

Après avoir installé une distribution binaire ou bien avoir construit Csound à partir des sources, il faut configurer Csound afin de l'adapter à votre système. Les installateurs réalisent habituellement ces étapes automatiquement pour vous.

Sur toutes les plates-formes il faut s'assurer que le ou les répertoires contenant les bibliothèques des greffons de Csound sont indiqués dans une variable d'environnement `OPCODE6DIR` ou `OPCODE6DIR64` en fonction de la précision utilisée par les binaires compilés. (Noter que pour Csound5 ces variables d'environnement étaient `OPCODEDIR` et `OPCODEDIR64`).

Les opérateurs Python nécessitent actuellement au moins Python 2.4 que l'on peut télécharger à [www.python.org](http://www.python.org) [http://www.python.org] s'il n'est pas déjà installé sur votre système. On peut tester s'il est installé en tapant 'python' depuis une invite de commande ou une fenêtre DOS.

## Windows

Sur Windows, assurez-vous que le ou les répertoires (normalement le répertoire `C:\Program Files\Csound`) contenant le répertoire des exécutables de Csound est dans votre variable `PATH`, ou bien copiez tous les fichiers exécutables dans le répertoire `system32` de Windows. En fonction de votre méthode d'installation, il peut être aussi nécessaire de fixer les variables d'environnement `OPCODE6DIR` et `OPCODE6DIR64`. En supposant que Csound est installé dans le répertoire par défaut `C:\Program Files\Csound` vous pouvez utiliser (sinon fixez les chemins en conséquence) :

```
set OPCODE6DIR=C:\Program Files\Csound\greffons
set OPCODE6DIR64=C:\Program Files\Csound\greffons64
set PATH=%PATH%;C:\Program Files\Csound\bin
```



### python24.dll ou python25.dll manquante

S'il apparaît une fenêtre pop-up au sujet de la bibliothèque Python manquante (`python24.dll` ou `python25.dll`) et que vous n'avez pas besoin des opérateurs python, effacez simplement `C:\Program Files\Csound\greffons\py.dll` et `C:\Program Files\Csound\greffons64\py.dll`, et la fenêtre pop-up au sujet de la bibliothèque Python manquante ne devrait plus réapparaître

## Unix et Linux

Sur Unix et Linux, installez le programme Csound dans l'un des répertoires `bin` du système, normalement `/usr/local/bin`, et les bibliothèques partagées de Csound et des greffons dans des endroits comme `/usr/local/lib/csound/greffons` ou `/usr/local/lib/csound/greffons64` et assurez-vous que les variables d'environnement `OPCODE6DIR` et `OPCODE6DIR64` sont remplies correctement.

## CsoundAC

CsoundAC nécessite quelques configurations supplémentaires. Sur toutes les plates-formes, CsoundAC nécessite que vous ayez installé Python sur votre ordinateur. Le répertoire contenant la bibliothèque partagée `_CsoundAC` et le fichier `CsoundAC.py` doit être dans votre variable d'environnement `PYTHONPATH`, afin que le runtime Python sache comment charger ces fichiers.

---

# Syntaxe de l'orchestre

L'orchestre Csound (.orc) ou la section `<CsInstruments>` d'un fichier csd, contient :

- Une *section d'en-tête*, qui spécifie les options globales pour l'exécution des instruments.
- Une liste d'*opcodes définis par l'utilisateur (UDO)* et de *blocs d'instrument* contenant les définitions des UDO et des instruments.

L'en-tête de l'orchestre, les blocs d'instrument, et les UDOs contiennent des *instructions d'orchestre*. Dans Csound une *instruction d'orchestre* a le format :

```
étiquette:  résultat opcode argument1, argument2, ... ;commentaires
```

L'étiquette est facultative et identifie l'instruction de base qui suit comme cible potentielle d'une opération goto (voir *contrôle du déroulement du programme*). Une étiquette n'a aucun effet sur l'instruction en soi.

Selon leur fonction, certains opcodes ne produisent pas de sortie et n'ont donc pas de valeur de retour. D'autres ne prennent pas d'argument et produisent seulement un résultat.

Chaque instruction d'orchestre doit tenir sur une seule ligne, cependant les longues lignes peuvent être continuées sur la ligne suivante grâce au caractère '\'. Ce caractère indique que la ligne suivante fait partie de la ligne courante, de façon à pouvoir couper une ligne pour en faciliter la lecture, comme ceci :

```
a2  oscbnk kcps, 1.0, kfmd1, 0.0, 40, 203, 0.1, 0.2, kamfr, kamfr2, 148, \
      0, 0, 0, 0, 0, 0, -1, \
      kfnum, 3, 4
```

Les commentaires sont facultatifs et ils ont pour but de permettre à l'utilisateur de commenter le code de son orchestre. Les commentaires commencent par un point-virgule (;) ou // et s'étendent jusqu'à la fin de la ligne. Les commentaires peuvent optionnellement être écrits en style C, s'étendant sur plusieurs lignes comme ceci :

```
/* Tout ce qui se trouve ici -----
   est un commentaire qui peut couvrir
   plusieurs lignes ----- */
```

Le reste (résultat, opcode, et arguments) forme l'instruction de base. C'est également facultatif, ce qui veut dire qu'une ligne peut n'avoir qu'une étiquette ou un commentaire ou bien être complètement blanche. Si elle est présente, l'instruction de base doit être entièrement contenue dans une ligne, et elle est terminée par un retour chariot et un linefeed.

L'opcode détermine l'opération à effectuer ; habituellement, il prend un certain nombre de valeurs en entrée (ou arguments, au maximum environ 800) ; et il a normalement un champ résultat variable dans lequel il envoie les valeurs de sortie à un certain taux de cadencement fixe. Il y a quatre taux de cadencement possibles :

1. une seule fois, au moment de l'initialisation de l'orchestre (en fait une affectation permanente)
2. une fois au début de chaque note (à la date (init) de l'initialisation : taux-i)
3. à chaque passage dans la boucle de contrôle de l'exécution (taux de contrôle, ou taux-k)
4. à chaque échantillon sonore de chaque boucle de contrôle (taux d'exécution audio, ou taux-a)

## Instructions de l'en-tête de l'orchestre

L'*en-tête de l'orchestre* contient l'information globale qui s'applique à tous les instruments et qui définit les aspects de la sortie de Csound. On y fait parfois référence comme *instr 0*, parce qu'il se comporte comme un instrument, mais sans traitement de taux-k ou de taux-a (seuls les opcodes et les instructions qui fonctionnent au taux-i y sont autorisés).

Une *instruction d'en-tête d'orchestre* n'opère qu'une fois, à l'initialisation de l'orchestre. La plupart du temps il s'agit de l'affectation d'une valeur à un *symbole global réservé*, par exemple `sr = 20000`. Toutes les instructions d'en-tête d'orchestre appartiennent au pseudo instrument 0, dont un passage *init* est effectué avant tout autre instrument au temps 0 de la partition. Toute *instruction ordinaire* peut servir d'instruction d'en-tête d'orchestre, par exemple `gfreq = cpspch(8.09)` à condition d'être seulement une opération du moment d'initialisation. Les instructions placées normalement dans un en-tête d'orchestre sont :

- *odbfs*
- *A4*
- *ctrlinit*
- *ftgen*
- *kr*
- *ksmps*
- *massign*
- *nchnls*
- *pgmassign*
- *pset*
- *seed*
- *sr*
- *strset*

Par exemple, un en-tête de Csound peut ressembler à ceci :

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
odbfs = 1

massign 1, 10
```

## Instructions de bloc d'instrument et d'opcode

Un bloc d'instrument comprend des *instructions ordinaires* qui fixent des valeurs, contrôlent le déroulement logique, ou appellent les différents sous-programmes de traitement du signal qui mènent à la sortie audio. Les instructions qui définissent un bloc d'instrument sont :

- *instr*



- *endin*

Un bloc d'instrument ressemble à ceci :

```
instr 1 ; Un simple oscillateur sinusoïdal
aout oscils 10000, 440, 0
out aout
endin
```

Les instructions qui définissent un bloc d'opcode défini par l'utilisateur (UDO) sont :

- *opcode*
- *endop*

voir la section *UDO* pour plus d'information.

## Instructions ordinaires

On utilise une *instruction ordinaire* soit lors de l'initialisation soit pendant l'exécution soit durant les deux. Les opérations qui produisent un résultat fonctionnent formellement au taux de ce résultat (c'est-à-dire, pendant l'initialisation pour les résultats de taux-i ; pendant l'exécution pour les résultats de taux-k et de taux-a), avec pour seule exception l'opcode *init*. Cependant, la plupart des générateurs et des modificateurs produisent des signaux que ne dépendent pas seulement de la valeur instantanée de leurs arguments mais aussi d'un état interne conservé. Ainsi, ces unités de la période d'exécution ont un composant implicite de la période d'initialisation pour créer cet état. Le type temporel d'une opération qui ne produit pas de résultat est apparent dans l'opcode.

Les arguments sont des valeurs qui sont envoyées à une opération. La plupart des arguments accepteront des expressions arithmétiques composées de constantes, de variables, de symboles réservés, de convertisseurs de valeur, d'opérations arithmétiques, et de valeurs conditionnelles.

## Types, constantes et variables

Les *constantes* sont des nombres en virgule flottante tels que 1, 3.14159 ou -73.45. Elles sont constamment disponibles et leur valeur ne change pas.

Les *variables* sont des cellules nommées contenant des nombres. Elles sont constamment disponibles et peuvent être mises à jour à l'un des quatre taux de mise à jour (initialisation seulement, taux-i, taux-k, taux-a). Les variables de taux-i et de taux-k sont scalaires (c'est-à-dire qu'elles ne peuvent prendre qu'une valeur à la fois) et sont utilisées principalement pour stocker et rappeler des données de contrôle, données qui changent au rythme des notes (pour les variables de taux-i) ou au taux de contrôle (pour les variables de taux-k). Les i- et les k-variables sont ainsi utiles pour stocker les valeurs des paramètres de note, hauteurs, durées, fréquences variant lentement, vibratos, etc. D'un autre côté, les variables de taux-a sont des tableaux ou vecteurs d'information. Bien que rafraichies pendant le même passage de contrôle de la période d'exécution que les variables de taux-k, ces cellules de tableau représentent une résolution temporelle plus fine en divisant la période de contrôle en durées d'échantillons (voir *ksmps*). Les variables de taux-a sont utilisées pour stocker et rappeler des données qui changent au taux d'échantillonnage audio (par exemple les signaux de sortie des oscillateurs, des filtres, etc.).

Certains types de variables peuvent être considérés comme des signaux. Par exemple les variables de taux-a et de taux-k sont des signaux qui ont une fréquence de mise à jour constante (voir *kr* et *sr*). Cette abstraction est en général assez utile, mais il faut être conscient que les signaux de taux-a sont en fait des vecteurs qui sont traités au taux-k, c'est-à-dire que Csound travaille en interne au taux-k mais qu'il traite *ksmps* échantillons numériques pour chaque variable de taux-a à chaque cycle de contrôle.

Il y a d'autres types de signaux qui nécessitent des taux qui ne concordent pas avec *kr* ou *sr*. Les signaux de taux-f et de taux-w sont utilisés pour le traitement spectral et leur taux est déterminé par la taille de fenêtre et le facteur de recouvrement.

On distingue également les variables locales des variables globales. Les variables *locales* sont privées dans un instrument, et un autre instrument ne peut y accéder ni en lecture ni en écriture. Leurs valeurs sont conservées, et leur information est reportée de passage en passage (par exemple de la période d'initialisation à la période d'exécution) à l'intérieur d'un instrument. Les noms de variable locale commencent par la lettre *p*, *i*, *k*, ou *a*. Le même nom de variable locale peut apparaître dans plusieurs blocs d'instrument différents sans conflit.

Les variables *globales* sont des cellules qui sont accessibles par tous les instruments. Leurs noms sont formés soit comme les noms locaux précédés de la lettre *g*, soit de symboles réservés spéciaux. Les variables globales sont utilisées pour diffuser des valeurs générales, pour la communication entre instruments (sémaphores), ou pour envoyer un son d'un instrument à l'autre (par exemple un mixage avant une réverbération).

Etant données ces distinctions, il y a neuf formes de variables locales et globales :

**Tableau 3. Types de variables**

Type	Renouvellement	Local	Global
symboles réservés	permanent	--	rsymbole
p-champs de partition	temps-i	p nombre	--
variables d'initialisation	temps-i	i nom	gi nom
signaux de contrôle	temps-p, taux-k	k nom	gk nom
signaux audio	temps-p, taux-k (tous les échantillons audio dans une passe-k)	a nom	ga nom
types de données spectrales	taux-k	w nom	--
flots de données spectrales	taux-k	f nom	gf nom
variables chaînes	temps-i et optionnellement temps-k	S nom	gS nom
variables vecteur	taux-k	t nom	--

où *rsymbole* est un symbole réservé spécial (par exemple *sr*, *kr*), *nombre* est un entier positif faisant référence à un p-champ de partition ou à un numéro de séquence, et *nom* est une chaîne composée de lettres, du caractère de soulignement, et/ou de chiffres, avec une signification locale ou globale. Comme on peut le voir, les paramètres de partition sont des variables de taux-i dont les valeurs sont copiées à partir de l'instruction de partition appelante juste avant la passe d'initialisation d'un instrument, tandis que les contrôleurs MIDI sont des variables que l'on peut mettre à jour de manière asynchrone depuis un fichier MIDI ou un périphérique MIDI.

## Initialisation de variable

Les opcodes qui permettent l'initialisation de variable sont :

- *assign*
- *divz*

- *init*
- *tival*

## Macros de constantes mathématiques prédéfinies

Csound définit plusieurs constantes mathématiques importantes par des *macros*. On peut consulter la liste complète *ici*.

## Expressions

On peut composer des expressions de n'importe quelle profondeur. Chaque partie d'une expression est évaluée à son propre taux. Par exemple, si tous les termes d'une sous-expression changent au taux de contrôle ou plus lentement, cette sous-expression ne sera évaluée qu'au taux de contrôle ; le résultat peut alors être utilisé dans une évaluation au taux audio. Par exemple, dans

```
k1 + abs(int(p5) + frac(p5) * 100/12 + sqrt(k1))
```

100/12 sera évalué à l'initialisation de l'orchestre, les expressions en p5 seront évaluées à l'initialisation de la note, et le reste de l'expression à chaque période-k. Le tout pourrait apparaître en position d'argument dans un générateur unitaire, ou bien faire partie d'une instruction d'affectation.

## Répertoires et fichiers

Plusieurs générateurs et la commande Csound elle-même spécifient des noms de fichier pour l'écriture ou la lecture. Ceux-ci peuvent parfois être des chemins complets, dont le répertoire cible est complètement spécifié. Lorsque le chemin n'est pas complet, les noms de fichiers sont recherchés dans plusieurs répertoires dans un ordre dépendant de leur type et de la valeur de certaines variables d'environnement. Ces dernières sont facultatives, mais elles peuvent servir à partitionner et à organiser les répertoires de façon à partager les fichiers plutôt que de les dupliquer dans plusieurs répertoires de l'utilisateur. Les variables d'environnement peuvent définir des répertoires pour les fichiers son (*SFDIR*), les sons échantillonnés (*SSDIR*), les analyses de son (*SADIR*), et les fichiers à inclure pour l'orchestre et la partition (*INCDIR*).

A partir de la version 5.00 de Csound, ces variables d'environnement peuvent spécifier plusieurs répertoires dans une liste dont le séparateur est le point-virgule (;). Si un fichier est trouvé à plusieurs endroits, c'est le premier de ceux-ci qui a la priorité.

L'ordre de recherche est :

1. Les fichiers son en écriture sont placés dans *SFDIR* (s'il existe), sinon dans le répertoire courant.
2. Les fichiers son en lecture sont recherchés dans le répertoire courant. Si les chemins par défaut ne sont pas désactivés, les fichiers sont ensuite recherchés relativement au fichier CSD/ORC/SCO. Enfin, ils sont recherchés dans *SSDIR* puis dans *SFDIR*.
3. Les fichiers de contrôle d'analyse en lecture sont recherchés dans le répertoire courant. Si les chemins par défaut ne sont pas désactivés, les fichiers sont ensuite recherchés relativement au fichier CSD/ORC/SCO. Enfin, ils sont recherchés dans *SADIR*.
4. Les fichiers MIDI en lecture sont recherchés dans le répertoire courant. Si les chemins par défaut ne sont pas désactivés, les fichiers sont ensuite recherchés relativement au fichier CSD/ORC/SCO. Enfin, ils sont recherchés dans *MFDIR*, *SSDIR* et *SFDIR*.

5. Les fichiers de code à inclure dans les fichiers d'orchestre et de partition (avec *#include*) sont recherchés d'abord dans le répertoire courant, ensuite dans le même répertoire que le fichier d'orchestre ou de partition (respectivement), enfin dans INCDIR.

## Nomenclature

Tout au long de ce document, les opcodes sont indiqués en **caractères gras** et les mnémoniques de leurs arguments et de leur résultat, lorsqu'ils sont mentionnés dans le texte, sont écrits en *italique*. Les noms d'arguments sont généralement des mnémoniques (*amp*, *phs*), et le résultat est souvent dénoté par la lettre *r*. Tous commencent par une qualification de type *i*, *k*, *a*, ou *x* (par exemple *kamp*, *iphs*, *ar*). Le préfixe *i* dénote des valeurs scalaires au temps de l'initialisation de note ; les préfixes *k* ou *a* dénotent des valeurs de contrôle (scalaires) et audio (vectorielles), modifiées et référencées en continu tout au long de l'exécution (c'est-à-dire à chaque période de contrôle tant que l'instrument est actif). Les arguments sont utilisés aux temps indiqués par leur préfixe ; les résultats sont créés aux temps de leur préfixe, et restent disponibles ensuite pour être utilisés comme entrées ailleurs. A part quelques exceptions, les taux des arguments ne peuvent pas dépasser le taux du résultat. La validité des entrées est définie comme suit :

- arguments avec préfixe *i* doivent être valides à l'initialisation ;
- arguments avec préfixe *k* peuvent être des valeurs de contrôle ou d'initialisation (qui restent valides) ;
- arguments avec préfixe *a* doivent être des entrées vectorielles ;
- arguments avec préfixe *x* peuvent être soit des vecteurs soit des scalaires (le compilateur distinguera).

Tous les arguments, sauf précision contraire, peuvent être des expressions dont les résultats sont conformes à la liste ci-dessus. La plupart des opcodes (tels que **linen** et **oscil**) peuvent être utilisés dans plusieurs modes, le choix étant déterminé par le préfixe ou le symbole du résultat.

Tout au long de ce manuel, le terme "opcode" est utilisé pour indiquer une commande qui produit habituellement une sortie au taux-*a*, -*k* ou -*i*, et qui forme toujours la base d'une instruction complète d'un orchestre Csound. Des éléments comme "+" ou "*sin(x)*" ou, "( *a* >= *b* ? *c* : *d* )" sont appelés "opérateurs."

## Macros

Les macros de l'orchestre fonctionnent comme les macros du préprocesseur C, et remplacent le contenu de la macro dans l'orchestre avant sa compilation. Les opcodes qui servent à créer, appeler, ou annuler les macros de l'orchestre sont :

- *#define*
- *\$NAME*
- *#ifdef*
- *#ifndef*
- *#end*
- *#else*
- *#include*
- *#includestr*

- *#undef*

On peut aussi définir des macros de l'orchestre au moyen de l'option de la ligne de commande *--omacro:*.

On peut trouver plus d'information et des exemples sur l'utilisation des macros de l'orchestre à *#define*.

Ces opcodes font référence aux macros de l'orchestre ; pour les macros de la partition, voir *macros de partition*.

## Instruments nommés

La syntaxe de l'orchestre a été modifiée récemment pour permettre de définir des instruments avec des noms en chaîne de caractères. On peut appeler les instruments ainsi nommés depuis la partition et ils sont supportés par un certain nombre d'opcodes.

## Syntaxe

Un instrument nommé est déclaré comme suit :

```
instr Nom[ , Nom2[ , Nom3[ , ... ] ]
[ ... ]
endin
```

Un instrument seul peut avoir autant de noms que l'on veut, et chacun de ces noms peut être utilisé pour appeler l'instrument. De plus, il est possible d'utiliser des nombres comme des noms, dénotant un instrument numéroté de façon standard, ce qui fait que la déclaration suivante est également valide :

```
instr 100, Nom1, 99, Nom2, 1, 2, 3
```

Un nom d'instrument est constitué de lettres, de chiffres, et du caractère de soulignement (\_), sans limite de taille, cependant, le premier caractère ne doit pas être un chiffre. Optionnellement, le nom de l'instrument peut-être préfixé par un caractère '+' (voir ci-dessous), par exemple :

```
instr +Reverb
```

Pour tous les noms d'instrument, un numéro est affecté automatiquement (note : si le niveau des messages (-m) n'est pas nul, ces numéros sont imprimés sur la console pendant la compilation de l'orchestre), en suivant ces règles :

- le nombre est choisi parmi les numéros d'instrument non affectés en ordre ascendant, en commençant par 1
- les numéros sont affectés dans l'ordre de définition des noms d'instrument, si bien que les derniers instruments nommés auront toujours un numéro plus élevé (sauf si le modificateur '+' est utilisé)
- si le nom de l'instrument est préfixé par un '+', le numéro affecté sera plus grand que tous ceux des autres instruments sans le '+' (numérotés et nommés). S'il y a plusieurs instruments '+', la numérotation de ceux-ci suivra l'ordre de leur définition, selon la règle ci-dessus.

L'utilisation de '+' est surtout utile pour la sortie globale ou les instruments d'effets, qui doivent être exécutés après les autres instruments.

Exemple de numérotation d'instruments :

```
instr 1, 2
endin

instr Instr1
endin

instr +Effet1, Instr2
endin

instr 100, Instr3, +Effet2, Instr4, 5
endin
```

Dans cet exemple, les numéros d'instrument sont affectés comme suit :

```
Instr1: 3
Effet1: 101
Instr2: 4
Instr3: 6
Effet2: 102
Instr4: 7
```

## Utilisation des instruments nommés

On peut appeler les instruments nommés en utilisant le nom entre guillemets à la place du numéro d'instrument (note : le caractère '+' doit être omis). Actuellement (depuis Csound 4.22.4), les instruments nommés sont supportés par :

- les événements de partition 'i' et 'q'



### Notes

1. dans les fichiers de partition, il faut éviter les guillemets non appariés, et les espaces et autres caractères illégaux dans les chaînes, sinon (au moins dans la version actuelle) un comportement imprévisible peut apparaître (ce problème n'existe pas pour les événements en ligne -L). Cependant, il y a un test pour détecter les instruments non définis, et dans ce cas, l'évènement est simplement ignoré avec un avertissement.
2. Les utilitaires autonomes (scsort et extract) ne supportent pas les instruments nommés. Il est toujours possible de trier de telles partitions en utilisant l'option -t0 de l'exécutable Csound.

- les événements en ligne (-L)
- les opcodes event, schedkwhen, subinstr, et subinstrinit
- les opcodes massign, pgmassign, prealloc, et mute

De plus, il y a un nouvel opcode (nstrnum) qui retourne le numéro d'un instrument nommé :

```
insno nstrnum "nom"
```

Dans l'exemple ci-dessus, nstrnum "Effet1" retournerait 101. S'il n'existe aucun instrument avec le nom spécifié, une erreur d'initialisation est levée et -1 est retourné.

## Exemple

```

; ---- orchestre ----

sr      = 44100
ksmps   = 10
nchnls  = 1

    prealloc "SineWave", 20
    prealloc "MIDISineWave", 20

    massign 1, "MIDISineWave"

gaOutSend      init 0

    instr +OutputInstr

    out gaOutSend
    clear gaOutSend

    endin

    instr SineWave

a1      oscils p4, p5, 0
    vincr gaOutSend, a1

    endin

    instr MIDISineWave

iamp     veloc
inote     notnum
icps      = cpsoct(inote / 12 + 3)
a1        oscils iamp * 100, icps, 0
    vincr gaOutSend, a1

    endin

; ---- partition ----

i "SineWave" 0 2 12000 440
i "OutputInstr" 0 3
e

```

## Auteur

Istvan Varga

2002

## Opcodes définis par l'utilisateur (UDO)

Csound permet la définition d'opcodes dans l'en-tête de l'orchestre au moyen des opcodes *opcode* et *endop*. L'opcode défini peut fonctionner avec un nombre d'échantillons par période de contrôle (*ksmps*) différent en utilisant *setksmps*.

Pour connecter les entrées et les sorties de l'UDO, on utilise *xin* et *xout*.

Un UDO ressemble à ceci :

```

opcode Lowpass, a, akk

setksmps 1 ; nécessite sr=kr

```

```

ain, kal, ka2  xin          ; lire les paramètres d'entrée
aout  init 0          ; initialiser la sortie
aout  = ain*kal + aout*ka2 ; filtre simple comme tone
xout aout          ; écrire la sortie

endop

```

Cet UDO appelé *Lowpass* reçoit trois entrées (la première au taux-a, et les deux autres au taux-k), et délivre une sortie au taux-a. Noter l'utilisation de *xin* pour recevoir les entrées et de *xout* pour délivrer les sorties. Noter aussi l'utilisation de *setksmps*, qui est nécessaire pour que le filtre fonctionne correctement.

Pour utiliser cet UDO depuis un instrument, on écrirait quelque chose comme :

```
afiltre Lowpass asource, kvaieur1, kvaieur2
```

voir l'entrée *opcode* pour des informations détaillées sur la définition d'UDO.

Vous pouvez trouver plusieurs UDO déjà rédigés (ou apporter votre propre contribution) à *User Defined Opcode Database* [<http://www.csounds.com/udo/>] sur *Csounds.com* [<http://www.csounds.com/>].

## Vecteurs et tableaux

On peut utiliser des vecteurs et des tableaux valeurs de taux-i, de taux-k ou de taux-a dans de nombreux cas. Il sont créés par l'opcode *init* et par d'autres, et l'on peut les utiliser dans la plupart des opérations arithmétiques ou comme opcodes de tableaux spécifiques.

Pendant une courte période Csound avait un schéma plus limité de vecteurs unidimensionnels au taux-k, mais ces derniers ont été intégrés dans les tableaux et les variables-t sont maintenant obsolètes.

## Syntaxe fonctionnelle dans Csound6

Csound6 a introduit une nouvelle syntaxe alternative dans le code de l'orchestre. C'est à l'origine une caractéristique expérimentale qui a certaines limitations comme expliqué ci-dessous. Elle permettra aussi l'introduction de certaines caractéristiques du langage sans compatibilité ascendante.

### Vue d'ensemble

Le principal aspect de la nouvelle syntaxe est que certains opcodes peuvent être appelés en tant que fonctions en ligne dans le code de l'orchestre. La forme générale en est l'expression:

```
var* = op(exprlist*)
```

où \* indique une option, *var* est une variable unique d'un des types de Csound6 et *exprlist* est une liste d'expressions séparées par des virgules (ou une expression unique ou une variable). On peut placer ces expressions n'importe où dans un instrument ou dans un bloc d'opcodes. Les opérations d'initialisation peuvent également être placées hors des blocs d'instrument. La syntaxe fonctionnelle peut être mélangée avec du code de Csound standard.

Voici un exemple de ces expressions :

```
a1 = oscil(p4,p5)
```



```
out(vco2(p4*linen(1, 0.1, p3, 0.1), p5)  
outs(oscili(in(), p5), in())
```

## Restrictions

La principale restriction est que seuls les opcodes à sortie unique (ou sans sortie) sont autorisés. De plus, l'analyse des opcodes à plusieurs sorties facultatives échouera sous cette forme. On peut contourner cela en les incluant dans des opcodes définis par l'utilisateur ou juste en mélangeant la syntaxe de Csound standard avec ce nouveau style.

Pour résoudre les ambiguïtés d'opcode, nous avons introduit les annotations de type sous la forme `op:type(exprlist)`. Par exemple le code :

```
a1 = oscili(oscili:k(p4,p5), 440)
```

choisit un opcode au taux de contrôle pour moduler l'amplitude de la porteuse audio, au lieu d'un opcode au taux audio. Dans certains cas, l'annotation de type est nécessaire si les arguments d'entrée ne permettent pas de déterminer le type correct d'opcode à appliquer.

## Serveur UDP

Csound 6 comprend un serveur UDP pleinement fonctionnel, qui peut accepter un ensemble de commandes et/ou d'orchestres.

## Vue d'ensemble

On peut démarrer le serveur UDP avec l'option suivante :

```
--port=N
```

où N indique un numéro de port pour écouter les messages UDP. Il n'est pas nécessaire de donner à Csound un CSD ou un orchestre, mais on peut aussi le faire.

## Commandes

Les commandes sont formées d'un opcode suivi d'un ou de plusieurs arguments. Le serveur accepte les commandes suivantes :

```
&[line event]
```

Envoie une ligne de texte représentant un événement de partition [live event]. On peut envoyer plusieurs événements sur plusieurs lignes. Utiliser cette commande pour des événements simples ou multiples qui ne nécessitent pas de prétraitement.

`$(score)`

Envoie une partition [score], pour laquelle la majeure partie du prétraitement (à l'exception du tempo) peut être appliquée. Utiliser cette option pour des blocs d'évènements de partition plus importants.

`@(channel_name) [value]`

Donne la valeur [value] au canal de contrôle [channel\_name]. Exemple : si le csd contient la ligne 'chn\_k "freq", 440' alors '@freq 330' fixe "freq" à 330.

`%(channel_name) [string]`

Donne la chaîne de caractères [string] au canal de contrôle [channel\_name].

`:@(channel_name) [address] [port]`

Demande que la valeur du canal de contrôle [channel\_name] soit envoyée sous forme de chaîne de caractères via UDP à l'adresse [address] sur le port [port]. La chaîne de caractères contiendra le nom du canal suivi de deux double-points (::) et de sa valeur courante.

`:%(channel_name) [address] [port]`

Demande que le contenu du canal de chaîne de caractères [channel\_name] soit envoyé sous forme de chaîne de caractères via UDP à l'adresse [address] sur le port [port]. La chaîne de caractères contiendra le nom du canal suivi de deux double-points (::) et de sa valeur courante.

## Code d'orchestre

En plus des commandes ci-dessus, le serveur UDP accepte aussi une chaîne de caractères contenant le code d'un orchestre, qui est compilé immédiatement. Cette chaîne de caractères n'est préfixée par aucun opcode de commande spécial. Cette chaîne de caractères doit être envoyée en un seul message UDP. Par exemple : `scoreline_i "i 2 0 1 550 60"`

Si le code de l'orchestre dépasse le nombre de caractères que peut contenir un message unique, on peut le scinder en messages séparés. Pour que ça fonctionne, le code complet de l'orchestre doit être encadré par des accolades ({ }). L'accolade ouvrante ({) commence l'enregistrement du code par le serveur et l'accolade fermante (}) envoie le code pour compilation. De cette manière, le code peut être envoyé en plusieurs messages.

## Fermeture du serveur

On peut fermer le serveur (et Csound) avec une des commandes suivantes :

`##close##`

ou

!!close!!

---

# La partition numérique standard

La section de la partition contient des événements qui démarrent des instances d'instruments de l'orchestre. La partition propose diverses instructions qui permettent l'élaboration de partitions complexes avec le langage de csound.

Actuellement, la longueur maximale de la partition dépend de l'architecture de la plate-forme ; sur un système 32 bit elle est de  $2^{31}-1$  périodes de contrôle. Par exemple, avec  $kr=1500$ , on peut exécuter une partition pendant une durée maximale de 16,5 jours avant l'apparition de problèmes provoqués par un dépassement des variables entières signées sur 32 bit. Sur une machine 64 bit avec les mêmes conditions la durée serait d'environ neuf milliard d'années. L'entité 'z' est lue comme une durée d'environ 25367 années.

Il faut noter également que lorsque l'on utilise des nombres flottants en simple précision (c-à-d les installeurs 'f' plutôt que les 'd'), la précision temporelle se détériore après une longue durée d'exécution.

## Prétraitement des partitions standard

Une *Partition* (un ensemble d'instructions de partition) se divise en sections ordonnées dans le temps par l'*instruction s*. Avant sa lecture par l'orchestre, une partition est prétraitée section par section. Chaque section est normalement traitée par trois routines : *Carry* (report de valeur), *Tempo*, et *Sort* (tri).

### Carry

Dans un groupe d'*instructions i* consécutives dont les nombres entiers p1 sont indentiques, tout p-champ non rempli prendra la même valeur que celle du p-champ correspondant dans l'instruction précédente. Un p-champ vide peut-être marqué par un point (.) entouré d'espaces. Il n'y a pas besoin de point après le dernier p-champ non vide. La sortie du prétraitement Carry montre explicitement les valeurs reportées. La Fonction Carry n'est pas affectée par les commentaires rencontrés ou les lignes blanches ; elle s'arrête seulement lorsqu'elle rencontre une instruction autre que l'*instruction i* ou une *instruction i* avec un nombre entier p1 différent.

Il y a trois fonctions supplémentaires, pour p2 seulement : +, ^+x, et ^-x. Le symbole + en p2 recevra la valeur de p2 + p3 de l'instruction i précédente. Cela permet de déterminer automatiquement l'instant du début d'une note à partir de la somme des durées précédentes. Le symbole + peut lui-même être reporté. Il n'est autorisé que dans p2. Par exemple : les instructions

```
i1 0 .5 100
i . +
i
```

se transformeront en

```
i1 0 .5 100
i1 .5 .5 100
i1 1 .5 100
```

Les symboles ^+x et ^-x déterminent la valeur de p2 en additionnant ou en soustrayant respectivement la valeur x du p2 précédent. Ils ne peuvent être utilisés qu'en p2 et ne sont *pas* reportés comme le symbole +. Noter aussi qu'il ne doit pas y avoir d'espaces après la partie ^, + ou - de ces symboles -- le nombre doit suivre directement comme dans ^+2.3. Si l'exemple ci-dessus avait été

```
i1  0    .5      100
i .  ^+1
i .  ^+1
```

le résultat aurait été

```
i1  0    .5      100
i1  1    .5      100
i1  2    .5      100
```

On peut se servir largement de la fonction Carry. Son utilisation, spécialement dans les grandes partitions, peut réduire grandement la frappe au clavier et elle simplifiera les modifications ultérieures.

Il y a des circonstances où l'on ne veut pas que les p-champs "manquants" après le dernier qui a été entré soient implicitement reportés. Par exemple dans un instrument prévu pour prendre un nombre variable de p-champs. A partir de Csound 5.08, on peut empêcher le report implicite des p-champs à la fin d'une instruction *i* en utilisant le symbole ! (appelé le "symbol de non-report"). Le ! doit apparaître à la fin d'une instruction *i* et il ne peut pas être utilisé en p1, p2 ou p3, car ces p-champs sont obligatoires. Voici un exemple :

```
i1  0    .5      100
i .  +
i .  .      .      !
i
```

Cette partition sera interprétée comme ceci

```
i1  0    .5      100
i1  .5    .5      100
i1  1    .5      ; no p4
i1  1.5  .5      ; only p1 to p3 are carried here
```

Alternativement à l'utilisation de !, on peut désactiver le report automatique en dehors de p1, p2 et p3. Ceci est réalisé par l'instruction de partition "C 0", et peut être réactivé avec "C 1".

## Tempo

Cette opération modifie l'information temporelle d'une section de partition selon les directives de l'*instruction t*. L'opération tempo convertit p2 (et pour les *instructions i*, p3) de la valeur originale en pulsations vers des secondes réelles, celles-ci étant les unités temporelles requises par l'orchestre. Après la modification temporelle, les fichiers partitions apparaîtront dans un format lisible par l'orchestre comme ceci :

*i* p1 p2pulsations p2secondes p3pulsations p3secondes p4 p5 ...

## Sort

Cette routine trie toutes les instructions d'action temporelle chronologiquement selon la valeur de p2. Elle place aussi les événements simultanés par ordre de priorité. Chaque fois qu'une *instruction f* et une *instruction i* ont la même valeur en p2, l'*instruction f* sera placée en premier. Chaque fois que plusieurs *instructions i* ont la même valeur en p2, elles seront triées par ordre croissant de leur valeur en p1. Si elles ont aussi la même valeur en p1, elles seront triées par ordre croissant de leur valeur en p3. Le tri de la partition est effectué par section (voir l'*instruction s*). Ce tri automatique permet d'écrire les instructions de partition dans n'importe quel ordre à l'intérieur d'une section.



## Note

Les opérations Carry, Tempo et Sort sont combinées dans une seule passe en trois phases sur le fichier de partition, pour produire un nouveau fichier dans un format lisible par l'orchestre (voir l'exemple de Tempo). Ce traitement peut être invoqué explicitement par la commande *Scsort*, ou implicitement par Csound qui traite la partition avant d'appeler l'orchestre. Les fichiers en format source et en format lisible par l'orchestre sont encodés en caractères ASCII, et peuvent être consultés ou modifiés dans un éditeur de texte standard. L'utilisateur peut écrire ses propres routines pour modifier les fichiers de partition avant ou après le processus décrit ci-dessus, pourvu que le format final lisible par l'orchestre soit respecté. Les sections de formats différents peuvent être traitées séquentiellement par lots ; et les sections de même format peuvent être réunies pour le tri automatique.

# Instructions de partition

Les instructions utilisées dans les partitions sont :

- *a* - Avance le temps de la partition d'une quantité spécifiée
- *b* - Réinitialise l'horloge
- *C* - Bascule la fonction de report automatique
- *d* - Efface un instrument infini
- *e* - Marque la fin de la dernière section de la partition
- *f* - Appelle une *routine GEN* pour placer des valeurs dans une table de fonction stockée
- *i* - Active un instrument à une date spécifique et pour une certaine durée
- *m* - Positionne une marque nommée dans la partition
- *n* - Répète une section
- *q* - Rend un instrument silencieux
- *r* - Commence une section répétée
- *s* - Marque la fin d'une section
- *t* - Fixe le tempo
- *v* - Permet une modification temporelle variable localement des événements de la partition
- *x* - Ignore le reste de la section courante
- *y* - Fixe la "graine" pour les nombres aléatoires, soit la valeur de p1, soit la valeur de l'horloge système si p1 est omis.
- *{* - Commence une boucle imbriquable ne délimitant pas de section
- *}* - Termine une boucle imbriquable ne délimitant pas de section

Les commentaires sont dénotés par un point-virgule (;), un double slash (//) ou le caractère c et durent jusqu'à la fin de la ligne. Les commentaires dans le style du langage C /\* ... \*/ sont également permis.

# Symboles next-P et previous-P

A la fin de chacune des opération *Carry*, *Tempo*, et *Sort*, trois fonctions de partition supplémentaires sont interprétées durant l'écriture du fichier : *next-p*, *previous-p*, et *ramping*.

Les p-champs d'une *instruction i* contenant les symboles *npx* ou *ppx* (où *x* est un entier) seront remplacés par la valeur du p-champ approprié de l'instruction *i* suivante (ou de l'instruction *i* précédente) ayant le même p1. Par exemple, le symbole *np7* sera remplacé par la valeur du p7 de la note suivante devant être jouée par le même instrument. Les symboles *np* et *pp* sont récursifs et peuvent référencer d'autres symboles *np* et *pp* qui peuvent en référencer d'autres, etc. Les références doivent se terminer par un nombre réel ou un *symbole ramp*. Il faut éviter les références en boucle fermée. Les symboles *np* et *pp* sont interdits en p1, p2 et p3 (bien qu'ils puissent référencer ces derniers). Les symboles *np* et *pp* peuvent être reportés (*Carry*). Les référence de *np* et de *pp* ne peuvent traverser une limite de Section. Toute référence avant ou arrière à une instruction de note inexistante recevra la valeur zéro.

Par exemple : les instructions

```
i1  0  1  10  np4  pp5
i1  1  1  20
i1  1  1  30
```

se transformeront en

```
i1  0  1  10  20  0
i1  1  1  20  30  20
i1  2  1  30  0  30
```

Les symboles *np* et *pp* peuvent apporter à un instrument une connaissance contextuelle de la partition, ce qui permettra de réaliser un glissando ou un crescendo, par exemple, vers la hauteur ou l'intensité d'un évènement futur (qui peut être immédiatement adjacent ou non). A noter que bien que la fonction *Carry* propage *np* et *pp* vers des instructions non triées, l'opération d'interprétation de ces symboles se fait sur une version de la partition complètement triée. L'opération de tempo est appliquée après le traitement de *np* et/ou de *pp*.

## Ramping

Les p-champs d'une *instruction i* contenant le symbole *<* seront remplacés par des valeurs issues de l'interpolation linéaire d'une pente temporelle. Les pentes sont attachées à chaque extrémité au premier nombre réel trouvé dans le même p-champ de notes précédentes et suivantes jouées par le même instrument. Par exemple : les instructions

```
i1  0  1  100
i1  1  1  <
i1  2  1  <
i1  3  1  400
i1  4  1  <
i1  5  1  0
```

se transformeront en

```
i1  0  1  100
i1  1  1  200
i1  2  1  300
```

```
i1 3 1 400
i1 4 1 200
i1 5 1 0
```

Les pentes ne peuvent pas traverser une limite de Section. Les pentes ne peuvent pas être attachées à un symbole *np* ou *pp* (mais elles peuvent être référencées par ceux-ci). Les symboles de pente sont interdits en p1, p2 et p3. Les symboles de pente peuvent être reportés. A noter cependant que, bien que la fonction *Carry* propage les symboles de pente vers des instructions non triées, l'opération d'interprétation de ces symboles se fait sur une version de la partition résolue temporellement et complètement triée. En fait, l'interpolation linéaire temporelle est basé sur le temps de partition résolu, de façon à ce qu'une pente couvrant un groupe de notes *accelerando* reste linéaire par rapport au temps strictement chronologique.

A partir de la version 3.52 de Csound, l'utilisation des symboles ( ou ) donne une pente d'interpolation exponentielle, comme *expon*. L'utilisation du symbole ~ donnera une distribution aléatoire uniforme entre la première et la dernière valeur de la pente. L'utilisation de ces fonctions suit les mêmes règles que la fonction de pente linéaire.

## Macros de partition

### Description

Les macros sont des substitutions de texte qui sont réalisées dans la partition lors de sa présentation au système. Le système de macro de Csound est très simple, et il utilise les caractères # et \$ pour définir et appeler des macros. C'est un moyen de simplifier l'écriture d'une partition, et une alternative élémentaire aux systèmes de génération de partition complète. Le système de macros de partition est similaire, mais de façon indépendante, au système de macros du langage de l'orchestre.

*#define* NOM -- définit une macro simple. Le nom de la macro doit commencer par une lettre et peut être une combinaison de lettres et de nombres. La casse est significative. Cette forme est restrictive dans le sens que les noms de variable sont fixes. On peut obtenir plus de souplesse au moyen d'une macro avec arguments, décrite ci-dessous.

*#define* NOM(a' b' c') -- définit une macro avec arguments. On peut l'utiliser dans des situations plus complexes. Le nom de la macro doit commencer par une lettre et peut être suivi par une combinaison de lettres et de chiffres. Dans le texte de substitution, les arguments sont remplacés par la forme : \$A. En fait, les arguments sont implémentés comme des macros simples. Il peut y avoir jusqu'à 5 arguments, et leur nom peut être n'importe quel choix de lettres. Rappelez-vous que la casse est significative dans les noms de macro.

*\$NOM.* -- appelle une macro définie. Pour appeler une macro, on utilise son nom précédé d'un caractère \$. Le nom se termine par le premier caractère qui n'est ni une lettre ni un chiffre. S'il est nécessaire que le nom soit suivi d'une lettre ou d'un chiffre, un point, qui sera ignoré, peut être utilisé pour terminer le nom. La chaîne, *\$NOM.*, est remplacée par le texte de substitution de la définition. Le texte de substitution peut aussi contenir des appels de macro.

*#undef* NOM -- rend un nom de macro indéfini. Si l'on a plus besoin d'une macro, on peut la rendre indéfinie avec *#undef* NOM.

### Syntaxe

```
#define NOM # texte de substitution #

#define NOM(a' b' c') # texte de substitution #

$NOM.
```



```
#undef NOM
```

## Initialisation

*# texte de substitution #* -- Le texte de substitution est une chaîne de caractères (ne contenant pas de #) et peut s'étendre sur plusieurs lignes. Le texte de substitution est délimité par des caractères #, ce qui permet d'éviter l'insertion de caractères supplémentaires par inadvertance.

## Exécution

Il faut prendre quelques précautions avec les macros de substitution de texte, car elle peuvent parfois produire d'étranges résultats. Elles ne tiennent compte d'aucune valeur sémantique, et ainsi les espaces sont significatifs. C'est pourquoi, au contraire du langage C, la définition délimite le texte de substitution par des caractères #. Utilisé avec discernement, ce système de macro est un concept puissant, mais il peut aussi être mal employé.

**Une autre utilisation des macros.** Lorsque l'on écrit une partition complexe, on oublie parfois trop facilement à quoi les différents numéros d'instruments font référence. On peut utiliser des macros pour nommer ces nombres. Par exemple

```
#define Flute #i1#
#define Whoop #i2#

$Flute. 0 10 4000 440
$Whoop. 5 1
```

## Exemples

### Exemple 1. Macro simple

Une note a un ensemble de p-champs qui sont répétés :

```
#define ARGS # 1.01 2.33 138#
i1 0 1 8.00 1000 $ARGS
i1 0 1 8.01 1500 $ARGS
i1 0 1 8.02 1200 $ARGS
i1 0 1 8.03 1000 $ARGS
```

Ce sera développé avant le tri en :

```
i1 0 1 8.00 1000 1.01 2.33 138
i1 0 1 8.01 1500 1.01 2.33 138
i1 0 1 8.02 1200 1.01 2.33 138
i1 0 1 8.03 1000 1.01 2.33 138
```

On économise ainsi de la frappe au clavier, et les révisions sont plus faciles. Avec deux ensembles de p-champs on pourrait avoir une seconde macro (il n'y pas de réelle limite au nombre de macros que l'on peut définir).

```
#define ARGS1 # 1.01 2.33 138#
#define ARGS2 # 1.41 10.33 1.00#
i1 0 1 8.00 1000 $ARGS1
i1 0 1 8.01 1500 $ARGS2
i1 0 1 8.02 1200 $ARGS1
i1 0 1 8.03 1000 $ARGS2
```

## Exemple 2. Macros avec arguments

```
#define ARG(A) # 2.345 1.03 $A 234.9#  
i1 0 1 8.00 1000 $ARG(2.0)  
i1 + 1 8.01 1200 $ARG(3.0)
```

qui se développe en

```
i1 0 1 8.00 1000 2.345 1.03 2.0 234.9  
i1 + 1 8.01 1200 2.345 1.03 3.0 234.9
```

## Crédits

Auteur : John ffitich

University of Bath/Codemist Ltd.

Bath, UK

Avril 1998 (Nouveau dans la version 3.48 de Csound)

## Partition dans plusieurs fichiers

### Description

Disposer la partition dans plusieurs fichiers.

### Syntaxe

```
#include "nomfichier"  
#includestr "filename"
```

### Exécution

Il est parfois commode de disposer la partition dans plusieurs fichiers. On peut le faire en utilisant *#include* qui fait partie du système de macro. Par une ligne contenant le texte

```
#include "nomfichier"
```

où le caractère " peut être remplacé par n'importe quel caractère adéquat. Pour la plupart des usages, le symbole des guillemets sera probablement le plus adapté. Le nom de fichier peut comprendre un nom de chemin complet.

On prend en entrée le contenu du fichier nommé, puis on revient à l'entrée précédente. La profondeur des fichiers inclus et des macros est actuellement limitée à 20.

On peut utiliser *#include* pour définir un ensemble de macros qui font partie du style du compositeur. On peut aussi l'utiliser pour répéter des sections.

```
s  
#include :section1:
```

```
;; Répéter ceci  
s  
#include :section1:
```

Pour d'autres méthodes de répétition, utiliser l'instruction *r*, l'instruction *m*, et l'instruction *n*.

La forme *#includestr* est semblable à *#include* sauf que le délimiteur doit être une double apostrophe et que le nom de fichier peut contenir des appels de macro.

## Crédits

Auteur : John fitch

University of Bath/Codemist Ltd.

Bath, UK

Avril 1998 (Nouveau dans la version 3.48 de Csound)

Merci à Luis Jure d'avoir relevé la syntaxe incorrecte dans l'instruction d'inclusion de fichiers multiples.

## Evaluation des expressions

Dans les anciennes versions de Csound les nombres présents dans une partition étaient utilisés tels quels. Dans certains cas, une évaluation simple serait plus facile. Ce besoin est accru s'il y a des macros. Pour y arriver, on a introduit la syntaxe des expressions arithmétiques entre crochets [ ]. On peut utiliser des expressions avec les opérations +, -, \*, /, % ("modulo"), et ^ ("élévation à une puissance"), les groupements se faisant par parenthèses (). Les signes unaires plus et moins sont aussi supportés. Les expressions peuvent inclure des nombres et, naturellement, des macros dont la valeur est une chaîne numérique ou arithmétique. Tous les calculs sont faits en nombres en virgule flottante. Les règles de précedence usuelles sont suivies lors de l'évaluation : les expressions entre parenthèse () sont évaluées en premier et ^ est évalué avant \*, /, et % qui sont évalués avant + et -.

En plus des opérations arithmétiques, les opérateurs logiques bit à bit suivants sont aussi disponibles : & (ET), | (OU), et # (OU exclusif). Ces opérateurs arrondissent leurs opérands à l'entier (long) le plus proche avant l'évaluation. Les opérateurs logiques ont la même précedence que les opérateurs arithmétiques \*, /, et %.

Finalement, on peut utiliser le symbole tilde ~ dans une expression chaque fois qu'un nombre est permis. Chaque ~ sera remplacé par un nombre aléatoire compris entre zéro (0) et un (1).

## Exemple

```
r3 CNT  
  
i1 0 [0.3*$CNT.]  
i1 + [( $CNT./3)+0.2]  
  
e
```

Comme les trois copies de la section comprennent la macro \$CNT. avec les valeurs successives 1, 2 et 3, le développement est

```
s  
i1 0 0.3
```

```
i1 0.3 0.533333
s
i1 0 0.6
i1 0.6 0.866667
s
i1 0 0.9
i1 0.9 1.2
e
```

C'est une forme extrême, mais on peut aussi utiliser le système d'évaluation pour répéter des sections avec des différences subtiles.

Voici quelques exemples simples de chaque opérateur :

```
i1 0 1 [ 110 + 220 ] ; evaluates to 330
i1 + . [ 330 - 55 ] ; 275
i1 + . [ 44 * 10 ] ; 440
i1 + . [ 1100 / 2 ] ; 550
i1 + . [ 5 ^ 4 ] ; 625
i1 + . [ 5660 % 1000 ] ; 660
i1 + . [ 110 & 220 ] ; 76
i1 + . [ 110 | 220 ] ; 254
i1 + . [ 110 # 220 ] ; 178
i1 + . [ ~ ] ; random between 0-1
i1 + . [ ~ * 4 + 1 ] ; random between 1-5
i1 + . [ ~ * 95 + 5 ] ; random between 5-100

i1 + . [ 8 / 2 * 3 ] ; 12
i1 + . [ 4 + 3 - 2 + 1 ] ; 6
i1 + . [ 4 + 3 * 2 + 1 ] ; 11
i1 + . [ (4 + 3)*(2 + 1) ] ; 21

i1 + . [ 2 * 2 & 3 ] ; 4
i1 + . [ 3 & 2 * 2 ] ; 0
i1 + . [ 4 | 3 * 3 ] ; 13
```

## L'opérateur @

On a ajouté dans la version 3.56 de Csound @ $x$  (la première puissance de deux supérieure ou égale à  $x$ ) et @@ $x$  (la première puissance de deux plus un supérieure ou égale à  $x$ ).

```
[ @ 11 ] will evaluate to 16
[ @@ 11 ] to 17
```

## Crédits

Auteur : John ffitich

University of Bath/Codemist Ltd.

Bath, UK

Avril 1998 (Nouveau dans la version 3.48 de Csound)

## Chaînes de caractères dans les p-champs

On peut passer une chaîne de caractères dans un p-champ au lieu d'un nombre, comme ceci :

i 1 0 10 "A4"

Cette chaîne de caractères peut être reçue par l'instrument et traitée par les *opcodes de chaîne de caractères*.



### Note

Actuellement un seul p-champ peut contenir une chaîne de caractères (c-à-d qu'on n'autorise pas plus d'une chaîne de caractères par ligne). On peut contourner ceci en utilisant *strset* et *strget*.

---

# Frontaux

Les frontaux sont des programmes fournissant une sorte d'interface utilisateur pour Csound. Avec ces programmes, Csound est utilisé comme générateur de son, et il faut donc avoir une certaine familiarité avec le code de Csound pour les utiliser. Les frontaux apportent habituellement des facilités comme la coloration syntaxique, des widgets graphiques, ou des outils de génération algorithmique de partition, qui ne font pas partie de Csound lui-même. La plupart de ces programmes sont créés par une seule personne, ce qui fait que certains d'entre eux ne sont pas maintenus. Ci-dessous, une liste (sans doute incomplète et pas forcément à jour) des frontaux disponibles pour Csound.

La plupart du temps, on téléchargera et on installera Csound lui-même avant de télécharger et d'installer un frontal. Certains frontaux nécessitent des versions particulières de Csound. Il est donc recommandé, si l'on veut utiliser un frontal, de vérifier sa compatibilité avant d'installer Csound.

## CsoundQt

CsoundQt est un programme GUI (interface utilisateur graphique) multi plates-formes aux fonctions variées qui est fourni avec la distribution standard de Csound. Créé et maintenu par Andrés Cabrera, CsoundQt comprend un éditeur à plusieurs onglets, des widgets graphiques pour le contrôle du son en temps réel, et un système d'aide des opcodes qui pointe dans ce manuel. A l'heure actuelle (2013), CsoundQt est dans une phase active de développement, ce qui fait que la version installée sur votre système avec Csound peut ne pas être la plus récente. On peut trouver la dernière version à <http://qutecsound.sourceforge.net/>.

## Blue

Frontal multi plates-formes orienté composition, écrit en Java par Steven Yi. L'interface utilisateur fournit une structure de ligne temporelle comme sur un multipiste numérique, mais en diffère en ce que les lignes temporelles peuvent être intégrées dans d'autres lignes temporelles (polyObjects). Cela permet une organisation compositionnelle dans le temps que beaucoup trouveront intuitive, compréhensible et flexible. Chaque instrument et chaque section de partition dans un projet blue ont leur propre fenêtre d'édition, ce qui facilite l'organisation des projets de grande taille. On peut télécharger Blue à la Blue Home Page [<http://csounds.com/stevenyi/blue/>].

## Cabbage

Cabbage est un frontal de Csound qui permet de développer des greffons audio et des logiciels autonomes sur les trois principaux systèmes d'exploitation. Bien que Cabbage utilise des technologies sous-jacentes de greffon telles que le SDK VST de Steinberg, ASIO, etc, Csound est utilisé pour traiter les flux audio entrant et sortant. Cabbage fournit aussi une collection croissante de contrôles graphiques allant de la simple réglette aux surfaces XY programmables. Tous les contrôles graphiques dans un greffon de Cabbage peuvent être contrôlés via des automatismes de l'hôte du greffon, fournissant ainsi un moyen rapide et efficace d'automatiser les paramètres d'instrument de Csound dans des stations audio numériques commerciales ou non. On peut télécharger Cabbage sur la page d'accueil de Cabbage [<https://github.com/cabbageaudio/cabbage/releases>].

## WinXound

WinXound est un éditeur graphique frontal libre et open-source avec coloration syntaxique pour Csound6, CSoundAV, CSoundAC, avec le support de Python et de Lua. Il est développé par Stefano Bonetti. Il fonctionne sous Windows de Microsoft, Mac OS X d'Apple et Linux. On peut le télécharger sur la page principale de WinXound [<http://winxound.codeplex.com/>].

## Winsound

Winsound appartenait à la branche principale de Csound. Il n'est plus maintenant disponible que sous forme de code source. Winsound est un portage multi plates-formes en FLTK d'un frontal antérieur pour Windows. Certains utilisateurs mal-voyants ou aveugles ont rapporté que Winsound fonctionne bien avec les logiciels de traduction texte-parole.

## CsoundAC

### Programmation Python

Vous pouvez utiliser CsoundAC comme un module d'extension de Python. Vous pouvez faire cela dans un interpréteur Python standard tel que la ligne de commande Python ou le Idle Python GUI.

Pour utiliser CsoundAC dans un interpréteur Python standard, importez CsoundAC.

```
import CsoundAC
```

Le module CsoundAC crée automatiquement une instance de CppSound nommée `csound`, qui fournit une interface orientée objet à l'API de Csound. Dans un interpréteur Python standard, vous pouvez charger un fichier Csound `.csd` et l'exécuter de cette manière :

```
C:\Documents and Settings\mkg>python
Python 2.3.3 (#51, Dec 18 2003, 20:22:39) [MSC v.1200 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import CsoundAC
>>> csound.load("c:/projects/csound5/examples/trapped.csd")
1
>>> csound.exportForPerformance()
1
>>> csound.perform()
BEGAN CppSound::perform(5, 988ee0)...
BEGAN CppSound::compile(5, 988ee0)...
Using default language
0dBFS level = 32767.0
Csound version 5.00 beta (float samples) Jun  7 2004
libsndfile-1.0.10pre6
orchname: temp.orc
scorename: temp.sco
orch compiler:
398 lines read
instr  1
instr  2
instr  3
instr  4
instr  5
instr  6
instr  7
instr  8
instr  9
instr 10
instr 11
instr 12
instr 13
instr 98
instr 99
sorting score ...
... done
Csound version 5.00 beta (float samples) Jun  6 2004
```

```

displays suppressed
0dBFS level = 32767.0
orch now loaded
audio buffered in 16384 sample-frame blocks
SFDIR undefined. using current directory
writing 131072-byte blks of shorts to test.wav
WAV
SECTION 1:
ENDED CppSound::compile.
ftable 1:
ftable 2:
ftable 3:
ftable 4:
ftable 5:
ftable 6:
ftable 7:
ftable 8:
ftable 9:
ftable 10:
ftable 11:
ftable 12:
ftable 13:
ftable 14:
ftable 15:
ftable 16:
ftable 17:
ftable 18:
ftable 19:
ftable 20:
ftable 21:
ftable 22:
new alloc for instr 1:
B 0.000 .. 1.000 T 1.000 TT 1.000 M: 32.7 0.0
new alloc for instr 1:
B 1.000 .. 3.600 T 3.600 TT 3.600 M: 207.6 0.1

...

B 93.940 .. 94.418 T 98.799 TT281.799 M: 477.6 85.0
B 94.418 ..100.000 T107.172 TT290.172 M: 118.9 11.5
end of section 4 sect peak amps: 25950.8 26877.4
inactive allocs returned to freespace
end of score. overall amps: 32204.8 31469.6
overall samples out of range: 0 0
0 errors in performance
782 131072-byte soundblks of shorts written to test.wav WAV
Elapsed time = 13.469000 seconds.
ENDED CppSound::perform.
1
>>>

```

Le script `koch.py` montre comment utiliser Python pour faire une composition algorithmique pour Csound. Vous pouvez utiliser des chaînes de caractères littérales à triples guillemets pour incorporer vos fichiers Csound directement dans votre script, et les assigner à Csound :

```

csound.setOrchestra(''sr = 44100
kr = 441
ksmps = 100
nchnls = 2
0dbfs = .1
instr 1,2,3,4,5 ; FluidSynth General MID
I; INITIALIZATION
; Channel, bank, and program determine the preset, that is, the actual sound.
ichannel = p1
iprogram = p6
ikey = p4

```



```
ivelocity      =      p5 + 12
ijunk6         =      p6
ijunk7         =      p7
; AUDIO
istatus        =      144;
print          iprogram, istatus, ichannel, ikey, ivelocityaleft, aright
fluid          "c:/projects/csound5/samples/VintageDreamsWaves-v2.sf2", \\
iprogram, istatus, ichannel, ikey, ivelocity, 1
outs           aleft, arightendin'')
csound.setCommand("csound --opcode-lib=c:/projects/csound5/fluid.dll \\
-RWdfo ./koch.wav ./temp.orc ./temp.sco")
csound.exportForPerformance()
csound.perform()
```

---

# Construire Csound

Csound est devenu un projet complexe et peut impliquer plusieurs dépendances. A moins d'être un développeur de Csound ou d'avoir besoin d'écrire des greffons pour Csound, il vaut mieux utiliser une des distributions pré-compilées de <https://csound.com/download>.

On trouvera des informations détaillées et à jour sur la construction de Csound à partir des sources, dans le fichier BUILD.md [<https://github.com/csound/csound/blob/develop/BUILD.md>] des sources de Csound6.

---

# Liens Csound

La "page d'accueil" de Csound se trouve ici : <http://csound.github.io>.

Une autre page Csound, maintenue par Richard Boulanger, se trouve ici : <http://csounds.com>.

Le code source de Csound est maintenu par John ffitch et d'autres à <https://github.com/csound>. Les versions les plus récentes et les paquetages précompilés pour la plupart des plates-formes peuvent être téléchargés ici [<https://csound.com/download>].

Il existe une liste de diffusion pour discuter de Csound. Elle est animée par John ffitch et Victor Lazzarini de l'université de Maynooth en Irlande. Pour vous inscrire sur cette liste de diffusion envoyez un mél à : [listserv@heanet.ie](mailto:listserv@heanet.ie) [<mailto:listserv@listserv.heanet.ie>] avec "subscribe csound" dans le corps du message. Vous pouvez aussi souscrire à la version condensée (1 message par jour) en envoyant un message à : [listserv@listserv.heanet.ie](mailto:listserv@listserv.heanet.ie) [<mailto:listserv@listserv.heanet.ie>] avec "subscribe csound set digest" dans le corps du message. Les messages envoyés à [csound@listserv.heanet.ie](mailto:csound@listserv.heanet.ie) [<mailto:csound@listserv.heanet.ie>] sont distribués à tous les membres de la liste.

De même, la liste de diffusion `csound-devel` existe pour discuter du développement de Csound. Pour plus d'information sur cette liste, aller à <http://listserv.heanet.ie> [<http://listserv.heanet.ie/>] et suivez le lien vers `csound-dev`. Les messages envoyés à [csound-dev@listserv.heanet.ie](mailto:csound-dev@listserv.heanet.ie) [<mailto:csound-dev@listserv.heanet.ie>] vont à tous les membres de la liste.

Les suspicions de bogues dans le code peuvent être soumises par le biais du système de traçage de bogues sur github [<https://github.com/csound/csound/issues>].

---

## **Partie II. Vue d'ensemble des opcodes**

---

---

## Table des matières

Générateurs de signal .....	90
Synthèse/resynthèse additive .....	90
Oscillateurs élémentaires .....	90
Oscillateurs à spectre dynamique .....	90
Synthèse FM .....	91
Synthèse granulaire .....	91
Synthèse Hyper Vectorielle .....	92
Générateurs linéaires et exponentiels .....	92
Générateurs d'enveloppe .....	93
Modèles et émulations .....	93
Phaseurs .....	94
Générateurs de nombres aléatoires (de bruit) .....	95
Reproduction de sons échantillonnés .....	96
Soundfonts .....	97
Synthèse par balayage .....	98
Accès aux tables .....	99
Synthèse par terrain d'ondes .....	100
Modèles physiques par guide d'onde .....	100
Entrée et sortie de signal .....	101
Entrées et sorties fichier .....	101
Entrée de signal .....	101
Sortie de signal .....	101
Bus logiciel .....	102
Impression et affichage .....	102
Requêtes sur les fichiers sons .....	102
Modificateurs de signal .....	104
Modificateurs d'amplitude et traitement des dynamiques .....	104
Convolution et morphing .....	104
Retard .....	104
Panoramique et spatialisation .....	105
Réverbération .....	106
Opérateurs du niveau échantillon .....	107
Limiteurs de signal .....	107
Effets spéciaux .....	107
Filtres standard .....	108
Filtres spécialisés .....	110
Guides d'onde .....	110
Distorsion non-linéaire et distorsion de phase .....	110
Contrôle d'instrument .....	112
Contrôle d'horloge .....	112
Valeurs conditionnelles .....	112
Instructions de contrôle de durée .....	112
Widgets FLTK et contrôleurs GUI .....	112
Conteneurs FLTK .....	115
Valuateurs FLTK .....	115
Autres widgets FLTK .....	116
Modifier l'apparence des widgets FLTK .....	116
Opcodes généraux relatifs aux widgets FLTK .....	117
Appel d'instrument .....	117
Contrôle séquentiel d'un programme .....	118
Contrôle de l'exécution en temps réel .....	119

Initialisation et réinitialisation .....	119
Détection et contrôle .....	120
Piles .....	121
Contrôle de sous-instrument .....	121
Lecture du temps .....	121
Contrôle des tables de fonction .....	123
Requêtes sur une table .....	123
Opérations de lecture/écriture de table .....	123
Lecture de table avec sélection dynamique .....	124
Opérations mathématiques .....	125
Conversion d'amplitude .....	125
Opérations arithmétiques et logiques .....	125
Compareurs et accumulateurs .....	125
Fonctions mathématiques .....	126
Opcodes équivalents à des fonctions .....	126
Fonctions aléatoires .....	126
Fonctions trigonométriques .....	127
Opcodes d'algèbre linéaire .....	128
Opcodes de tableaux .....	136
Conversion des hauteurs .....	144
Fonctions .....	144
Opcodes de hauteurs .....	144
Support MIDI en temps réel .....	145
Clavier virtuel MIDI .....	146
Entrée MIDI .....	149
Sortie de message MIDI .....	150
Entrée et sortie génériques .....	150
Convertisseurs .....	150
Extension d'évènements .....	150
Sortie de note-on/note-off .....	151
Opcodes pour l'interopérabilité MIDI/partition .....	151
Messages système temps réel .....	152
Banques de réglettes .....	152
Traitement spectral .....	154
Resynthèse par transformée de Fourier à court-terme (STFT) .....	154
Resynthèse par codage prédictif linéaire (LPC) .....	154
Traitement spectral non-standard .....	155
Outils pour le traitement spectral en temps réel (opcodes pvs) .....	155
Traitement spectral avec ATS .....	157
Opcodes Loris .....	157
Opcodes spectraux basés sur des tableaux .....	161
Chaînes de caractères .....	162
Opcodes de manipulation de chaîne .....	163
Opcodes de conversion de chaîne .....	164
Opcodes vectoriels .....	165
Opérateurs de tableaux de vecteurs .....	165
Opérations entre un signal vectoriel et un signal scalaire .....	165
Opérations entre deux signaux vectoriels .....	166
Générateurs vectoriels d'enveloppe .....	166
Limitation et enroulement des signaux vectoriels de contrôle .....	167
Chemins de retard vectoriel au taux de contrôle .....	167
Générateurs de signal aléatoire vectoriel .....	167
Système de patch zak .....	168
Accueil de greffon .....	169

DSSI et LADSPA pour Csound .....	169
OSC et réseau .....	170
Opcodes Ableton Link .....	170
OSC .....	171
Réseau .....	171
Opcodes pour le traitement à distance .....	171
Opcodes Mixer .....	172
Opcodes de graphe de fluence .....	173
Opcodes Jacko .....	176
Opcodes Python .....	179
Introduction .....	179
Syntaxe de l'orchestre .....	179
Opcodes pour le traitement d'image .....	181
Opcodes STK .....	182
Opcodes divers .....	184

---

# Générateurs de signal

## Synthèse/resynthèse additive

Les opcodes pour la synthèse et la resynthèse additives sont :

- *adsyn*
- *adsynt*
- *adsynt2*
- *hsboscil*
- *beadsynt*

Voir la section *Traitement Spectral* pour plus d'information et des opcodes de synthèse/resynthèse additive supplémentaires.

## Oscillateurs élémentaires

Les opcodes des oscillateurs élémentaires sont : (noter que les opcodes qui se terminent par un 'i' implémentent l'interpolation linéaire et que ceux qui se terminent par un '3' implémentent l'interpolation cubique)

- Banques d'Oscillateurs : *oscbnk*
- Oscillateurs simples à table : *oscil*, *oscil3* et *oscili*.
- Oscillateur sinusoïdal simple, rapide : *oscils*
- Oscillateurs de précision : *poscil* et *poscil3*.
- Oscillateur à bande améliorée: *beosc*
- Oscillateurs plus flexibles : *oscilikt*, *osciliktp*, *oscilikts* et *osciln* (aussi appelé *oscilx*).

On peut aussi construire des oscillateurs à partir des opcodes de lecture de table. Voir la section *Opérations de Lecture/Ecriture de Table*.

## LFOs

- *lfo*
- *vibr*
- *vibrato*

Voir la section *Accès aux Tables* pour d'autres opcodes de lecture de table que l'on peut utiliser comme oscillateurs. Voir aussi la section *Oscillateurs à Spectre Dynamique*.

## Oscillateurs à spectre dynamique

Les opcodes qui génère des spectres dynamiques sont :



- Spectres harmoniques : *buzz* et *gbuzz*
- Générateur d'impulsions : *mpulse*
- Oscillateurs à bande limitée (d'après des modèles analogiques) : *vco* et *vco2*

On peut utiliser les opcodes suivants pour générer des formes d'onde à bande limitée pour une utilisation avec *vco2* et d'autres oscillateurs :

- *vco2init*
- *vco2ft*
- *vco2ift*

## Synthèse FM

Les opcodes de synthèse FM sont :

- *foscil*
- *foscili*
- *crossfm*, *crossfmi*, *crosspm*, *crosspmi*, *crossfmpm* et *crossfmpmi*.

## Modèles d'instrument FM

- *fmb3*
- *fmbell*
- *fmmetal*
- *fmpercfl*
- *fmrhode*
- *fmvoice*
- *fmwurlie*

## Synthèse granulaire

Les opcodes de synthèse granulaire sont :

- *diskgrain*
- *fof*
- *fof2*
- *fog*
- *grain*
- *grain2*

- *grain3*
- *granule*
- *partikkel*
- *partikkelsync*
- *sndwarp*
- *sndwarpst*
- *syncgrain*
- *syncloop*
- *vosim*

## Synthèse Hyper Vectorielle

- *vphaseseg*
- *hvs1*
- *hvs2*
- *hvs3*

## Générateurs linéaires et exponentiels

Les opcodes qui génèrent des courbes ou des segments linéaires ou exponentiels sont :

- *expon*
- *expcurve*
- *expseg*
- *expsega*
- *expsegr*
- *gainslider*
- *jspline*
- *line*
- *linseg*
- *linsegr*
- *logcurve*
- *loopseg*
- *loopsegp*

- *lpshold*
- *lpsholdp*
- *rspline*
- *scale*
- *transeg*
- *bpf*
- *bpfcos*
- *linlin*
- *lincos*
- *xyscale*

## Générateurs d'enveloppe

Les générateurs d'enveloppe suivants sont disponibles :

- *adsr*
- *madsr*
- *mxadsr*
- *xadsr*
- *linen*
- *linenr*
- *envlpx*
- *envlpxr*
- *lineto*
- *tlineto*
- *linlin*
- *lincos*

Consulter la section des *Générateurs Linéaires et Exponentiels* pour d'autres méthodes de création d'enveloppes.

## Modèles et émulations

Les opcodes suivants réalisent la modélisation ou l'émulation des sons d'autres instruments (certains basés sur la boîte à outils STK par Perry Cook) :

- *bamboo*

- *barmodel*
- *cabasa*
- *crunch*
- *dripwater*
- *gogobel*
- *guiro*
- *mandol*
- *marimba*
- *moog*
- *sandpaper*
- *sekere*
- *shaker*
- *sleighbells*
- *stix*
- *tambourine*
- *vibes*
- *voice*

Voir aussi la section sur les *opcodes STK* pour une information détaillée sur ces derniers.

Autres modèles et émulations

- *lorenz*
- *planet*
- *prepiano*
- Générateur de Nombres Fractals (ensemble de Mandelbrot) : *mandel*
- *chuap*
- *gendy*
- *gendyc*
- *gendyx*

On peut trouver une section sur les modèles physiques par guide d'onde : *ici*.

## Phaseurs

Les opcodes qui génèrent une valeur de phase mobile :

- *ephasor*
- *phasor*
- *phasorbnk*
- *syncphasor*
- *sc\_phasor*

Ces opcodes sont utiles en combinaison avec les opcodes d'*Accès aux Tables*.

## Générateurs de nombres aléatoires (de bruit)

Les opcodes qui génèrent des nombres aléatoires sont :

- *betarnd*
- *bexprnd*
- *cauchy*
- *cuserrnd*
- *duserrnd*
- *dust*
- *dust2*
- *exprand*
- *fractalnoise*
- *gauss*
- *gausstrig*
- *linrand*
- *noise*
- *pcauchy*
- *pinkish*
- *pinker*
- *poisson*
- *rand*
- *randh*
- *randi*
- *rnd31*

- *random*
- *randomh*
- *randomi*
- *trirand*
- *unirand*
- *urd*
- *weibull*
- *jitter*
- *jitter2*
- *trandom*

Voir *seed* qui fixe la valeur de la racine globale pour tous les générateurs de bruit de classe *x*, ainsi que d'autres opcodes qui utilisent un appel de fonction aléatoire comme *grain.rand*, *randh*, *randi*, *rnd(x)* et *birnd(x)* ne sont pas affectés par *seed*.

Voir aussi les fonctions qui génèrent des nombres aléatoires dans la section *Fonctions Aléatoires*.

## Reproduction de sons échantillonnés

Les opcodes qui implémentent la reproduction de sons échantillonnés (samples) et les boucles sont :

- *bbcutm*
- *bbcuts*
- *flooper*
- *flooper2*
- *loscil*
- *loscil3*
- *loscilx*
- *lphasor*
- *lposcil*
- *lposcil3*
- *lposcila*
- *lposcilsa*
- *lposcilsa2*
- *sndloop*

- *waveset*

Voir aussi la section *Entrée de Signal* pour d'autres types d'entrées sonores.

## Soundfonts

### Opcodes Fluid

La famille des opcodes fluid encapsule le lecteur SoundFont 2 de Peter Hannape, FluidSynth : *fluidEngine* pour instancier un moteur FluidSynth, *fluidSetInterpMethod* pour fixer la méthode d'interpolation d'un canal dans un moteur FluidSynth, *fluidLoad* pour charger des SoundFonts, *fluidProgramSelect* pour assigner des presets d'un SoundFont à un canal MIDI d'un moteur FluidSynth, *fluidNote* pour jouer une note sur un canal MIDI d'un moteur FluidSynth, *fluidCCi* pour envoyer un message de contrôleur au temps-i sur un canal MIDI d'un moteur FluidSynth, *fluidCCk* pour envoyer un message de contrôleur au taux k sur un canal MIDI d'un moteur FluidSynth. *fluidControl* pour jouer et contrôler les Soundfonts chargés (en utilisant des messages MIDI 'bruts'), *fluidOut* pour recevoir de l'audio depuis un seul moteur FluidSynth, et *fluidAllOut* pour recevoir de l'audio depuis tous les moteurs FluidSynth.

- *fluidAllOut*
- *fluidCCi*
- *fluidCCk*
- *fluidControl*
- *fluidEngine*
- *fluidLoad*
- *fluidNote*
- *fluidOut*
- *fluidProgramSelect*
- *fluidSetInterpMethod*

### "Anciens" opcodes Soundfont

Ces opcodes peuvent aussi employer des soundfonts pour générer du son. *sfplay*, etc. ont été créés dans un but -- utiliser les échantillons dans les SoundFonts. Les opcodes fluid ont été créés dans un autre but -- utiliser les SoundFonts plus ou moins comme ils ont été conçus, c'est-à-dire en utilisant des mappages de clavier, des couches, un traitement interne, etc.

- *sflist*
- *sfinstr*
- *sfinstr3*
- *sfinstr3m*
- *sfinstrm*
- *sfload*

- *sfpassign*
- *sfplay*
- *sfplay3*
- *sfplay3m*
- *sfplaym*
- *sflooper*
- *sfplist*
- *sfpreset*

## Synthèse par balayage

La synthèse par balayage (scanned synthesis) est une variante des modèles physiques, dans laquelle un réseau de masses connectées par des ressorts est utilisé pour générer une forme d'onde dynamique. L'opcode *scanu* définit le réseau de masses/ressorts et le met en mouvement. L'opcode *scans* suit un chemin prédéfini (une trajectoire) à travers le réseau et donne en sortie la forme d'onde détectée. Plusieurs instances de *scans* peuvent suivre différents chemins à travers le même réseau.

Ce sont des algorithmes de modélisation mécanique hautement efficaces à la fois pour la synthèse et l'animation sonore via un traitement algorithmique. Il vaut mieux les utiliser en temps réel. Ainsi, la sortie est utile soit directement pour l'audio, soit comme valeurs de contrôleur pour d'autres paramètres.

L'implémentation dans Csound ajoute le support pour un chemin de balayage ou matrice. Essentiellement, ceci offre la possibilité de reconnecter les masses dans d'autres configurations, provoquant une propagation du signal assez différente. Elles ne doivent pas nécessairement être connectées à leurs voisines directes. La matrice a essentiellement l'effet de « modeler » la surface en une forme radicalement différente.

Pour produire les matrices, le format du tableau est direct. Par exemple, pour 4 masses nous avons la grille suivante qui décrit les connexions possibles :

	1	2	3	4
1				
2				
3				
4				

Chaque fois que deux masses sont connectées, le point qu'elles définissent vaut 1. Si deux masses ne sont pas connectées, le point qu'elles définissent vaut alors 0. Par exemple, une corde unidirectionnelle a les connexions suivantes : (1,2), (2,3), (3,4). Si elle est bidirectionnelle, elle a aussi (2,1), (3,2), (4,3). Pour la corde unidirectionnelle, la matrice est :

	1	2	3	4
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	0	0	0	0



Le format de tableau ci-dessus pour la matrice de connexion n'est donné que par commodité conceptuelle. Les valeurs actuellement montrées dans le tableau sont obtenues par *scans* depuis un fichier ASCII en utilisant *GEN23*. Le fichier ASCII lui-même est créé à partir du tableau modèle ligne par ligne. Ainsi, le fichier ASCII pour le tableau de l'exemple montré ci-dessus devient :

0100001000010000

Cet exemple de matrice est très simple et très petit. En pratique, la plupart des instruments de synthèse par balayage utiliseront bien plus que quatre masses, et donc leurs matrices seront bien plus grandes et plus complexes. Voir l'exemple dans la documentation de *scans*.

Prière de noter que les tables d'onde dynamiques générées sont très instables. Certaines valeurs de masses, de centrage, et d'amortissement peuvent provoquer une « explosion » du système et l'apparition des sons les plus intéressants sur vos haut-parleurs.

Le supplément de ce manuel contient un tutoriel sur la synthèse par balayage. Le tutoriel, des exemples, et d'autres informations sur la synthèse par balayage sont disponibles sur la page Scanned Synthesis à csounds.com [<http://www.csounds.com/scanned/>].

La synthèse par balayage a été développée par Bill Verplank, Max Mathews et Rob Shaw à Interval Research entre 1998 et 2000.

Les opcodes qui implémentent la synthèse par balayage sont :

- *scanhammer*
- *scans*
- *scantable*
- *scanu*
- *xscanmap*
- *xscans*
- *xscansmap*
- *xscanu*

## Accès aux tables

Les opcodes qui permettent l'accès aux tables sont :

- *oscill*
- *oscilli*
- *osciln*
- *oscilx*
- *table*
- *table3*
- *tablei*

Les opcodes se terminant par 'i' implémentent l'interpolation linéaire et les opcodes se terminant par '3' implémentent l'interpolation cubique.

Les opcodes suivants implémentent la lecture/écriture rapide dans une table sans en tester les limites :

- *tab*
- *tab\_i*
- *tabw*
- *tabw\_i*

Voir les sections *Requêtes de Table*, *Opérations de Lecture/Ecriture de Table* et *Lecture de Table avec Sélection Dynamique* pour d'autres opérations de table.



### Note

Bien que des tables avec une taille qui n'est pas une puissance de deux puissent être créées en utilisant une taille négative (voir *instruction de partition f*), certains opcodes ne les accepteront pas.

## Synthèse par terrain d'ondes

L'opcode qui utilise la synthèse par terrain d'ondes est : *wterrain*.

## Modèles physiques par guide d'onde

Les opcodes qui implémentent les modèles physiques par guide d'onde sont :

- *pluck*
- *repluck*
- *wgbow*
- *wgbowedbar*
- *wgbrass*
- *wgclar*
- *wgflute*
- *wgpluck*
- *wgpluck2*
- *wguide1*
- *wguide2*

---

# Entrée et sortie de signal

## Entrées et sorties fichier

Les opcodes pour les entrées et sorties fichier sont :

- Ouverture/fermeture de fichier : *fiopen* et *ficlose*.
- Sortie fichier : *dumpk*, *dumpk2*, *dumpk3*, *dumpk4*, *fout*, *fouti*, *foutir*, *foutk* et *hdf5write*
- Entrée fichier : *readk*, *readk2*, *readk3*, *readk4*, *fin*, *fini* et *fink*
- Utilitaires à utiliser avec les opcodes *fout* : *clear*, *vincr*
- Impression dans un fichier : *fprints* et *fprintks*

## Entrée de signal

Les opcodes qui reçoivent des signaux audio sont :

- Entrée synchrone : *in*, *in32*, *inch*, *inh*, *ino*, *inq*, *inrg*, *ins* et *inx*
- Flux de fichier : *diskin*, *diskin2*, *soundin* et *hdf5read*
- Canal d'entrée défini par l'utilisateur : *invalue*
- Flux d'entrée : *soundin*
- Entrée WebSocket : *websocket*
- Entrée directe dans *zak* : *inz*

Voir la section *Bus Logiciel* pour les entrées et les sorties au moyen de l'API.

*mp3in* permet la lecture des fichiers mp3, qui n'est pas supportée par les méthodes de lecture usuelles dans Csound.

## Sortie de signal

Les opcodes qui écrivent des signaux audio sont :

- Sortie synchrone : *out*, *out32*, *outc*, *outch*, *outh*, *outo*, *outrg*, *outq*, *outq1*, *outq2*, *outq3*, *outq4*, *outs*, *outs1*, *outs2*, *outx* et *hdf5write*
- Flux de sortie : *soundout* et *soundouts*
- Canal de sortie défini par l'utilisateur : *outvalue*
- Sortie directe depuis *zak* : *outz*
- Sortie WebSocket : *websocket*

L'opcode *monitor* peut être utilisé pour surveiller la sortie complète de *csound* (trame de sortie *spout*).

Voir la section *Bus Logiciel* pour les entrées et les sorties au moyen de l'API.

## Bus logiciel

Csound implémente un bus logiciel pour le routage interne ou le routage vers des logiciels externes en appelant l'API de Csound.

Les opcodes pour utiliser le bus logiciel sont :

- *chn\_k*
- *chn\_a*
- *chn\_S*
- *chnclear*
- *chnexport*
- *chnmix*
- *chnparams*

## Impression et affichage

Les opcodes pour imprimer et afficher des valeurs sont :

- *dispfft*
- *display*
- *flashtxt*
- *print*
- *printf*
- *printf\_i*
- *printk*
- *printk2*
- *printks*
- *prints*
- *printarray*
- *ftprint*

## Requêtes sur les fichiers sons

Les opcodes qui demandent de l'information sur les fichiers sont :

- *filelen*

- *flenchnls*
- *filepeak*
- *filesr*
- *filevalid*

---

# Modificateurs de signal

## Modificateurs d'amplitude et traitement des dynamiques

Les opcodes qui modifient l'amplitude sont :

- *balance*
- *compress*
- *clip*
- *dam*
- *gain*

L'opcode *0dbfs* facilite la manipulation d'amplitude en supprimant la nécessité d'utiliser des valeurs d'échantillon explicites.

## Convolution et morphing

Les opcodes qui font la convolution et le morphing de signaux sont :

- *convolve* aussi nommé *convle*
- *cross2*
- *dconv*
- *ftconv*
- *ftmorf*
- *pconvolve*

## Retard

### Retards fixes

- *delay*
- *delayl*
- *delayk*

### Lignes à retard

- *delayr*
- *delayw*

- *deltap*
- *deltap3*
- *deltapi*
- *deltapn*
- *deltapx*
- *deltapxw*

## Retards variables

- *vdelay*
- *vdelay3*
- *vdelayx*
- *vdelayxs*
- *vdelayxq*
- *vdelayxw*
- *vdelayxwq*
- *vdelayxws*

## Retards multiples

- *multitap*

# Panoramique et spatialisation

## Spatialisation d'amplitude

- *locsend*
- *locsig*
- *pan*
- *pan2*
- *space*
- *spdist*
- *spsend*

## Spatialisation 3D avec simulation d'acoustique des salles

- *spat3d*

- *spat3di*
- *spat3dt*

## Panning d'amplitude à base vectorielle

- *vbap16*
- *vbap16move*
- *vbap4*
- *vbap4move*
- *vbap8*
- *vbap8move*
- *vbaplsinit*
- *vbapz*
- *vbapzmove*

## Spatialisation binaurale

- *hrtfer*
- *hrtfmove*
- *hrtfmove2*
- *hrtfstat*

## Ambisonics

- *bformdec1*
- *bformenc1*

## Réverbération

Les opcodes qu'on peut utiliser pour la réverbération sont :

- *alpass*
- *babo*
- *comb*
- *freeverb*
- *nestedap*
- *nreverb* (aussi appelé *reverb2*)



- *reverb*
- *reverb**sc*
- *valpass*
- *vcomb*

## Opérateurs du niveau échantillon

Les opérateurs que l'on peut utiliser pour modifier les signaux sont :

- *a(k)*
- *denorm*
- *diff*
- *downsamp*
- *fold*
- *i(k)*
- *integ*
- *interp*
- *k(i)*
- *ntrpol*
- *samphold*
- *upsamp*
- *vaget*
- *vaset*

## Limiteurs de signal

Les opcodes que l'on peut utiliser pour limiter des signaux sont :

- *limit*
- *mirror*
- *wrap*

## Effets spéciaux

Les opcodes qui génèrent des effets spéciaux sont :

- *distort*

- *distort1*
- *exciter*
- *flanger*
- *harmon*
- *phaser1*
- *phaser2*

## Filtres standard

### Filtres passe-bas à résonance

- *areson*
- *lowpass2*
- *lowres*
- *lowresx*
- *lpf18*
- *moogvcf*
- *moogladder*
- *mvclpf1*
- *mvclpf2*
- *mvclpf3*
- *mvclpf4*
- *reson*
- *resonr*
- *resonx*
- *resony*
- *resonz*
- *rezy*
- *statevar*
- *svfilter*
- *tbvcf*
- *vlowres*

- *bqrez*

## Filtres standard

- Filtres passe-haut : *atone*, *atonex*, *mvchpf*
- Filtres passe-bas : *tone*, *tonex*
- Filtres biquadratiques : *biquad* et *biquada*.
- Filtres de Butterworth : *butterbp*, *butterbr*, *butterhp*, *butterlp* (qui sont aussi appelés *butbp*, *butbr*, *buthp*, *butlp*)
- Filtres généraux : *clfilt*

## Filtres avec rétroaction sans retard (analogues virtuels)

- *zdf\_1pole*
- *zdf\_1pole\_mode*
- *zdf\_2pole*
- *zdf\_2pole\_mode*
- *zdf\_ladder*
- *diode\_ladder*
- *K35\_hpf*
- *K35\_lpf*

## Filtres de signal de contrôle

- *aresonk*
- *atonek*
- *lineto*
- *port*
- *portk*
- *resonk*
- *resonxk*
- *tlineto*
- *tonek*
- *sc\_lag*
- *sc\_lagud*

## Filtres spécialisés

### Filtres passe-haut

- *dcblock*
- *dcblock2*

### Egaliseurs paramétriques

- *pareq*
- *rbjeq*
- *eqfil*

### Autres filtres

- *nlfilt*
- *filter2*
- *fofilter*
- *hilbert*
- *mode*
- *zfilter2*

## Guides d'onde

Les opcodes qui utilisent des guides d'onde pour modifier un signal sont :

- *streson*
- *wguide1*
- *wguide2*

## Distorsion non-linéaire et distorsion de phase

Ces opcodes peuvent exécuter de façon dynamique une distorsion non-linéaire ou une distorsion de phase. Il diffèrent des méthodes traditionnelles de distorsion non-linéaire basées sur une table, en calculant directement la fonction de transfert avec un ou plusieurs paramètres variables pour modifier l'importance ou les résultats de la distorsion. La plupart de ces opcodes peuvent être utilisés sur un signal audio (pour la distorsion non-linéaire) ou sur un phaseur (pour la distorsion de phase) mais ils ont tendance à fonctionner au mieux pour une de ces applications.

Ces opcodes sont adaptés à la distorsion non-linéaire :

- *chebyshevpoly*

- *clip*
- *distort*
- *distort1*
- *polynomial*
- *powershape*

Ces opcodes sont adaptés à la distorsion de phase :

- *pdclip*
- *pdhalf*
- *pdhalfy*

---

# Contrôle d'instrument

## Contrôle d'horloge

Les opcodes pour démarrer et arrêter les horloges internes sont :

- *clockoff*
- *clockon*

Ces horloges comptent le temps CPU. On dispose de 32 horloges indépendantes. On peut utiliser l'opcode *readclock* pour lire les valeurs courantes d'une horloge. Voir *Lecture du Temps* pour d'autres opcodes de chronométrage.

## Valeurs conditionnelles

Les opcodes pour les valeurs conditionnelles sont `==`, `>=`, `>`, `<`, `<=` et `!=`.

## Instructions de contrôle de durée

Les opcodes que l'on peut utiliser pour manipuler la durée d'une note sont :

- *ihold*
- *turnoff*
- *turnoff2*
- *turnon*

Pour d'autres contrôles d'instrument en temps réel voir *Contrôle de l'exécution en temps réel* et *Appel d'instrument*.

## Widgets FLTK et contrôleurs GUI

Les widgets permettent de dessiner une Interface Utilisateur Graphique (GUI) personnalisée pour contrôler un orchestre en temps réel. Ils sont dérivés de la bibliothèque libre FLTK (Fast Light ToolKit). Cette bibliothèque est une des plus rapides parmi les bibliothèques disponibles, supporte OpenGL et devrait être compatible avec différentes plates-formes (Windows, Linux, Unix et Mac OS). Le sous-ensemble de FLTK implémenté dans Csound fournit les types d'objets suivants :

Conteneurs

Les *Conteneurs FLTK* sont des widgets qui contiennent d'autres widgets tels que des panneaux, des fenêtres, etc. Csound fournit les objets conteneurs suivants :

- Panneaux
- Zones déroulantes
- Paquets
- Onglets
- Groupes

Valuateurs Les objets les plus utiles sont appelés *Valuateurs FLTK*. Ces objets permettent à l'utilisateur de modifier les valeurs des paramètres de synthèse en temps réel. Csound fournit les objets valuateurs suivants :

- Réglettes
- Boutons rotatifs
- Molettes
- Champs texte
- Joysticks
- Compteurs

Autres widgets Il y a d'autres *widgets FLTK* qui ne sont ni des valuateurs ni des conteneurs :

- Boutons
- Bancs de boutons
- Etiquettes
- Détection Clavier et Souris

Il y a aussi d'autres opcodes utiles pour modifier l'apparence des widgets :

- Mettre à jour la valeur d'un widget.
- Choisir les couleurs principale et de sélection d'un widget.
- Choisir le type, la taille et la couleur de police des widgets.
- Redimensionner un widget.
- Cacher et Montrer un widget.

Il y a aussi ces *opcodes généraux* qui permettent les actions suivantes :

- Lancer le processus léger (thread) des widgets : *FLrun*
- Charger des instantanés contenant l'état de tous les valuateurs d'un orchestre : *FLgetsnap* et *FLloadsnap*.
- Sauvegarder des instantanés contenant l'état de tous les valuateurs d'un orchestre : *FLsavesnap* et *FLsetsnap*
- Fixer le groupe d'instantanés d'un valuateur déclaré : *FLsetSnapGroup*

Ci-dessous un exemple simple de code Csound pour créer une fenêtre. Noter que tous les opcodes sont de taux-init et ne doivent être appelés qu'une seule fois par session. La meilleure manière de les utiliser est de les placer dans la section d'en-tête de l'orchestre, avant tout instrument. Même s'il n'est pas interdit de les placer dans un instrument, cela peut conduire à des résultats imprévisibles si l'instrument est appelé plus d'une fois.

Chaque conteneur est fait d'un couple d'opcodes : le premier indique le début du bloc du conteneur et le deuxième indique la fin du bloc du conteneur. Certains blocs de conteneur peuvent être imbriqués mais il

ne peuvent pas se chevaucher. Après avoir défini tous les conteneurs, il faut lancer un processus léger de widgets en utilisant l'opcode spécial *FLrun* qui ne prend pas d'argument.

```
<CsoundSynthesizer>
<CsOptions>
; Sélectionner les options audio/midi ici, en fonction de la plate-forme
; Sortie audio   Entrée audio   Pas de messages
    -odac         -iadc         -d             ;; E/S audio en Temps Réel
; Pour une sortie différée ne garder que la ligne ci-dessous :
; -o linseg.wav -W ;; pour une sortie dans un fichier sur toute plate-forme
</CsOptions>
<CsInstruments>
;*****
sr=48000
kr=480
ksmps=100
nchnls=1

;*** Il est recommandé de placer presque tout le code GUI dans la
;*** section d'en-tête de l'orchestre

    FLpanel          "Panel1",450,550 ,***** début du conteneur
; placer ici quelques widgets
    FLpanelEnd       ,***** fin du conteneur

    FLrun            ,***** lance le thread FLTK, toujours requis !
instr 1
; placer ici du code de synthèse
endin
;*****
</CsInstruments>
<CsScore>
f 0 3600 ; table bidon pour l'entrée en temps réel
e

</CsScore>
</CsoundSynthesizer>
```

Le code précédent crée simplement un panneau (une fenêtre vide car aucun widget n'est défini à l'intérieur du conteneur).

L'exemple suivant crée deux panneaux et insère une réglette dans chacun d'entre eux :

```
<CsoundSynthesizer>
<CsOptions>
; Sélectionner les options audio/midi ici, en fonction de la plate-forme
; Sortie audio   Entrée audio   Pas de messages
    -odac         -iadc         -d             ;; E/S audio en Temps Réel
; Pour une sortie différée ne garder que la ligne ci-dessous :
; -o linseg.wav -W ;; pour une sortie dans un fichier sur toute plate-forme
</CsOptions>
<CsInstruments>
;*****
sr=48000
kr=480
ksmps=100
nchnls=1

    FLpanel          "Panel1",450,550,100,100 ,***** début de conteneur
gk1,iha FLslider     "FLslider 1", 500, 1000, 0 ,1, -1, 300,15, 20,50
    FLpanelEnd       ,***** fin de conteneur

    FLpanel          "Panel2",450,550,100,100 ,***** début de conteneur
gk2,ihb FLslider     "FLslider 2", 100, 200, 0 ,1, -1, 300,15, 20,50
```



```

        FLpanelEnd      ;***** fin de conteneur

        FLrun           ;***** lance le thread FLTK, toujours requis !

instr 1
; les variables gk1 et gk2 qui contiennent les valeurs de sortie des valuateurs
; définis précédemment, peuvent être utilisées à l'intérieur des instruments
printk2 gk1
printk2 gk2 ; imprime les valeurs des valuateurs chaque fois qu'elles changent
endin
;*****
</CsInstruments>
<CsScore>
f 0 3600 ; table bidon pour l'entrée en temps réel
e

</CsScore>
</CsoundSynthesizer>

```

Tous les opcodes de widget sont des opcodes de taux-init, même si les valuateurs donnent en sortie des variables de taux-k. Ceci est dû au fait qu'un processus léger indépendant est exécuté sur la base d'un mécanisme de fonctions de rappel. Cela permet de consommer très peu de ressources système car on évite la scrutation. (A la différence des autres opcodes de contrôleurs basés sur le MIDI). On peut ainsi utiliser n'importe quel nombre de fenêtres et de valuateurs sans dégrader l'exécution en temps réel.

## Conteneurs FLTK

Les opcodes pour les conteneurs FLTK sont :

- *FLgroup*
- *FLgroupEnd*
- *FLpack*
- *FLpackEnd*
- *FLpanel*
- *FLpanelEnd*
- *FLscroll*
- *FLscrollEnd*
- *FLtabs*
- *FLtabsEnd*

## Valuateurs FLTK

Les opcodes pour les valuateurs FLTK sont :

- *FLcount*
- *FLjoy*
- *FLknob*
- *FLroller*

- *FLslider*
- *FLtext*

## Autres widgets FLTK

Les opcodes des autres widgets FLTK sont :

- *FLbox*
- *FLbutBank*
- *FLbutton*
- *FLexecButton*
- *FLkeyIn*
- *FLhvsBox*
- *FLhvsBoxSetValue*
- *FLmouse*
- *FLprintk*
- *FLprintk2*
- *FLslidBnk*
- *FLslidBnk2*
- *FLslidBnkGetHandle*
- *FLslidBnkSet*
- *FLslidBnk2Set*
- *FLslidBnk2Setk*
- *FLvalue*
- *FLvkeybd*
- *FLvslidBnk*
- *FLvslidBnk2*
- *FLxyin*

## Modifier l'apparence des widgets FLTK

Les opcodes suivants modifient l'apparence des widgets FLTK :

- *FLcolor*
- *FLcolor2*

- *FLhide*
- *FLlabel*
- *FLsetAlign*
- *FLsetBox*
- *FLsetColor*
- *FLsetColor2*
- *FLsetFont*
- *FLsetPosition*
- *FLsetSize*
- *FLsetText*
- *FLsetTextColor*
- *FLsetTextSize*
- *FLsetTextType*
- *FLsetVal\_i*
- *FLsetVal*
- *FLshow*

## Opcodes généraux relatifs aux widgets FLTK

Les opcodes généraux relatifs aux widgets FLTK sont :

- *FLgetsnap*
- *FLloadsnap*
- *FLrun*
- *FLsavesnap*
- *FLsetsnap*
- *FLupdate*
- *FLsetSnapGroup*

## Appel d'instrument

Les opcodes que l'on peut utiliser pour créer des événements de partition depuis un orchestre sont :

- *event*
- *event\_i*

- *scoreline\_i*
- *scoreline*
- *schedule*
- *schedwhen*
- *schedkwhen*
- *schedkwhennamed*

L'opcode *mute* peut être utilisé pour rendre silencieux/sonore un instrument pendant une exécution.

Les définitions d'instrument peuvent être supprimées au moyen de l'opcode *remove*.

## Contrôle séquentiel d'un programme

Les opcodes pour modifier l'ordre d'exécution des instructions de l'orchestre sont :

- *cgoto*
- *cigoto*
- *ckgoto*
- *cngoto*
- *elseif*
- *else*
- *endif*
- *goto*
- *if*
- *igoto*
- *kgoto*
- *tigoto*
- *timeout*

Les opcodes pour créer des structures de boucle sont :

- *loop\_ge*
- *loop\_gt*
- *loop\_le*
- *loop\_lt*
- *until*

- *while*



### Avertissement

Certains de ces opcodes fonctionnent au taux-i même s'ils contiennent des comparaisons aux taux-k ou -a. Voir la section *Réinitialisation*.

## Contrôle de l'exécution en temps réel

Les opcodes qui surveillent et contrôlent l'exécution en temps réel sont :

- *active*
- *cpuprc*
- *maxalloc*
- *prealloc*
- *jacktransport*

Le processus csound en cours peut être terminé au moyen de *exitnow*.

## Initialisation et réinitialisation

Les opcodes utilisés pour l'initialisation des variables sont :

- *init*
- *tival*
- *=*
- *passign*
- *pset*

Les opcodes qui peuvent générer une autre passe d'initialisation sont :

- *reinit*
- *rigoto*
- *rireturn*

L'opcode *p* peut être utilisé pour lire les valeurs des p-champs aux taux-i ou -k.

*nstrnum* retourne le numéro d'instrument d'un instrument nommé.



### Note

Noter qu'un instrument peut modifier le paramètre p3 (durée) pendant l'initialisation. Par exemple ces instructions

```
iattack = 0.02
irelease = 0.04
```

```
isustain = p3  
p3 = iattack + isustain + irelease
```

sont valides.

## Détection et contrôle

### Widgets TCL/TK

- *button*
- *checkbox*
- *control*
- *setctrl*

### Détection clavier et souris

- *sensekey* (aussi appelé *sense*)
- *xyin*

### Suiveurs d'enveloppe

- *follow*
- *follow2*
- *peak*
- *rms*

### Estimation de tempo et de hauteur

- *ptrack*
- *pitch*
- *pitchamdf*
- *tempest*

### Tempo et séquençement

- *tempo*
- *miditempo*
- *tempoval*
- *seqtime*
- *seqtime2*

- *trigger*
- *trigseq*
- *timedseq*
- *changed*

## Système

- *getcfg*

## Contrôle de la partition

- *rewindscore*
- *setscorepos*

## Piles

Csound implémente une pile globale qui peut être manipulée par les opcodes suivants :

- *stack*
- *pop*
- *push*
- *pop\_f*
- *push\_f*

## Contrôle de sous-instrument

Ces opcodes permettent la définition et l'utilisation d'un sous-instrument :

- *subinstr*
- *subinstrinit*

Voir aussi les sections *UDO* et *Macros d'Orchestre* pour des fonctionnalités similaires.

## Lecture du temps

Les opcodes que l'on peut utiliser pour lire des valeurs temporelles sont :

- *readclock*
- *rtclock*
- *timeinstk*
- *timeinsts*

- *times*
- *timek*

On peut obtenir la date du système au moyen de :

- *date* - Retourne le nombre de secondes écoulées depuis le 1er janvier 1970.
- *dates* - Retourne sous format chaîne la date et le temps spécifiés.

On peut aussi mettre en place des compteurs au moyen de *clockoff* et de *clockon*.



---

# Contrôle des tables de fonction

Se reporter aux sections *Instruction de partition f*, *ftgen*, *ftgentmp*, *ftgenonce* et *Routines GEN* pour savoir comment créer des tables.

On peut supprimer des tables de la mémoire au moyen de l'opcode *ftfree*.

Les tables requièrent par défaut une taille qui est une puissance de deux. On peut cependant générer des tables de n'importe quelle taille en spécifiant celle-ci comme un nombre négatif (voir l'*instruction de partition f*).



## Note

Certains opcodes n'acceptent pas des tables dont la taille n'est pas une puissance de deux, car ceci peut être une nécessité pour le traitement interne.

Pour savoir comment accéder aux tables, consulter la section *Accès aux Tables*.

Les tables à utiliser avec l'opcode *loscilx* peuvent être chargées au moyen de *sndload*.

## Requêtes sur une table

Les opcodes qui permettent d'obtenir des informations sur une table sont :

- Pour les tables chargées depuis un fichier son (au moyen de *GEN01*) : *fchnls*, *ftcps*, *ftlen*, *ftlptim* et *ftsr*
- Pour n'importe quelle table : *nsamp*, *ftlen*, *tbleng*

L'opcode *tabsum* calcule la somme des valeurs dans une table.

## Opérations de lecture/écriture de table

Les opcodes pour la lecture et l'écriture dans une table sont :

- *ftloadk*
- *ftload*
- *ftsavek*
- *ftsave*
- *tablecopy*
- *tablegpw*
- *tableicopy*
- *tableigpw*
- *tableimix*
- *tableiw*
- *tablemix*

- *tablera*
- *tablew*
- *tablewa*
- *tablewkt*
- *tabmorph*
- *tabmorpha*
- *tabmorphak*
- *tabmorphi*
- *tabrec*
- *tabplay*
- *ftmorf*
- *ftslice*
- *ftprint*

Les valeurs d'une table peuvent être lues depuis une expression grâce à la famille d'opcodes *tb*.

Plusieurs oscillateurs sont en fait des lecteurs de table spécialisés. Voir la section *Oscillateurs Elémentaires*.

## Lecture de table avec sélection dynamique

Les opcodes qui permettent de sélectionner des tables dynamiquement (au taux-k) sont :

- *tableikt*
- *tablekt*
- *tablexkt*

---

# Opérations mathématiques

## Conversion d'amplitude

Les opcodes pour opérer des conversions entre différentes mesures d'amplitude sont :

- *ampdb*
- *ampdbfs*
- *db*
- *dbamp*
- *dbfsamp*

Utiliser *rms* pour trouver la valeur de la moyenne quadratique d'un signal. Voir aussi *0dbfs* pour un autre moyen de gérer les amplitudes dans csound.

## Opérations arithmétiques et logiques

Les opcodes qui effectuent les opérations arithmétiques et logiques sont : -, +, &&, ||, \*, /, ^, % et *cmp*.

Voir aussi la section *Valeurs Conditionnelles* et la famille des opcodes *if* pour l'utilisation des opérateurs logiques.

## Comparateurs et accumulateurs

Les opcodes suivants effectuent la comparaison entre des signaux de taux-a ou de taux-k, trouvent les maxima ou les minima, ou accumulent les résultats de plusieurs calculs ou comparaisons :

- *max*
- *max\_k*
- *maxabs*
- *maxabsaccum*
- *maxaccum*
- *min*
- *minabs*
- *minabsaccum*
- *minaccum*
- *vincr*
- *clear*
- *cmp*

## Fonctions mathématiques

Les opcodes qui réalisent les fonctions mathématiques sont :

- *abs*
- *ceil*
- *exp*
- *floor*
- *frac*
- *int*
- *log*
- *log10*
- *logbtwo*
- *pow*
- *powershape*
- *powoftwo*
- *round*
- *sqrt*

## Opcodes équivalents à des fonctions

Les opcodes suivants sont équivalents à des fonctions mathématiques :

- *chebyshevpoly*
- *divz*
- *mac*
- *maca*
- *polynomial*
- *pow*
- *product*
- *sum*
- *taninv2*

## Fonctions aléatoires

Les opcodes qui effectuent des fonctions aléatoires sont :

- *birnd*
- *rnd*

Voir la section *Générateurs de Nombres Aléatoires (Bruit)* pour les opcodes qui génèrent des signaux aléatoires.

## Fonctions trigonométriques

Les opcodes qui effectuent les fonctions trigonométriques sont :

- *cos*, *cosh* et *cosinv*
- *sin*, *sinh* et *sininv*
- *tan*, *tanh*, *taninv* et *taninv2*.

# Opcodes d'algèbre linéaire

Opcodes d'algèbre linéaire — Arithmétique scalaire, vectorielle et matricielle sur des valeurs réelles et complexes.

## Description

Opcodes du greffon `linear_algebra`.

Ces opcodes implémentent plusieurs opérations d'algèbre linéaire, depuis l'arithmétique scalaire, vectorielle et matricielle jusqu'aux décompositions en valeurs propres basées sur la décomposition QR. Les opcodes sont conçus pour le traitement numérique du signal, et bien sûr pour d'autres opérations mathématiques, dans le langage d'orchestre de Csound.

L'implémentation numérique utilise la bibliothèque `gmm++` de [home.gna.org/getfem/gmm\\_intro](http://home.gna.org/getfem/gmm_intro) [[http://home.gna.org/getfem/gmm\\_intro](http://home.gna.org/getfem/gmm_intro)].



### Avertissement

Pour les applications avec des variables f-sig, l'arithmétique sur les tableaux ne peut être exécutée que si le f-sig est "actuel", car le taux-f est une fraction du taux-k ; ce caractère actuel peut être déterminé avec l'opcode `la_k_current_f`.

Pour les applications que utilisent des affectations entre vecteurs réels et variables de taux-a, l'arithmétique sur les tableaux ne peut être exécutée que si les vecteurs sont "actuels", car la taille du vecteur peut être un multiple entier de ksmpps ; ce caractère actuel peut être déterminé au moyen de l'opcode `la_k_current_vr`.

**Tableau 4. Types de données de l'algèbre linéaire**

Type mathématique	Code	Type(s) de Csound correspondant(s)
scalaire réel	r	variable de taux-i ou de taux-k
scalaire complexe	c	paire de variables de taux-i ou de taux-k, par exemple "kr, ki"
vecteur réel	vr	variable de taux-i contenant l'adresse d'un tableau
vecteur réel	a	variable de taux-a
vecteur réel	t	numéro d'une table de fonction
vecteur complexe	vc	variable de taux-i contenant l'adresse d'un tableau
vecteur complexe	f	variable fsig
matrice réelle	mr	variable de taux-i contenant l'adresse d'un tableau
matrice complexe	mc	variable de taux-i contenant l'adresse d'un tableau

Tous les tableaux sont indexés à partir de 0 ; le premier indice parcourt les lignes pour donner les colonnes, le deuxième indice parcourt les colonnes pour donner les éléments.

Un tableau peut avoir pour code de type vr, vc, mr ou mc et il est stocké dans un objet de taux-i. Dans le code de l'orchestre, un tableau est passé comme une variable MYFLT de taux-i qui contient l'adresse de l'objet tableau, celui-ci étant stocké dans l'espace d'allocation de l'instance de l'opcode. Bien que les variables tableau soient de taux-i, leurs valeurs et même leur forme peuvent être modifiées au taux-i ou au taux-k.

Les tableaux sont libérés automatiquement lorsque leur instrument est libéré.

1. "la" pour "famille d'opcode d'algèbre linéaire".
2. "i" ou "k" pour le taux d'exécution.
3. Code(s) de type (voir ci-dessus) pour la ou les valeurs de sortie, seulement si le type n'est pas déduit implicitement des valeurs en entrée.
4. Nom d'opération : nom mathématique usuel (de préférence) ou abréviation.
5. Code(s) de type pour les valeurs en entrée, s'ils ne sont pas implicites.

# Syntaxe

<code>ivr</code>	<code>la_i_vr_create</code>	<code>irows</code>
Crée un vecteur réel de <i>irows</i> lignes.		
<code>ivc</code>	<code>la_i_vc_create</code>	<code>irows</code>
Crée un vecteur complexe de <i>irows</i> lignes.		
<code>imr</code>	<code>la_i_mr_create</code>	<code>irows, icolumns [, odiagonal]</code>
Crée une matrice réelle de <i>irows</i> lignes et <i>icolumns</i> colonnes, avec une valeur facultative sur sa diagonale.		
<code>imc</code>	<code>la_i_mc_create</code>	<code>irows, icolumns [, odiagonal_r, odiagonal_i]</code>
Crée une matrice complexe de <i>irows</i> lignes et <i>icolumns</i> colonnes, avec une valeur facultative sur sa diagonale.		

```

irows          la i size vr          ivr

```

```

irows      la i size vc      ivc

```

```
irows, icolumns      la i size mr      imr
```

```
irows, icolumns      la_i_size_mc      imc
```

kfiscurrent	la k current f	fsig
-------------	----------------	------

kvriscurrent	la k current vr	ivr
--------------	-----------------	-----

```
la i print vr      ivr
```

```
la i print vc      ivc
```

```
la i print mr      imr
```

```
la i print mc      imc
```

## de tableau

```
ivr                                la i assign vr                                ivr
```

Affecte la valeur du vecteur réel à droite au vecteur réel à gauche, au taux- $i$ .

`ivr`                      `la_k_assign_vr`                      `ivr`

Affecte la valeur du vecteur réel à droite au vecteur réel à gauche, au taux-k.

`ivc`                      `la_i_assign_vc`                      `ivc`

```
ivc                                la_k_assign_vc                        ivr
```

```
imr                                la_i_assign_mr                                imr
```

```
imr                                la_k_assign_mr                                imr
```

```
imc                                la_i_assign_mc                                imc
```

```
imc                                la_k_assign_mc                                imr
```





Les affectations vers des vecteurs à partir de variables de taux-a, ou vers des variables de taux-a à partir de vecteurs, seront exécutées de manière incrémentielle, un bloc de ksmps éléments par période-k. C'est pourquoi l'arithmétique vectorielle sur ces vecteurs ne peut être pratiquées que si ceux-ci sont actuels, selon la détermination par l'opcode *la\_k\_current\_vr*.

ivr	<b>la_k_assign_a</b>	asig
ivr	<b>la_i_assign_t</b>	itablenumber
ivr	<b>la_k_assign_t</b>	itablenumber
ivc	<b>la_k_assign_f</b>	fsig
asig	<b>la_k_a_assign</b>	ivr
itablenum	<b>la_i_t_assign</b>	ivr
itablenum	<b>la_k_t_assign</b>	ivr
fsig	<b>la_k_f_assign</b>	ivc

## Remplissage des Tableaux par des Éléments Aléatoires

ivr	<b>la_i_random_vr</b>	[ifill_fraction]
ivr	<b>la_k_random_vr</b>	[kfill_fraction]
ivc	<b>la_i_random_vc</b>	[ifill_fraction]
ivc	<b>la_k_random_vc</b>	[kfill_fraction]
imr	<b>la_i_random_mr</b>	[ifill_fraction]
imr	<b>la_k_random_mr</b>	[kfill_fraction]
imc	<b>la_i_random_mc</b>	[ifill_fraction]
imc	<b>la_k_random_mc</b>	[kfill_fraction]

## Accès aux éléments d'un tableau

ivr	<b>la_i_vr_set</b>	irow, ivalue
kvr	<b>la_k_vr_set</b>	krow, kvalue
ivc	<b>la_i_vc_set</b>	irow, ivalue_r, ivalue_i
kvc	<b>la_k_vc_set</b>	krow, kvalue_r, kvalue_i
imr	<b>la_i_mr_set</b>	irow, icolumn, ivalue
kmr	<b>la_k_mr_set</b>	krow, kcolumn, ivalue
imc	<b>la_i_mc_set</b>	irow, icolumn, ivalue_r, ivalue_i
kmc	<b>la_k_mc_set</b>	krow, kcolumn, kvalue_r, kvalue_i
ivalue	<b>la_i_get_vr</b>	ivr, irow
kvalue	<b>la_k_get_vr</b>	ivr, krow
ivalue_r, ivalue_i	<b>la_i_get_vc</b>	ivc, irow
kvalue_r, kvalue_i	<b>la_k_get_vc</b>	ivc, krow
ivalue	<b>la_i_get_mr</b>	imr, irow, icolumn
kvalue	<b>la_k_get_mr</b>	imr, krow, kcolumn

ivalue_r, ivalue_i	la_i_get_mc	imc, irow, icolumn
kvalue_r, kvalue_i	la_k_get_mc	imc, krow, kcolumn

## Opérations sur un tableau

imr	la_i_transpose_mr	imr
imr	la_k_transpose_mr	imr
imc	la_i_transpose_mc	imc
imc	la_k_transpose_mc	imc
ivr	la_i_conjugate_vr	ivr
ivr	la_k_conjugate_vr	ivr
ivc	la_i_conjugate_vc	ivc
ivc	la_k_conjugate_vc	ivc
imr	la_i_conjugate_mr	imr
imr	la_k_conjugate_mr	imr
imc	la_i_conjugate_mc	imc
imc	la_k_conjugate_mc	imc

## Opérations scalaires

ir	la_i_norm1_vr	ivr
kr	la_k_norm1_vr	ivc
ir	la_i_norm1_vc	ivc
kr	la_k_norm1_vc	ivc
ir	la_i_norm1_mr	imr
kr	la_k_norm1_mr	imr
ir	la_i_norm1_mc	imc
kr	la_k_norm1_mc	imc
ir	la_i_norm_euclid_vr	ivr
kr	la_k_norm_euclid_vr	ivr
ir	la_i_norm_euclid_vc	ivc
kr	la_k_norm_euclid_vc	ivc
ir	la_i_norm_euclid_mr	mvr
kr	la_k_norm_euclid_mr	mvr
ir	la_i_norm_euclid_mc	mvc
kr	la_k_norm_euclid_mc	mvc
ir	la_i_distance_vr	ivr
kr	la_k_distance_vr	ivr
ir	la_i_distance_vc	ivc
kr	la_k_distance_vc	ivc

ir	la_i_norm_max	imr
kr	la_k_norm_max	imc
ir	la_i_norm_max	imr
kr	la_k_norm_max	imc
ir	la_i_norm_inf_vr	ivr
kr	la_k_norm_inf_vr	ivr
ir	la_i_norm_inf_vc	ivc
kr	la_k_norm_inf_vc	ivc
ir	la_i_norm_inf_mr	imr
kr	la_k_norm_inf_mr	imr
ir	la_i_norm_inf_mc	imc
kr	la_k_norm_inf_mc	imc
ir	la_i_trace_mr	imr
kr	la_k_trace_mr	imr
ir, ii	la_i_trace_mc	imc
kr, ki	la_k_trace_mc	imc
ir	la_i_lu_det	imr
kr	la_k_lu_det	imr
ir	la_i_lu_det	imc
kr	la_k_lu_det	imc

## Opérations sur les éléments entre tableaux

ivr	la_i_add_vr	ivr_a, ivr_b
ivc	la_k_add_vc	ivc_a, ivc_b
imr	la_i_add_mr	imr_a, imr_b
imc	la_k_add_mc	imc_a, imc_b
ivr	la_i_subtract_vr	ivr_a, ivr_b
ivc	la_k_subtract_vc	ivc_a, ivc_b
imr	la_i_subtract_mr	imr_a, imr_b
imc	la_k_subtract_mc	imc_a, imc_b
ivr	la_i_multiply_vr	ivr_a, ivr_b
ivc	la_k_multiply_vc	ivc_a, ivc_b
imr	la_i_multiply_mr	imr_a, imr_b
imc	la_k_multiply_mc	imc_a, imc_b
ivr	la_i_divide_vr	ivr_a, ivr_b
ivc	la_k_divide_vc	ivc_a, ivc_b
imr	la_i_divide_mr	imr_a, imr_b
imc	la_k_divide_mc	imc_a, imc_b

## Produits scalaires

ir	la_i_dot_vr	ivr_a, ivr_b
kr	la_k_dot_vr	ivr_a, ivr_b
ir, ii	la_i_dot_vc	ivc_a, ivc_b
kr, ki	la_k_dot_vc	ivc_a, ivc_b
imr	la_i_dot_mr	imr_a, imr_b
imr	la_k_dot_mr	imr_a, imr_b
imc	la_i_dot_mc	imc_a, imc_b
imc	la_k_dot_mc	imc_a, imc_b
ivr	la_i_dot_mr_vr	imr_a, ivr_b
ivr	la_k_dot_mr_vr	imr_a, ivr_b
ivc	la_i_dot_mc_vc	imc_a, ivc_b
ivc	la_k_dot_mc_vc	imc_a, ivc_b

## Inversion de matrice

imr, icondition	la_i_invert_mr	imr
imr, kcondition	la_k_invert_mr	imr
imc, icondition	la_i_invert_mc	imc
imc, kcondition	la_k_invert_mc	imc

## Décompositions et résolutions de matrice

ivr	la_i_upper_solve_mr	imr [, j_1_diagonal]
ivr	la_k_upper_solve_mr	imr [, j_1_diagonal]
ivc	la_i_upper_solve_mc	imc [, j_1_diagonal]
ivc	la_k_upper_solve_mc	imc [, j_1_diagonal]
ivr	la_i_lower_solve_mr	imr [, j_1_diagonal]
ivr	la_k_lower_solve_mr	imr [, j_1_diagonal]
ivc	la_i_lower_solve_mc	imc [, j_1_diagonal]
ivc	la_k_lower_solve_mc	imc [, j_1_diagonal]
imr, ivr_pivot, isize	la_i_lu_factor_mr	imr
imr, ivr_pivot, ksize	la_k_lu_factor_mr	imr
imc, ivr_pivot, isize	la_i_lu_factor_mc	imc
imc, ivr_pivot, ksize	la_k_lu_factor_mc	imc
ivr_x	la_i_lu_solve_mr	imr, ivr_b
ivr_x	la_k_lu_solve_mr	imr, ivr_b
ivc_x	la_i_lu_solve_mc	imc, ivc_b
ivc_x	la_k_lu_solve_mc	imc, ivc_b
imr_q, imr_r	la_i_qr_factor_mr	imr

<code>imr_q, imr_r</code>	<code>la_k_qr_factor_mr</code>	<code>imr</code>
<code>imc_q, imc_r</code>	<code>la_i_qr_factor_mc</code>	<code>imc</code>
<code>imc_q, imc_r</code>	<code>la_k_qr_factor_mc</code>	<code>imc</code>
<code>ivr_eig_vals</code>	<code>la_i_qr_eigen_mr</code>	<code>imr, i_tolerance</code>
<code>ivr_eig_vals</code>	<code>la_k_qr_eigen_mr</code>	<code>imr, k_tolerance</code>
<code>ivr_eig_vals</code>	<code>la_i_qr_eigen_mc</code>	<code>imc, i_tolerance</code>
<code>ivr_eig_vals</code>	<code>la_k_qr_eigen_mc</code>	<code>imc, k_tolerance</code>



### Avertissement

Une matrice doit être hermitienne si l'on veut calculer ses valeurs propres.

<code>ivr_eig_vals, imr_eig_vecs</code>	<code>la_i_qr_sym_eigen_mr</code>	<code>imr, i_tolerance</code>
<code>ivr_eig_vals, imr_eig_vecs</code>	<code>la_k_qr_sym_eigen_mr</code>	<code>imr, k_tolerance</code>
<code>ivc_eig_vals, imc_eig_vecs</code>	<code>la_i_qr_sym_eigen_mc</code>	<code>imc, i_tolerance</code>
<code>ivc_eig_vals, imc_eig_vecs</code>	<code>la_k_qr_sym_eigen_mc</code>	<code>imc, k_tolerance</code>

## Crédits

Michael Gogins

Nouveau dans la version 5.09 de Csound

# Opcodes de tableaux

Opcodes de tableaux

Liste des opcodes de tableaux (6.12) :

- *init* initialise un tableau
- *fillarray* remplit un tableau avec des valeurs
- *genarray* crée un tableau à partir d'une suite arithmétique
- *=* crée ou réinitialise un tableau à partir d'un autre tableau
- *slicearray* prend une partie d'un tableau
- *maparray* applique une fonction à un tableau
- *scalearray* pondère les valeurs d'un tableau
- *sorta* tri un tableau par ordre ascendant
- *sortd* tri un tableau par ordre descendant
- *limit* et *limit1* limite les valeurs d'un tableau
- *reshapearray* change les dimensions d'un tableau
- *trim* ajuste la taille d'un tableau unidimensionnel
- *copya2ftab* copie un tableau dans une table de fonction
- *copyf2array* copie une table de fonction dans un tableau
- *tab2array* copie une partie d'une table de fonction dans un tableau
- *dot* calcule le produit scalaire de deux tableaux
- *interleave* entrelace deux tableaux dans un seul
- *deinterleave* désentrelace un tableau dans deux tableaux
- *getrow* retourne une ligne d'un tableau bidimensionnel
- *getcol* retourne une colonne d'un tableau bidimensionnel
- *setrow* remplit une ligne d'un tableau bidimensionnel
- *setcol* remplit une colonne d'un tableau bidimensionnel
- *getrowlin* copie une ligne d'un tableau 2D et interpole entre les lignes
- *lenarray* retourne la longueur d'un tableau
- *minarray* retourne la valeur minimale d'un tableau

- *maxarray* retourne la valeur maximale d'un tableau
- *sumarray* retourne la somme des valeurs d'un tableau
- *cmp* compare deux tableaux
- *printarray* affiche un tableau
- les fonctions unaires *ceil*, *floor*, *round*, *int*, *frac*, *powoftwo*, *abs*, *log2*, *log10*, *log*, *exp*, *sqrt*, *cos*, *sin*, *tan*, *cosinv*, *sininv*, *taninv*, *sinh*, *cosh*, *tanh*, *cbrt*, *limit1* et les fonctions binaires *taninv2*, *pow*, *hypot*, *fmod*, *fmax*, *fmin* acceptent des tableaux en entrée.

Certaines instructions pour travailler avec des tableaux dans Csound (voir aussi le *chapître array dans le Csound FLOSS Manual*: [<http://write.flossmanuals.net/csound/e-arrays/>])

## Nom de variable

Un tableau doit être créé (via *init* ou *fillarray*) sous la forme *kNomTableau* **suivi** de crochets. Les crochets déterminent les dimensions du tableau. Ainsi,

```
kArr[] init 10
```

crée un tableau unidimensionnel (un vecteur) de longueur 10, tandis que

```
kArr[][] init 10, 10
```

crée un tableau à deux dimensions avec 10 lignes et 10 colonnes.

Après l'initialisation du tableau, on y fait référence comme un tout **sans** les crochets. Les crochets ne sont utilisés que pour indexer un élément :

```
kArr[]    init 10                ;with brackets because of initialization
kLen      = lenarray(kArr)       ;without brackets
kFirstEl  = kArr[0]              ;indexing with brackets
```

On utilise la même syntaxe pour une simple copie via l'opérateur '=' :

```
kArr1[] fillarray 1, 2, 3, 4, 5
kArr2[] = kArr1                  ;creates kArr2 as copy of kArr1
```

## k-rate

Noter que la plupart des opérations sur les tableaux ne se font actuellement qu'au taux-k. Ainsi, comme pour tout autre opcode de taux-k, une opération sur les tableaux sera automatiquement répétée à chaque cycle-k. Par exemple, le code suivant réécrit le tableau avec différentes valeurs aléatoires à chaque cycle-k, tant que l'instrument est actif :

```
kArr[] init 10
kIndx = 0
until kIndx == lenarray(kArr) do
  kArr[kIndx] rnd31 10, 0
  kIndx += 1
```

*od*

Pour éviter cela, on doit l'organiser de la manière habituelle, par exemple en utilisant un déclencheur :

```
kArr[] init 10
kTrig metro 1
if kTrig == 1 then ;do the following once a second
  kIndx = 0
  until kIndx == lenarray(kArr) do
    kArr[kIndx] rnd31 10, 0
    kIndx += 1
  od
endif
```

## Création/initialisation

La manière habituelle de créer un tableau est d'utiliser *init* :

```
kArr[] init 10 ;creates one-dimensional array with length 10
kArr[][] init 10, 10 ;creates two-dimensional array
```

On peut aussi créer un tableau unidimensionnel et le remplir de valeurs distinctes au moyen de l'opcode *fillarray*. La ligne suivante crée un vecteur de longueur 4 et le remplit avec les nombres [1, 2, 3, 4]:

```
kArr[] fillarray 1, 2, 3, 4
```

## Longueur

La fonction *lenarray*(kArr) retourne la longueur d'un tableau. Voir l'exemple pour la fonction *lenarray*.

## Copie d'un tableau vers ou d'une table

```
copyf2array kArr, kfn
```

copie les données d'une table de fonction dans un vecteur.

```
copya2ftab kArr, kfn
```

copie les données d'un vecteur dans une table de fonction.

Voir les exemples pour les opcodes *copyf2array* and *copya2ftab*.

## Opérations sur les tableaux : maths

### +, -, \*, / avec un nombre

Si les quatre opérateurs de base sont utilisés entre un tableau et un nombre scalaire, l'opération est appliquée à chaque élément. La manière la plus sûre de le faire est de stocker le résultat dans un nouveau tableau :

```
kArr1[] fillarray 1, 2, 3
```



```
kArr2[] = kArr1 + 10      ;(kArr2 is now [11, 12, 13])
```

Voici un exemple des opérations tableau/scalaire. Il utilise le fichier *array\_scalar\_math.csd* [examples/array\_scalar\_math.csd].

### Exemple 3. Exemple des opérations tableau/scalaire

```
<CsoundSynthesizer>
<CsOptions>
-n -m128
</CsOptions>
<CsInstruments>

  instr 1

;create array and fill with numbers 1..10
kArr1[] fillarray 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

;print content
    printf "%s", 1, "\nInitial content:\n"
kndx = 0
    until kndx == lenarray(kArr1) do
        printf "kArr[%d] = %f\n", kndx+1, kndx, kArr1[kndx]
kndx += 1
    od

;add 10
kArr2[] = kArr1 + 10

;print content
    printf "%s", 1, "\nAfter adding 10:\n"
kndx = 0
    until kndx == lenarray(kArr2) do
        printf "kArr[%d] = %f\n", kndx+1, kndx, kArr2[kndx]
kndx += 1
    od

;subtract 5
kArr3[] = kArr2 - 5

;print content
    printf "%s", 1, "\nAfter subtracting 5:\n"
kndx = 0
    until kndx == lenarray(kArr3) do
        printf "kArr[%d] = %f\n", kndx+1, kndx, kArr3[kndx]
kndx += 1
    od

;multiply by -1.5
kArr4[] = kArr3 * -1.5

;print content
    printf "%s", 1, "\nAfter multiplying by -1.5:\n"
kndx = 0
    until kndx == lenarray(kArr4) do
        printf "kArr[%d] = %f\n", kndx+1, kndx, kArr4[kndx]
kndx += 1
    od

;divide by -3/2
kArr5[] = kArr4 / -(3/2)

;print content
    printf "%s", 1, "\nAfter dividing by -3/2:\n"
```

```

kndx    =    0
until kndx == lenarray(kArr5) do
    printf "kArr[%d] = %f\n", kndx+1, kndx, kArr5[kndx]
    kndx += 1
od

;turnoff
    turnoff
endin

</CsInstruments>
<CsScore>
i 1 0 .1
</CsScore>
</CsoundSynthesizer>

```

## + , - , \* , / avec un autre tableau

Si les quatre opérateurs de base sont utilisés entre deux tableaux, l'opération est appliquée élément à élément. Le résultat peut être stocké directement dans un nouveau tableau :

```

kArr1[] fillarray 1, 2, 3
kArr2[] fillarray 10, 20, 30
kArr3[] = kArr1 + kArr2 ;(kArr3 is now [11, 22, 33])

```

Voici un exemple des opérations de tableau. Il utilise le fichier *array\_array\_math.csd* [examples/array\_array\_math.csd].

### Exemple 4. Exemple des opérations de tableau

```

<CsoundSynthesizer>
<CsOptions>
-n -m128
</CsOptions>
<CsInstruments>

instr 1

;create array and fill with numbers 1..10 resp .1..1
kArr1[] fillarray 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
kArr2[] fillarray 1, 2, 3, 5, 8, 13, 21, 34, 55, 89

;print contents
printf "%s", 1, "\nkArr1:\n"
kndx = 0
until kndx == lenarray(kArr1) do
    printf "kArr1[%d] = %f\n", kndx+1, kndx, kArr1[kndx]
    kndx += 1
od
printf "%s", 1, "\nkArr2:\n"
kndx = 0
until kndx == lenarray(kArr2) do
    printf "kArr2[%d] = %f\n", kndx+1, kndx, kArr2[kndx]
    kndx += 1
od

;add arrays
kArr3[] = kArr1 + kArr2

;print content
printf "%s", 1, "\nkArr1 + kArr2:\n"

```

```
kndx      =      0
  until kndx == lenarray(kArr3) do
    printf "kArr3[%d] = %f\n", kndx+1, kndx, kArr3[kndx]
  kndx    +=      1
  od

;subtract arrays
kArr4[] =      kArr1 - kArr2

;print content
  printf "%s", 1, "\nkArr1 - kArr2:\n"
kndx      =      0
  until kndx == lenarray(kArr4) do
    printf "kArr4[%d] = %f\n", kndx+1, kndx, kArr4[kndx]
  kndx    +=      1
  od

;multiply arrays
kArr5[] =      kArr1 * kArr2

;print content
  printf "%s", 1, "\nkArr1 * kArr2:\n"
kndx      =      0
  until kndx == lenarray(kArr5) do
    printf "kArr5[%d] = %f\n", kndx+1, kndx, kArr5[kndx]
  kndx    +=      1
  od

;divide arrays
kArr6[] =      kArr1 / kArr2

;print content
  printf "%s", 1, "\nkArr1 / kArr2:\n"
kndx      =      0
  until kndx == lenarray(kArr6) do
    printf "kArr6[%d] = %f\n", kndx+1, kndx, kArr6[kndx]
  kndx    +=      1
  od

;turnoff
  turnoff

  endin

</CsInstruments>
<CsScore>
i 1 0 .1
</CsScore>
</CsoundSynthesizer>
```

## Application d'une fonction à un tableau

```
kArrRes maparray kArrSrc, "fun"
```

applique la fonction de taux-k à un argument, définie dans la chaîne de caractères, à chaque élément du vecteur.

Les fonctions utilisables sont par exemple *abs*, *ceil*, *exp*, *floor*, *frac*, *int*, *log*, *log10*, *round*, *sqrt*. Voici un simple exemple :

```
kArrSrc[] fillarray 1, 2, 3, 4, 5
kArrRes[] init      5
kArrRes  maparray  kArrSrc, "sqrt"
```

Voir l'exemple pour l'opcode *maparray*.

## Opérations sur les tableaux : min, max, sum, scale, slice

### Minimum et maximum

```
kMin [ ,kMinIndx] minarray kArr
```

retourne la plus petite valeur d'un tableau et facultativement son indice.

```
kMax [ ,kMaxIndx] maxarray kArr
```

retourne la plus grande valeur d'un tableau et facultativement son indice. Voir les exemples pour les opcodes *minarray* and *maxarray*.

### Somme

```
kSum sumarray kArr
```

retourne la somme de toutes les valeurs dans *kArr*. Voir l'exemple pour l'opcode *sumarray*.

### Pondération

```
scalearray kArr, kMin, kMax
```

pondère toutes les valeurs dans *kArr* entre *kMin* et *kMax*.

```
kArr[] fillarray 1, 3, 9, 5, 6  
scalearray kArr, 1, 3
```

change *kArr* en [1, 1.5, 3, 2, 2.25]. Voir l'exemple pour l'opcode *scalearray*.

### Extrait

```
slicearray kArr, iStart, iEnd
```

retourne un extrait de *kArr* compris entre l'indice *iStart* et l'indice *iEnd* (inclus).

Il faut créer le tableau de destination de l'extrait par avance :

```
kArr[] fillarray 1, 2, 3, 4, 5, 6, 7, 8, 9  
kArr1[] init 5  
kArr2[] init 4  
kArr1 slicearray kArr, 0, 4 ;[1, 2, 3, 4, 5]  
kArr2 slicearray kArr, 5, 8 ;[6, 7, 8, 9]
```

Voir l'exemple pour l'opcode *slicearray*.

### Reformatage

utiliser *reshapearray* pour changer le format d'un tableau sans modifier sa capacité (changer 1D en 2D et vice-versa). Voir l'exemple de l'opcode *reshapearray*.

## Tableaux dans un UDO

La dimension d'un tableau en entrée doit être déclarée en deux endroits :

- comme `k[]` ou `k[][]` dans la liste des types en entrée
- comme `kNom[]`, `kNom[][]` etc dans la liste *xin*.

Par exemple :

```
opcode FirstEl, k, k[]
;returns the first element of vector kArr
kArr[] xin
      xout   kArr[0]
endop
```

Voici un exemple d'un tableau dans un UDO. Il utilise le fichier *array\_udo.csd* [examples/array\_udo.csd].

### Exemple 5. Exemple d'un tableau dans un UDO

```
<CsoundSynthesizer>
<CsOptions>
-nm128
</CsOptions>
<CsInstruments>

  opcode FirstEl, k, k[]
  ;returns the first element of vector kArr
kArr[] xin
xout kArr[0]
endop

  instr 1
kArr[] fillarray 6, 3, 9, 5, 1
kFirst FirstEl kArr
printf "kFirst = %d\n", 1, kFirst
turnoff
endin

</CsInstruments>
<CsScore>
i 1 0 .1
</CsScore>
</CsoundSynthesizer>
```

A noter que si un opcode (par exemple *inrg*) modifie les arguments de la liste d'arguments sur sa droite, on ne doit pas utiliser un élément de tableau indicé dans cette liste. Contrairement aux variables normales, les tableaux ne sont pas changés par les opcodes.

## Crédits

Cette page du manuel a été écrite par Joachim Heintz.  
Juillet 2013

Nouveau dans Csound 6.00

---

# Conversion des hauteurs

## Fonctions

Les opcodes qui effectuent les fonctions de hauteur communes sont :

- *cent*
- *cpsmidinn*
- *cpsoct*
- *cpspch*
- *octave*
- *octcps*
- *octmidinn*
- *octpch*
- *pchmidinn*
- *pchoct*
- *semitone*
- *ftom*
- *mtof*
- *mton*
- *ntom*
- *ntof*
- *pchtom*

## Opcodes de hauteurs

Les opcodes qui effectuent les fonctions d'accordage sont :

- *cps2pch*
- *cpsxpch*
- *cpstun*
- *cpstuni*

---

# Support MIDI en temps réel

Csound supporte les entrées et les sorties MIDI en temps réel, ainsi que les entrées depuis les fichiers MIDI. L'entrée MIDI en temps réel est activée au moyen de l'option de ligne de commande *-M* (ou *--midi-device=PERIPHERIQUE*). Pour une entrée sur un seul port, il faut spécifier le numéro ou le nom de périphérique après le *-M*. Pour une entrée sur plusieurs ports (actuellement seulement avec le module PortMidi), il faut utiliser 'a' ou 'm'. Par exemple, pour utiliser le périphérique numéro 2, on utilisera quelque chose comme :

```
csound -M2 monmiditr.csd
```

On peut trouver les périphériques disponibles en lançant Csound avec l'option *--midi-devices* :

```
csound --midi-devices
```

A partir de la version 6.14, le module PortMidi (voir ci-dessous pour une liste de tous les modules) permet d'associer des ports multiples à des canaux d'ordre plus élevé. En utilisant le nom de périphérique 'm', Csound prend ses entrées des périphériques MIDI existants dans le système et les associera à  $(N+1)*\text{canal}$ , où N est le numéro de périphérique dans la liste de PortMidi et canal est le canal d'entrée d'origine du périphérique. Alternativement, le nom de périphérique 'a' écoute sur toutes les entrées mais ne fait pas d'association avec des canaux d'ordre plus élevé.

La sortie MIDI en temps réel est activée au moyen de *-Q*, avec un numéro ou un nom de périphérique comme c'est montré ci-dessus.

Vous pouvez aussi charger un fichier MIDI en utilisant l'option de ligne de commande *-F* ou *--midifile=FI-CHIER*. Le fichier MIDI est lu en temps réel, et se comporte comme s'il était joué ou reçu en temps réel. Ainsi Csound ne sait pas si l'entrée MIDI vient d'un fichier MIDI ou directement d'une interface MIDI.

Une fois l'entrée et/ou la sortie MIDI activée(s), les opcodes comme *MIDI Input* et *MIDI Output* seront effectifs.

Quand l'entrée MIDI est activée (avec *-M* ou *-F*), chaque message de *noteon* entrant générera un événement de note pour un instrument qui a le même numéro que le canal de l'évènement (cela signifie que les instruments contrôlés par le MIDI sont polyphoniques par défaut, car chaque note générera une nouvelle instance de l'instrument). Si l'on a qu'un seul instrument, Csound travaille en mode omni, c'est-à-dire qu'il répond à tous les canaux au sein de cet instrument unique. Si l'on a plusieurs instruments numérotés de 1 à 16, alors par défaut instr 1 -> canal 1, instr 2 -> canal 2, à moins que l'on change la répartition (voir *massign* et *pgmassign* pour changer ce comportement). Pour un seul port en entrée, si l'on a plusieurs instruments, mais que instr N manque entre 1 et 16, canal N sera dirigé par défaut sur l'instrument ayant le numéro le plus bas.

Voir les opcodes pour l'*Interopérabilité MIDI/Partition* pour savoir comment concevoir des instruments utilisables depuis une partition ou pilotés par le MIDI.

Plusieurs modules MIDI en temps réel sont disponibles, et il faut utiliser l'option *-+rtmidi* (voir *-+rtmidi*), pour spécifier le module. Le module par défaut est portmidi qui fournit des E/S MIDI adéquates sur toutes les plates-formes, cependant, pour des performances améliorées et plus fiables, des modules spécifiques à certaines plates-formes sont également fournis.

Actuellement les modules midi disponibles sont :

- *portmidi* - Pour utiliser le système portmidi (sur toutes les plates-formes). C'est le réglage par défaut et il permet les entrées multi-ports (avec 'm' ou 'a' comme nom de périphérique).

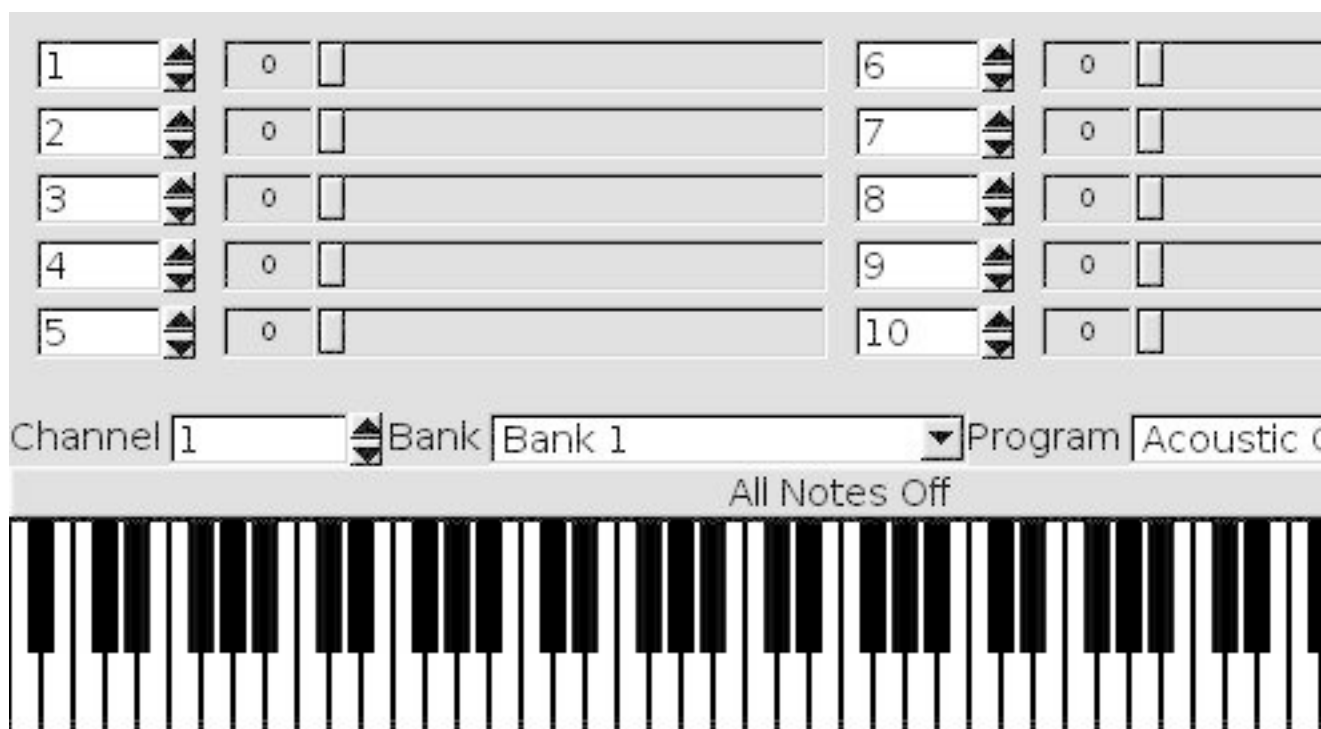
- *alsa* - Pour utiliser le système midi ALSA (seulement sur Linux).
- *jack* - Pour utiliser le système midi Jack.
- *winmme* - Pour utiliser le système windows MME (seulement sur Windows).
- *virtual* - Pour utiliser un clavier virtuel graphique (voir ci-dessous) comme entrée MIDI (sur toutes les plates-formes)



### Astuce

Lors de son exécution, Csound traite la partition puis se termine. S'il n'y a pas d'évènements dans la partition, Csound s'exécute indéfiniment. Si l'on désire n'utiliser que des évènements MIDI au lieu des évènements de partition et que l'on connaît la durée d'exécution désirée, il faut demander à Csound de s'exécuter pendant cette durée au moyen d'une *instruction f* comme "f 0 360".

## Clavier virtuel MIDI



Clavier virtuel MIDI.

Le module du clavier virtuel MIDI (activé par l'option `--rtmidi=virtual` sur la ligne de commande) fournit un moyen d'envoyer des informations MIDI en temps réel à Csound sans avoir besoin d'un périphérique MIDI. Il peut envoyer des informations de note, des changements de contrôle, des changements de banque et de programme sur un canal spécifié. L'information MIDI en provenance du clavier virtuel est traitée par Csound exactement de la même manière que si elle venait d'autres pilotes MIDI, si bien que si votre orchestre Csound est conçu pour travailler avec des périphériques matériels MIDI, cela marchera aussi.

Le clavier virtuel utilise l'option de périphérique (`-M`) pour récupérer le nom d'un fichier de mappage du clavier. Comme tous les pilotes MIDI, celui-ci nécessite un périphérique pour être activé. Si l'on désire seulement utiliser les réglages par défaut du clavier, il suffit de passer 0 (c'est-à-dire `-M0`). Si au lieu de



0 un nom de fichier est donné, le clavier essaiera de charger le fichier pour le mappage du clavier. Si le fichier n'a pas pu être ouvert ou lu correctement, les réglages par défaut seront utilisés.

Les fichiers de mappage du clavier permettent à l'utilisateur de personnaliser le nom et le numéro des banques ainsi que le nom et le numéro des programmes d'une banque. L'exemple suivant de mappage de clavier (nommé `keyboard.map`) a des commentaires intégrés sur le format de fichier. Ce fichier est aussi disponible dans la distribution des sources de Csound dans le répertoire `InOut/virtual_keyboard`.

```
# Carte de Personnalisation du clavier pour le clavier virtuel
# Steven Yi
#
# USAGE
#
# Lors de l'utilisation du clavier virtuel, vous pouvez fournir un nom de fichier
# pour un mappage des banques et des programmes via l'option -M, par exemple :
#
# csound -+rtmidi=virtual -Mkeyboard.map mon_projet.csd
#
# INFORMATION SUR LE FORMAT
#
# -les lignes commençant par '#' sont des commentaires
# -les lignes avec [] commencent les définitions d'une nouvelle banque,
# les contenus sont numBanque=nomBanque, avec numBanque=[1,16384]
# -les lignes suivant les instructions de banque sont des définitions de programme
# dans le format numProgramme=nomProgramme, avec numProgramme=[1,128]
# -les numéros de banque et de programme sont définis dans ce fichier
# en commençant à 1, mais ils sont convertis en valeurs midi (commençant
# à 0) lorsqu'ils sont lus
#
# NOTES
#
# -si une définition de banque invalide est trouvée, toutes les
# définitions de programme qui suivent seront ignorées jusqu'à ce
# qu'une nouvelle définition de banque valide soit trouvée
# -si une définition valide de banque sans programmes valides est
# trouvée, elle prendra par défaut les définitions de programme
# General MIDI
# -si une définition de programme invalide est trouvée, elle sera
# ignorée

[1=Ma Banque]
1=Mon Patch de Test 1
2=Mon Patch de Test 2
30=Mon Patch de Test 30

[2=Ma Banque2]
1=Mon Patch de Test 1(banque2)
2=Mon Patch de Test 2(banque2)
30=Mon Patch de Test 30(banque2)
```

Les dix réglettes du haut sont affectées par défaut aux contrôleurs MIDI numéro 1-10, mais on peut les changer à volonté. Les numéros de contrôleur et les valeurs de chaque réglette sont fixés par canal, si bien que l'on peut utiliser différents réglages et valeurs pour chaque canal.

Par défaut il y a 128 banques et pour chaque banque 128 patches réglés par défaut sur les noms General Midi. Le standard de banque MIDI utilise une résolution sur 14 bit pour supporter 16384 banques possibles, mais les numéros de banque par défaut sont 0-127. Pour utiliser des valeurs supérieures à 127, il faut utiliser un mappage de clavier personnalisé et fixer la valeur du numéro de banque désiré pour le nom de la banque. Le clavier virtuel transmettra correctement le numéro de banque comme MSB et LSB avec les contrôleurs 0 et 32.

Outre l'entrée disponible par l'interaction avec la GUI via la souris, on peut aussi déclencher les notes MIDI à partir du clavier ASCII quand la fenêtre du clavier virtuel a le focus. L'arrangement est organisé à la

manière d'un traceur et offre deux octaves et une tierce majeure, en partant du do médiant (note MIDI 60). La correspondance entre le clavier ASCII et les valeurs de note MIDI est donnée dans la table suivante.

**Tableau 5. Valeurs des notes MIDI du clavier ASCII**

Touche	Valeur MIDI
z	60
s	61
x	62
d	63
c	64
v	65
g	66
b	67
h	68
n	69
j	70
m	71
q	72
2	73
w	74
3	75
e	76
r	77
5	78
t	79
6	80
y	81
7	82
u	83
i	84
9	85
o	86
0	87
p	88

Voici un exemple de l'utilisation du clavier virtuel MIDI. Il utilise le fichier *virtual.csd* [exemples/virtual.csd].

```
<CsoundSynthesizer>
<CsOptions>; Select audio/midi flags here according to platform
; Audio out  Audio in    Virtual MIDI  -M0 is needed anyway
-odac        -iadc       --rtmidi=virtual -M0
</CsOptions>
```

```

<CsInstruments>
; By Mark Jamerson 2007

sr=44100
ksmps=10
nchnls=2

massign 1,1
prealloc 1,10

instr 1 ;Midi FM synth

inote cpsmidi
iveloc ampmidi 10000
idur = 2
    xtritim 1

kgate oscil 1,10,2
anoise noise 100*inote,.99
acps samphold anoise,kgate
aosc oscili 1000,acps,1
aout = aosc

; Use controller 7 to control volume
kvol ctrl7 1, 7, 0.2, 1

outs kvol * aout, kvol * aout

endin

</CsInstruments>

<CsScore>
f0 3600
f1 0 1024 10 1
f2 0 16 7 1 8 0 8
f3 0 1024 10 1 .5 .6 .3 .2 .5

e
</CsScore>
</CsoundSynthesizer>

```

## Entrée MIDI

Les opcodes suivants peuvent recevoir des informations MIDI :

- Information MIDI pour tous les instruments : *aftouch*, *chanctrl* et *polyaft*, *pchbend*.
- Information MIDI pour les instruments déclenchés par le MIDI : *veloc*, *midictrl*, *midichn* et *notnum*. Voir aussi *Convertisseurs*.
- Entrée de Contrôleur MIDI pour tous les instruments : *ctrl7*, *ctrl14* et *ctrl21*.
- Entrée de Contrôleur MIDI seulement pour les instruments déclenchés par le MIDI : *midic7*, *midic14* et *midic21*.
- Valeur d'initialisation de contrôleur MIDI : *initc7*, *initc14*, *initc21* et *ctrlinit*.
- Entrée MIDI générique : *midiiin*.

*massign* peut être utilisé pour spécifier l'instrument csound à déclencher par un canal MIDI particulier. *pgmassign* peut être utilisé pour assigner un instrument csound à un programme MIDI spécifique.

## Sortie de message MIDI

Les opcodes qui produisent des sorties MIDI sont :

- *mdelay*
- *nrpn*
- *outiat*
- *outic*
- *outic14*
- *outipat*
- *outipb*
- *outipc*
- *outkat*
- *outkc*
- *outkc14*
- *outkpat*
- *outkpb*
- *outkpc*
- *midiout*

## Entrée et sortie génériques

Les opcodes pour les entrées et les sorties MIDI génériques sont : *midiin* et *midiout*.

## Convertisseurs

Les opcodes suivants peuvent convertir de l'information MIDI provenant d'une instance d'un instrument déclenché par le MIDI :

- Convertisseurs de numéro de note MIDI en fréquence : *cpsmidi*, *cpsmidib*, *cpstmid*, *octmidi*, *octmidib*, *pchmidi* et *pchmidib*.
- Convertisseurs de hauteur/fréquence en numéro de note MIDI : *fom*, *pchtome*.
- Convertisseurs de vélocité MIDI en amplitude : *ampmidi* et *ampmidid*.

## Extension d'évènements

Les opcodes qui permettent d'étendre la durée d'un évènement sont :

- *release*
- *xtratim*

## Sortie de note-on/note-off

Les opcodes pour sortir des messages MIDI noteon ou noteoff sont :

- *midion*
- *midion2*
- *moscil*
- *noteoff*
- *noteon*
- *noteondur*
- *noteondur2*

## Opcodes pour l'interopérabilité MIDI/partition

Les opcodes suivants peuvent être utilisés pour concevoir des instruments qui fonctionnent de manière interchangeable avec du MIDI en temps réel et avec des événements de partition :

- *midichannelaftertouch*
- *midichn*
- *midicontrolchange*
- *mididefault*
- *midinoteoff*
- *midinoteoncps*
- *midinoteonkey*
- *midinoteonoct*
- *midinoteonpch*
- *midipitchbend*
- *midipolyaftertouch*
- *midiprogramchange*.



### **Adapter un instrument Csound déclenché par une partition.**

Pour adapter un instrument Csound ordinaire conçu pour être activé depuis une partition, à l'interopérabilité partition/MIDI :

- Changer tous les opcodes *linen*, *linseg*, et *expseg* respectivement en *linenr*, *linsegr*, et *expsegr*, sauf pour une enveloppe de décliquage ou d'atténuation. Cela ne changera en rien les exécutions pilotées par une partition.
- Ajouter les lignes suivantes au début de la définition de l'instrument :

```
; Pour être sûr qu'un instrument activé par le MIDI  
; aura un champ p3 positif.  
mididefault 60, p3  
; Met le numéro de touche MIDI traduit en cycles par  
; seconde dans p4, et la vélocité MIDI dans p5  
midinoteoncps p4, p5
```

Bien entendu, *midinoteoncps* pourrait être changé en *midinoteonoct* ou tout autre option, et le choix des p-champs est arbitraire.



## Options de ligne de commande d'Entrée/Sortie MIDI en temps réel

Les nouvelles *options d'E/S MIDI* dans Csound 5.02, peuvent remplacer la plupart des utilisations de ces opcodes d'interopérabilité, et en rendre l'usage plus facile.

# Messages système temps réel

Les opcodes pour les messages MIDI système temps réel sont : *mclock* et *mrtmsg*.

## Banques de réglottes

Les opcodes pour les bancs de réglottes de contrôleurs MIDI sont :

- *slider8*
- *slider8f*
- *slider16*
- *slider16f*
- *slider32*
- *slider32f*
- *slider64*
- *slider64f*
- *s16b14*
- *s32b14*
- *sliderKawai*

Les opcodes pour stocker des bancs de réglottes de contrôleurs MIDI dans des tables sont :

- *slider8table*

- *slider8tablef*
- *slider16table*
- *slider16tablef*
- *slider32table*
- *slider32tablef*
- *slider64table*
- *slider64tablef*

---

# Traitement spectral

voir la section *Synthèse/Resynthèse Additive* pour les opcodes élémentaires de resynthèse.

## Resynthèse par transformée de Fourier à court-terme (STFT)



### Utilisation des fichiers PVOC-EX avec les anciens opcodes pvoc de Csound

Tous les opcodes pvoc originaux peuvent lire maintenant des fichiers PVOC-EX, aussi bien que le format de fichier natif non portable. Comme un fichier PVOC-EX utilise une fenêtre d'analyse de taille double, les utilisateurs trouveront sans doute que le résultat est utilement amélioré, pour certains sons et certains traitements, malgré le fait que la resynthèse n'utilise pas la même taille de fenêtre.

En dehors du paramètre de taille de fenêtre, la différence principale entre le format original .pv et PVOC-EX est l'intervalle d'amplitude des trames d'analyse. Lorsque la pondération est appliquée, afin qu'il n'y ait pas de différences notables dans le niveau de sortie, quelque soit le format de fichier utilisé, de légères pertes d'amplitude peuvent encore se produire, car l'utilisation d'une fenêtre double modifie l'amplitude des trames, sans que le code de resynthèse en tienne compte. Noter que tous les opcodes pvoc originaux attendent un fichier d'analyse mono, et que les fichiers PVOC-EX multi-canaux seront ainsi réjetés.

Les opcodes qui implémentent la resynthèse STFT sont :

- *mincer*
- *temposcal*
- *tableseg*
- *pvadd*
- *pvbufread*
- *pvcross*
- *pvinterp*
- *pvoc*
- *pvread*
- *tableseg*
- *tablexseg*
- *vpvoc*

L'utilitaire *PVANAL* permet de générer les fichiers d'analyse pv.

## Resynthèse par codage prédictif linéaire (LPC)

Les opcodes de resynthèse par prédiction linéaire sont :



- *lpfreson*
- *lpinterp*
- *lpread*
- *lpreson*
- *lpslot*

On peut créer des fichiers d'analyse LPC au moyen de l'utilitaire *LPANAL*.

## Traitement spectral non-standard

Ces unités génèrent et traitent des types de données de signaux non-standard, tels que des signaux de contrôle du domaine temporel et des signaux audio sous-échantillonnés, et leur représentation dans le domaine fréquentiel (spectrale). Les types de données (*d-*, *w-*) se définissent par eux-mêmes et leur contenu n'est pas utilisable par les autres unités de Csound.

Les opcodes pour le traitement spectral non-standard sont *specaddm*, *specdiff*, *specdisp*, *specfilt*, *spechist*, *specptrk*, *specscal*, *specsum* et *spectrum*.

## Outils pour le traitement spectral en temps réel (opcodes pvs)

Avec ces opcodes, deux nouvelles facilités fondamentales sont ajoutées à Csound. Ils offrent une qualité audio améliorée, et une exécution rapide, permettant une analyse et une resynthèse de grande qualité (avec les transformations) à appliquer en temps réel aux signaux instantanés. Le vocodeur de phase original de Csound n'est pas changé ; les nouveaux opcodes utilisent un ensemble de fonctions complètement séparé basé sur « *pvoc.c* » dans la distribution CARL, écrite par Mark Dolson.

Les utilitaires de Csound *dnoise* et *srconv* (également par Dolson, de CARL) utilisent aussi ce moteur *pvoc*. *pvoc* de CARL est aussi la base pour le vocodeur de phase inclu dans le Composer's Desktop Project. Quelques petites modifications, mais importantes, ont été apportées au code CARL original pour supporter les flots de données en temps réel.

1. Support du nouveau format de fichier d'analyse PVOC-EX. C'est un format totalement portable et ouvert (multi plates-formes), supportant trois formats d'analyse, et les signaux multi-canaux. Actuellement seul le format standard amplitude+fréquence a été implémenté dans les opcodes, mais le format de fichier lui-même supporte les formats amplitude+phase et le format complexe (réel-imaginaire). En plus des nouveaux opcodes, les opcodes *pvoc* originaux de Csound ont été étendus (avec pour conséquence une qualité audio améliorée dans certains cas) pour lire les fichiers PVOC-EX aussi bien que le format original (non portable).

Les détails complets de la structure d'un fichier PVOC-EX sont disponibles sur le site web : <http://www.cs.bath.ac.uk/~jpff/NOS-DREAM/researchdev/pvocex/pvocex.html>. Ce site donne aussi les détails des programmes de console disponibles librement *pvocex* et *pvocex2* qui peuvent être utilisés pour créer des fichiers PVOC-EX dans tous les formats supportés.

2. Un nouveau type de signal du domaine fréquentiel, totalement transportable par flot de données, avec *f* comme premier caractère. Dans ce document on y fait référence par *fsig*. Le support principal des *fsigs* est fourni par les opcodes *pvsanal* et *pvsynth*, qui effectuent l'analyse et la resynthèse traditionnelles par

recouvrement-addition avec un vocodeur de phase, indépendamment du taux de contrôle de l'orchestre. La seule obligation est que le taux de contrôle *kr* soit supérieur ou égal au taux d'analyse, ce qui peut s'exprimer par  $ksmps \leq overlap$ , où *overlap* est la distance en échantillons entre deux trames d'analyse, comme spécifié pour *pvsanal*. Comme *overlap* vaut typiquement au moins 128, et le plus souvent 256, ce n'est pas une restriction coûteuse en pratique. L'opcode *pvsinfo* peut être utilisé au moment de l'initialisation pour acquérir les propriétés d'un fsig.

Le fsig permet la séparation nominale entre les étapes d'analyse et de resynthèse du vocodeur de phase pour une mise à disposition du programmeur Csound, ce qui permet non seulement d'employer des alternatives pour l'une ou les deux de ces étapes (pas seulement la resynthèse par banc d'oscillateur, mais aussi la génération synthétique de flots de données fsig), mais les opcodes opérant sur le flot fsig peuvent être eux-mêmes plus élémentaires. Ainsi le fsig permet la création d'un véritable environnement de greffon de flots de données pour les signaux du domaine fréquentiel. Avec les vieux opcodes pvoc, chaque opcode doit pouvoir agir comme un resynthétiseur, si bien que des facilités comme la transposition de hauteur sont dupliquées dans chaque opcode ; et dans la plupart des cas les opcodes ont beaucoup de paramètres. La séparation des étapes d'analyse et de synthèse au moyen du fsig encourage le développement d'une grande variété d'opcodes qui sont des briques élémentaires implémentant une ou deux fonctions, et avec lesquelles on peut construire des processus plus élaborés.

Cette réalisation en est encore à ses débuts et présente un caractère expérimental, et il est possible que la définition précise des opcodes change en réponse aux avis des utilisateurs. De plus, de nombreuses nouvelles possibilités d'opcode sont ouvertes ; ces facteurs peuvent aussi avoir une influence rétrospective sur les opcodes présentés ici.

Noter que certains paramètres d'opcode ont actuellement une implémentation restreinte ou manquante. Ceci, au moins en partie, afin de préserver la simplicité des opcodes à ce niveau, et aussi parce qu'ils concernent d'importantes questions de conception pour lesquelles aucune décision n'a encore été prise, et pour lesquelles l'opinion des utilisateurs est souhaitée.

Un point important au sujet de ce nouveau type de signal est que, parce que le taux d'analyse est typiquement très inférieur à *kr*, les nouvelles trames d'analyse ne sont pas disponibles à chaque k-cycle. En interne, les opcodes tracent *ksmps*, et maintiennent également un compteur de trames, afin que les trames soient lues et écrites aux bons moments ; ce processus est généralement transparent pour l'utilisateur. Cependant, cela signifie que les signaux de taux-k n'agissent sur un fsig qu'au taux d'analyse, pas à chaque k-cycle. L'opcode *pvsftw* retourne un drapeau au taux-k qui est positionné lorsque de nouvelles données fsig sont disponibles.

A cause de la nature du système de recouvrement-addition, l'utilisation des ces opcodes infère un délai, ou latence, petit mais significatif déterminé par la taille de la fenêtre ( $\max(\text{ifftsize}, \text{iwinsize})$ ). Il vaut typiquement 23ms. Dans cette première réalisation, le délai dépasse légèrement le minimum théorique, et l'on espère qu'il pourra être réduit, lorsque les opcodes seront optimisés pour le transport par flot de données en temps réel.

Les opcodes pour le traitement spectral en temps réel sont *pvsadsyn*, *pvsanal*, *pvsocross*, *pvsfread*, *pvsftr*, *pvsftw*, *pvsinfo*, *pvsmaska* et *pvsynth*.

De plus il y a un certain nombre d'opcodes disponibles sous forme de greffons dans Csound 5, et dans le noyau de Csound6. Ce sont *pvsstanal*, *pvsdiskin*, *pvscent*, *pvsdemix*, *pvsfreeze*, *pvsbuffer*, *pvsbufread*, *pvsbufread2*, *pvscale*, *pvsshift*, *pvsifd*, *pvsinit*, *pvsin*, *pvsout*, *pvsosc*, *pvsbin*, *pvsdisp*, *pvsfwrite*, *pvslock*, *pvmix*, *pvssmooth*, *pvsfilter*, *pvsblur*, *pvsstencil*, *pvsarp*, *pvsvoc*, *pvsrmorph*, *pvsbandp*, *pvsbandr*, *pvsrarp*, *pvs gain*, *pvs2tab*, *tab2pvs*.

Un certain nombre d'opcodes sont conçus pour générer et traiter des flots de données de pistes de partiels. Ce sont *partials*, *trcross*, *trfilter*, *trsplit*, *trmix*, *trscale*, *trshift*, *trlowest*, *trhighest*, *tradsyn*, *sinsyn*, *resyn*, *binit*

Voir la section *Piles* pour une information sur les opcodes qui peuvent empiler les signaux de type f.

## Traitement spectral avec ATS

Ces opcodes peuvent lire, transformer et resynthétiser des fichiers d'analyse ATS. Prière de noter que l'application ATS est nécessaire pour produire les fichiers d'analyse. Voici un extrait du Manuel de Référence d'ATS.

« *ATS est une bibliothèque de fonctions pour l'Analyse spectrale, la Transformation et la Synthèse du son basée sur un modèle sinusoïdal plus du bruit de bande critique. Un son dans ATS est un objet symbolique représentant un modèle spectral qui peut être sculpté au moyen de diverses fonctions de transformation.* »

Pour plus d'information sur ATS visiter : <http://www-ccrma.stanford.edu/~juan/ATS.html>.

Les fichiers d'analyse ATS peuvent être produits avec le logiciel ATS ou l'utilitaire csound ATSA.

Les opcodes pour le traitement ATS sont :

- *ATSinfo* : lit les données de l'en-tête d'un fichier ATS.
- *ATSread*, *ATSreadnz*, *ATSbufread*, *ATSinterpread*, *ATSpartialtap* : lisent les données d'un fichier ou d'un tampon ATS.
- *ATSadd*, *ATSaddnz*, *ATScross*, *ATSinnoi* : Resynthétisent le son.

## Crédits

Auteur : Alex Norman  
Seattle, Washington  
2004

## Opcodes Loris



### Note

Ces opcodes sont un composant facultatif de Csound5. Pour savoir s'ils sont installés utilisez la commande 'csound -z' qui donne la liste des opcodes disponibles.

La famille des opcodes Loris encapsule : *lorisread* qui importe un ensemble de partiels à largeur de bande adaptée depuis un fichier de données au format SDIF, en appliquant, au taux de contrôle, des enveloppes de pondération de fréquence, d'amplitude et de largeur de bande, et qui stocke les partiels modifiés en mémoire ; *lorismorph*, qui opère une transformation (morphing) entre deux ensembles stockés de partiels à largeur de bande adaptée et stocke un nouvel ensemble de partiels représentant le son transformé. La transformation est réalisée en interpolant linéairement les enveloppes des paramètres (fréquence, amplitude, et largeur de bande, ou aspect bruiteux) des partiels à largeur de bande adaptée selon des fonctions de transformation de la fréquence, de l'amplitude et de la largeur de bande, variant au taux de contrôle ; *lorisplay*, qui restitue un ensemble de partiels à largeur de bande adaptée en utilisant la méthode de Synthèse Additive à Largeur de Bande Adaptée implémentée dans le logiciel Loris, avec application d'enveloppes de pondération de fréquence, d'amplitude, et de largeur de bande, variant au taux de contrôle.

Pour plus d'information sur la transformation du son et sa manipulation avec Loris et le Modèle Additif à Largeur de Bande Adaptée Réassignée, visiter le site web de Loris à [www.cerlsoundgroup.org/Loris](http://www.cerlsoundgroup.org/Loris) [<http://www.cerlsoundgroup.org/Loris>].

## Exemples

### Exemple 6. Jouer les partiels sans modification

```

;
; Joue les partiels dans clarinet.sdif
; de 0 à 3 sec avec un temps de transition de 1 ms
; et sans modification de fréquence, d'amplitude,
; ou de largeur de bande.
;
instr 1
  ktime    linseg      0, p3, 3.0    ; fonction linéaire du temps de 0 à 3 secondes
          lorisread    ktime, "clarinet.sdif", 1, 1, 1, 1, .001
  asig     lorisplay    1, 1, 1, 1
          out          asig
endin

```

### Exemple 7. Ajouter une intonation et un vibrato

```

; Joue les partiels dans clarinet.sdif
; de 0 à 3 sec avec un temps de transition de 1 ms
; ajout d'une intonation et d'un vibrato, accroissement
; du "souffle" (aspect bruiteux) et de l'amplitude
; générale, et ajout d'un filtre passe-haut.
;
instr 2
  ktime    linseg      0, p3, 3.0    ; fonction linéaire du temps de 0 à 3 secondes

  ; calcule le rapport de fréquence pour l'intonation
  ; (la hauteur originale était sol#3)
  ifscale  =           cpspch(p4)/cpspch(8.08)

  ; faire une enveloppe de vibrato
  kvenv     linseg      0, p3/6, 0, p3/6, .02, p3/3, .02, p3/6, 0, p3/6, 0
  kvib      oscil       kvenv, 4, 1  ; table 1, sinusoid

  kbwenv     linseg      1, p3/6, 1, p3/6, 2, 2*p3/3, 2
          lorisread    ktime, "clarinet.sdif", 1, 1, 1, 1, .001
  a1         lorisplay    1, ifscale+kvib, 2, kbwenv
  a2         atone       a1, 1000    ; filtre passe-haut, coupure à 1000 Hz
          out          a2
endin

```

L'instrument du premier exemple synthétise un son de clarinette en utilisant du début à la fin les partiels dérivés de l'analyse à bande adaptée réassignée d'un son de clarinette de trois secondes, stockés dans le fichier `clarinet.sdif`. L'instrument de l'exemple 2 ajoute une intonation et un vibrato au son de clarinette synthétisé par l'instrument 1, renforce son amplitude et son aspect bruiteux, et applique un filtre passe-haut au résultat. La partition suivante peut être utilisée pour tester les deux instruments décrits ci-dessus.

```

; créer une sinus dans la table 1
f 1 0 4096 10 1

; jouer instr 1
;   début  dur
i 1      0   3
i 1      +   1
i 1      +   6
s

; jouer instr 2
;   début  dur  hauteur

```

```
i 2      1      3      8.08
i 2      3.5    1      8.04
i 2      4      6      8.00
i 2      4      6      8.07

e
```

## Exemple 8. Transformation de partiels

```
; Transforme les partiels de clarinet.sdif vers
; les partiels de flute.sdif sur la durée de la
; partie tenue des deux notes (de 0,2 à 2,0 secondes
; pour la clarinette, et de 0,5 à 2,1 secondes
; pour la flûte). Les portions d'attaque et de
; chute dans le son transformé sont spécifiées
; par les paramètres p4 et p5, respectivement.
; Le temps de transformation est le temps entre
; l'attaque et la chute. Les partiels de la
; clarinette sont transposés pour s'accorder à
; la hauteur de la note de la flûte (ré au-dessus
; du do médium).
;
instr 1
  ionset  =          p4
  idecay  =          p5
  itmorph =          p3 - (ionset + idecay)
  ipshift =          cpspch(8.02)/cpspch(8.08)

  ; fonction temporelle de la clarinette, transformation de 0,2 à 2,0 secondes
  ktcl    linseg      0, ionset, .2, itmorph, 2.0, idecay, 2.1
  ; fonction temporelle de la flûte, transformation de 0,5 à 2,1 secondes
  ktfl    linseg      0, ionset, .5, itmorph, 2.1, idecay, 2.3
  kmurph   linseg      0, ionset, 0, itmorph, 1, idecay, 1
          lorisread    ktcl, "clarinet.sdif", 1, ipshift, 2, 1, .001
          lorisread    ktfl, "flute.sdif", 2, 1, 1, 1, .001
          lorismorph    1, 2, 3, kmurph, kmurph, kmurph
  asig     lorisplay   3, 1, 1, 1
          out          asig
endin
```

## Exemple 9. Plus de transformation

```
; Transforme les partiels de trombone.sdif vers les
; partiels de meow.sdif. Les dates de début et de fin
; de la transformation sont spécifiées par les
; paramètres p4 et p5, respectivement. La transformation
; a lieu sur la deuxième des quatre notes dans chaque
; son, de 0,75 à 1,2 secondes pour le trombone flatterzung,
; et de 1,7 à 2,2 secondes pour le miaulement de chat.
; Des fonctions de transformation différentes sont
; utilisées pour les enveloppes de fréquence et
; d'amplitude, afin que l'amplitude des partiels
; ait une transition plus rapide du trombone au
; chat que les fréquences. (Les enveloppes de largeur
; de bande utilisent la même fonction de transformation
; que les amplitudes).
;
instr 2
  ionset  =          p4
  imorph  =          p5 - p4
  irelease =          p3 - p5

  ktbn    linseg      0, ionset, .75, imorph, 1.2, irelease, 2.4
  ktmeow  linseg      0, ionset, 1.7, imorph, 2.2, irelease, 3.4
```

```

kmfreq  linseg      0, ionset, 0, .75*imorph, .25, .25*imorph, 1, irelease, 1
kmamp    linseg      0, ionset, 0, .75*imorph, .9, .25*imorph, 1, irelease, 1

        lorisread    ktbn, "trombone.sdif", 1, 1, 1, 1, .001
        lorisread    ktmeow, "meow.sdif", 2, 1, 1, 1, .001
        lorismorph    1, 2, 3, kmfreq, kmamp, kmamp
    asig  lorisplay    3, 1, 1, 1
        out           asig
endin

```

L'instrument dans le premier exemple effectue une transformation du son entre une note de clarinette et une note de flûte en utilisant les partiels à bande adaptée réassignée stockés dans `clarinet.sdif` et dans `flute.sdif`.

La transformation est effectuée sur les portions tenues des notes, 0,2 à 2,0 secondes dans le cas de la note de clarinette et 0,5 à 2,1 secondes dans le cas de la note de flûte. Les fonctions d'index temporel, `ktcl` et `ktfl`, alignent les portions d'attaque et de chute des notes avec les temps d'attaque et de chute du son transformé, spécifiées respectivement par les paramètres `p4` et `p5`. L'attaque du son transformé est entièrement composée des données de partiel de la clarinette, et la chute est entièrement composée de données de la flûte. Les partiels de la clarinette sont transposés pour s'accorder à la hauteur de la note de flûte (ré au-dessus du do médium).

L'instrument dans le second exemple effectue une transformation du son entre une note de trombone *flutterzung* et un miaulement de chat en utilisant les partiels à bande adaptée réassignée stockés dans `trombone.sdif` et `meow.sdif`. Les données dans ces fichiers SDIF ont été réparties par canaux et séparées pour établir une correspondance entre partiels.

Les deux ensembles de partiels sont importés et stockés dans des positions mémoire étiquetées 1 et 2, respectivement. Les deux sons originaux ont quatre notes, et la transformation est effectuée sur la seconde note de chaque son (de 0,75 à 1,2 secondes pour le trombone *flutterzung*, et de 1,7 à 2,2 secondes pour le miaulement de chat). Les fonctions d'index temporel, `ktbn` et `ktmeow`, alignent ces segments des ensembles de partiels source et cible avec les paramètres spécifiés pour la durée du début, de la fin, et totale de la transformation. Deux fonctions de transformation différentes sont utilisées, afin que les amplitudes des partiels et les coefficients de largeur de bande se transforment rapidement des valeurs du trombone aux valeurs du miaulement de chat, tandis que les fréquences opèrent une transition plus graduelle. Les partiels transformés sont stockés dans la position mémoire étiquetée 3 et restitués par l'instruction `lorisplay` qui suit. Ils auraient pu aussi être utilisés comme source pour une autre transformation dans un instrument de transformation à trois étapes. La partition suivante peut être utilisée pour tester les deux instruments décrits ci-dessus.

```

; jouer instr 1
;   début   dur   attaque   chute
i 1    0     3     .25      .15
i 1    +     1     .10      .10
i 1    +     6     1.       1.
s

; jouer instr 2
;   début   dur   début_morph   fin_morph
i 2    0     4     .75          2.75
e

```

## Crédits

Cette implémentation des générateurs unitaires Loris a été écrite par Kelly Fitz ([loris@cerlsoundgroup.org](mailto:loris@cerlsoundgroup.org) [<mailto:loris@cerlsoundgroup.org>]).

Elle est construite d'après une implémentation prototype du générateur unitaire *lorisplay* écrite par Corbin Champion, et basée sur la méthode de Synthèse Additive à Largeur de Bande Adaptée et sur les algorithmes de transformation du son implémentés dans la bibliothèque Loris pour la modélisation et la manipulation du son. Les opcodes ont été ensuite adaptés en greffon pour Csound 5 par Michael Gogins.

## Opcodes spectraux basés sur des tableaux



### Note

Ces opcodes sont conçus pour travailler avec des tableaux de taux-k pour la manipulation de données spectrales.

- *fft*,
- *fftinv*,
- *rfft*,
- *rfft*,
- *pvs2array*,
- *pvsfromarray*,
- *cmplxprod*,
- *rect2pol*,
- *pol2rect*,
- *window*,
- *r2c*,
- *c2r*,
- *mags* et
- *phs*.

---

# Chaînes de caractères

Les variables chaîne de caractères sont des variables dont le nom commence par S ou par gS (pour les variables chaîne locales ou globales, respectivement). On peut utiliser ces variables comme argument d'entrée de n'importe quel opcode qui attend une chaîne constante entre apostrophes, et on peut les manipuler durant les périodes d'initialisation ou d'exécution avec les opcodes dont la liste suit.

Il est également possible d'utiliser des chaînes dans les p-champs. Un p-champ chaîne peut être utilisé directement par plusieurs opcodes de l'orchestre, ou il peut être d'abord copié dans une variable chaîne :

```
a1    diskin2 p5, 1
```

```
Snom  strget p5  
a1    diskin2 Snom, 1
```

Les chaînes dans Csound peuvent être exprimées par les doubles apostrophes traditionnelles (" "), mais aussi par {{ }}. La seconde méthode est utile si l'on veut utiliser les caractères ';' et '\$' dans la chaîne sans avoir recours aux codes ASCII.



## Note

Les variables chaînes et les opcodes correspondants ne sont pas disponibles dans les versions de Csound antérieures à la 5.00.

On peut également lier une chaîne à un numéro au moyen de *strset* et *strget*.

Csound 5 a aussi amélioré l'analyse des constantes chaîne. Il est possible de spécifier une chaîne multi-lignes en l'entourant avec {{ et }} à la place des habituelles doubles apostrophes, et les séquences d'échappement suivantes sont automatiquement converties :

- \a : cloche d'alerte
- \b : retour arrière
- \n : nouvelle ligne
- \r : retour chariot
- \t : tabulation
- \\ : le caractère '\'
- \nnn : le caractère ayant le code ASCII (en octal) nnn



## Note

Si l'on veut éviter qu'une séquence d'échappement soit automatiquement convertie, il faut l'échapper avec un caractère '\' supplémentaire afin que Csound sache qu'il ne doit pas l'interpréter. Par exemple la chaîne de caractères "Retour à la ligne\nnon échappé" sera convertie en

```
"Retour à la ligne  
non échappé"
```



avant d'être utilisée, tandis que le chaîne de caractères "Retour à la ligne\\néchappé" sera convertie en

"Retour à la ligne\néchappé"

avant d'être utilisée.

Les chaînes peuvent être utilement employées avec l'opcode *system* :

```
instr 1
; csound5 permet de placer une chaîne sur plusieurs lignes dans des accolades doubles
system {{      ps
              date
              cd ~/Desktop
              pwd
              ls -l
              whois csounds.com
            }}
endin
```

Et avec les *opcodes python*, entre autres :

```
pyruni {{
import random

pool = [(1 + i/10.0) ** 1.2 for i in range(100)]

def get_number_from_pool(n, p):
    if random.random() < p:
        i = int(random.random() * len(pool))
        pool[i] = n
    return random.choice(pool)
}}
```

## Opcodes de manipulation de chaîne

Ces opcodes effectuent des opérations sur les variables chaîne (note : la plupart des opcodes ne sont exécutés qu'au moment de l'initialisation, et ils ont une version avec un suffixe "k" qui s'exécute au taux-i et au taux-k ; les exceptions à cette règle comprennent *puts* et *strget*) :

- *strcpy* et *strcpyk* - Assignment à une variable chaîne.
- *strcat* et *strcatk* - Concaténation de chaînes, avec mémorisation du résultat dans une variable.
- *strcmp* et *strcmpk* - Comparaison de chaînes.
- *strget* - Assignment à une variable chaîne de la valeur trouvée dans la table *strset* à l'index spécifié, ou d'un p-champ chaîne de la partition.
- *strlen* et *strlenk* - Retourne la longueur d'une chaîne.
- *sprintf* - conversion de sortie formatée à la manière de *printf*, avec mémorisation du résultat dans une variable chaîne.
- *sprintfk* - conversion de sortie formatée à la manière de *printf*, avec mémorisation du résultat dans une variable chaîne au taux-k.

- *puts* - Impression d'une constante ou d'une variable chaîne.
- *strindex* et *strindexk* - Retourne la première occurrence d'une chaîne dans une autre chaîne.
- *strrindex* et *strrindexk* - Retourne la dernière occurrence d'une chaîne dans une autre chaîne.
- *strsub* et *strsubk* - Retourne une sous-chaîne de la chaîne passée en paramètre.

## Opcodes de conversion de chaîne

Ces opcodes convertissent des variables chaînes (note : la plupart des opcodes ne sont exécutés qu'au moment de l'initialisation, et ils ont une version avec un suffixe "k" qui s'exécute au taux-i et au taux-k ; les exceptions à cette règle comprennent *puts* et *strget*) :

- *strtod* et *strtodk* - Convertit une valeur de chaîne en une valeur en virgule flottante.
- *strtol* et *strtolk* - Convertit une valeur de chaîne en un entier signé.
- *strchar* et *strchark* - Retourne le code ASCII d'un caractère dans une chaîne.
- *strlower* et *strlowerk* - Convertit une chaîne en minuscules.
- *strupper* et *strupperk* - Convertit une chaîne en majuscules.

---

# Opcodes vectoriels

La famille des opcodes vectoriels est conçue pour pouvoir traiter des sections de ftable comme des vecteurs pour diverses opérations sur celles-ci.

## Opérateurs de tableaux de vecteurs

Les opcodes vectoriels suivants supportent les accès en lecture/écriture sur des tableaux de vecteurs (tableaux de tableaux) :

- *vtablei*
- *vtablelk*
- *vtablek*
- *vtablea*
- *vtablewi*
- *vtablewk*
- *vtablewa*
- *vtabi*
- *vtabk*
- *vtaba*
- *vtabwi*
- *vtabwk*
- *vtabwa*

## Opérations entre un signal vectoriel et un signal scalaire

Ces opcodes effectuent des opérations numériques entre un signal de contrôle vectoriel (contenu dans une ftable), et un signal scalaire. Le résultat est un nouveau vecteur qui remplace les anciennes valeurs de la table. Il y a des versions de ces opcodes de taux-k et de taux-i.

Tous ces opérateurs sont conçus pour être utilisés de concert avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

Opérations Entre un Signal Vectoriel et un Signal Scalaire :

- *vadd*
- *vmult*
- *vpow*

- *vexp*
- *vadd\_i*
- *vmult\_i*
- *vpow\_i*
- *vexp\_i*

## Opérations entre deux signaux vectoriels

Ces opcodes effectuent des opérations entre deux vecteurs, de telle manière que chaque élément du premier vecteur est traité avec l'élément correspondant de l'autre vecteur. Le résultat est un nouveau vecteur qui remplace les anciennes valeurs du vecteur source.

Opérations entre deux signaux vectoriels :

- *vaddv*
- *vsbv*
- *vmultv*
- *vdivv*
- *vpowv*
- *vexpv*
- *vcopy*
- *vmap*
- *vaddv\_i*
- *vsbv\_i*
- *vmultv\_i*
- *vdivv\_i*
- *vpowv\_i*
- *vexpv\_i*
- *vcopy\_i*

Ces opérateurs sont conçus pour être utilisés de concert avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Générateurs vectoriels d'enveloppe

Les opcodes pour générer des vecteurs contenant des enveloppes sont *vlinseg* et *vexpseg*.

Ces opérateurs sont semblables à *linseg* et *expseg*, mais ils opèrent avec des signaux vectoriels à la place des signaux scalaires.

La sortie est un vecteur dans une ftable (préalablement allouée), tandis que chaque point charnière de l'enveloppe est en fait un vecteur de valeurs. Tous les points charnière doivent contenir le même nombre d'éléments (*ielements*).

Ces opérateurs sont conçus pour être utilisés de concert avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Limitation et enroulement des signaux vectoriels de contrôle

Les opcodes pour effectuer la limitation et l'enroulement des éléments dans un vecteur sont :

- *vlimit*
- *vwrap*
- *vmirror*

Ces opérateurs sont semblables à *limit*, *wrap* et *mirror*, mais ils opèrent sur un signal vectoriel à la place d'un signal scalaire. Les résultats remplacent les anciennes valeurs du vecteur contenues dans une ftable si celles-ci sont en dehors de l'intervalle min/max. Si l'on veut conserver le vecteur d'entrée, il faut utiliser l'opcode *vcopy* pour le copier dans une autre table.

Tous ces opcodes travaillent au taux-k.

Tous ces opérateurs sont conçus pour être utilisés de concert avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Chemins de retard vectoriel au taux de contrôle

Chemins de Retard Vectoriel au Taux de Contrôle :

- *vdelayk*
- *vport*
- *vecdelay*

## Générateurs de signal aléatoire vectoriel

Ces opcodes génèrent des vecteurs de nombres aléatoires à stocker dans des tables. Ils génèrent une sorte de 'bruit vectoriel à bande limitée'. Tous ces opcodes fonctionnent au taux-k.

Générateurs de signal aléatoire vectoriel : *vrandh* et *vrandi*.

Des vecteurs d'automates cellulaires peuvent être générés au moyen de : *vcella*.

---

# Système de patch zak

Les opcodes zak sont utilisés pour créer un système de patch aux taux-i, -k et -a. On peut se représenter le système zak comme un tableau global de variables. Ces opcodes sont utiles pour réaliser de manière flexible des branchements et des routages d'un instrument à l'autre. Le système est semblable à une matrice de branchement sur une console de mixage ou à une matrice de modulation sur un synthétiseur. Il est aussi utile lorsque l'on a besoin d'un tableau de variables.

Le système zak est initialisé par l'opcode *zakinit* qui est habituellement placé juste après les autres initialisations globales : *sr*, *kr*, *ksmps*, *nchnls*. L'opcode *zakinit* définit deux plages de mémoire, une pour les patchs aux taux-i et -k, et l'autre pour les patchs au taux-a. L'opcode *zakinit* ne peut être appelé qu'une fois. Après l'initialisation de l'espace zak, on peut utiliser d'autres opcodes zak pour lire et écrire dans l'espace mémoire zak, ainsi qu'exécuter d'autres tâches.

Les opcodes zak sont comptés à partir de 0, si bien que si l'on définit un canal, le seul canal valide est le canal 0.

Les opcodes pour le système de patch zak sont :

- Taux Audio : *zacl*, *zakinit*, *zamod*, *zar*, *zarg*, *zaw* et *zawm*.
- Taux de Contrôle : *zkcl*, *zkmod*, *zkr*, *zkw*, et *zkwm*.
- A l'initialisation : *zir*, *ziw* et *ziwm*

---

# Accueil de greffon

Csound accueille actuellement des greffons externes au moyen de *dssi4cs* (pour les greffons LADSPA) sur Linux.

## DSSI et LADSPA pour Csound

*dssi4cs* permet l'utilisation des effets et des synthétiseurs greffon DSSI et LADSPA dans Csound sur Linux. Les opcodes suivants sont disponibles :

- *dssiinit* - Charge un greffon.
- *dssiactivate* - Active ou désactive un greffon si celui-ci le permet.
- *dssilist* - Liste tous les greffons disponibles trouvés dans les variables globales LADSPA\_PATH et DSSI\_PATH.
- *dssiaudio* - Traitement audio au moyen d'un greffon.
- *dssictls* - Envoie une information de contrôle sur le port de contrôle d'un greffon.

Voir l'entrée pour *dssiinit* pour un exemple d'utilisation.



### Note

Actuellement seuls les greffons LADSPA sont supportés, mais le support de DSSI est programmé.

---

# OSC et réseau

## Opcodes Ableton Link

Le but d'Ableton Link est de synchroniser le temps, la pulsation et la phase entre des applications musicales s'exécutant en temps réel depuis différents programmes, processus et adresses de réseau. C'est utile, par exemple pour des orchestres d'ordinateurs portables.

Ableton Live n'est pas nécessaire pour utiliser le protocole Ableton Link, car c'est un protocole de pair à pair. Il y a une session Link sur le réseau local qui maintient un temps, un tempo et une pulsation globaux. Chaque pair peut fixer le tempo et ensuite, tous les pairs dans la session partagent ce tempo. Un processus peut avoir n'importe quel nombre de pairs (c'est-à-dire n'importe quel nombre d'objets Link). Chaque pair peut aussi définir sa propre "quantification", un multiple de la pulsation, par exemple une quantification de 4 peut impliquer une pulsation à chaque mesure en 4/4. La phase du temps est définie en fonction de la quantification, par exemple une phase de 0,5 d'une quantification de 4 sera la seconde pulsation de la mesure. Les pairs peuvent lire et écrire des lignes de temps avec un temps, une pulsation et une phase locaux, en comptant à partir du moment où le pair est activé, mais le tempo et la pulsation de toutes les lignes de temps pour tous les pairs dans la session coïncideront.

Le premier pair dans une session détermine le tempo initial. Après cela, le tempo n'est changé que si un pair appelle explicitement la fonction de fixation du tempo (`link_tempo_set`, dans Csound).

Le tempo de Link est indépendant du tempo de la partition de Csound. Les exécutions qui nécessitent la synchronisation du tempo de la partition avec le tempo de Link peuvent utiliser l'opcode tempo pour caler le tempo de la partition sur celui de Link ; ou inversement, fixer le tempo de Link sur celui de la partition au moyen de l'opcode tempoal.

Noter que la phase et la pulsation obtenus ou fixés par ces opcodes n'a que la précision permise par la durée de la période-k de Csound, le pilote audio utilisé par Csound, la latence et la stabilité du réseau et l'horloge la plus précise du système.

- *link\_create* - Crée un objet pair Ableton Link.
- *link\_enable* - Active/désactive la synchronisation avec le tempo et la pulsation de la session sur le réseau Ableton Link.
- *link\_is\_enabled* - Indique si ce pair Ableton Link s'est joint à la session du réseau.
- *link\_tempo\_set* - Fixe le tempo pour la session du réseau Ableton Link.
- *link\_tempo\_get* - Retourne le tempo de la session du réseau Ableton Link.
- *link\_beat\_get* - Retourne la pulsation, la phase et le temps courant d'Ableton Link pour cette session pour une quantification donnée.
- *link\_metro* - Retourne un déclencheur qui vaut 1 sur la pulsation et 0 ailleurs, ainsi que la pulsation, la phase et temps courant d'Ableton Link pour cette session pour une quantification donnée.
- *link\_beat\_request* - Demande une pulsation avec un nombre spécifique à un temps spécifique et une quantification donnée.
- *link\_beat\_force* - Force la session du réseau Ableton Link à adopter une pulsation avec un nombre spécifique à un temps spécifique et une quantification donnée.
- *link\_peers* - Retourne le nombre de pairs actuellement joints à la session dans le réseau Ableton Link.



# OSC

*OSC* permet l'interaction entre différents processus audio, et en particulier entre Csound et d'autres moteurs de synthèse. Les opcodes suivants sont disponibles :

- *OSCinit* - Démarre un thread d'écoute OSC.
- *OSClisten* - Réçoit les messages OSC.
- *OSCsend* - Envoie un message OSC.

## Crédits

Par John ffitich avec le support et l'inspiration de la bibliothèque liblo.

## Réseau

Les opcodes suivants peuvent envoyer ou recevoir des données audio en UDP :

- *sockrecv*
- *socksend*

## Opcodes pour le traitement à distance

Les opcodes pour le Traitement à Distance permettent la transmission d'une partition ou d'évènements MIDI à travers un réseau, pour un traitement par des instances distantes (ou une autre instance locale). Les opcodes suivants sont disponibles :

- *insglobal* - Utilisé pour implémenter un orchestre distant.
- *insremot* - Utilisé pour implémenter un orchestre distant.
- *midiglobal* - Utilisé pour implémenter un orchestre MIDI distant.
- *midiremot* - Utilisé pour implémenter un orchestre MIDI distant.
- *remoteport* - Définit le port à utiliser avec le système distant.

---

# Opcodes Mixer

La famille d'opcodes Mixer fournit un mélangeur global pour Csound. Les opcodes Mixer comprennent *MixerSend* pour envoyer (c'est-à-dire mélanger en entrée) un signal de taux-a depuis n'importe quel instrument vers un canal d'un bus de mixage, *MixerReceive* pour recevoir un signal de taux-a depuis un canal de n'importe quel bus de mixage dans un instrument, *MixerSetLevel* (taux-k) et *MixerSetLevel\_i* (taux-i) pour contrôler le niveau du signal envoyé d'une source particulière vers un bus particulier, *MixerGetLevel* pour lire (au taux-k) le niveau d'envoi d'une source particulière à un bus particulier, et *MixerClear* pour réinitialiser les bus à zéro avant la k-période suivante d'une exécution.

---

# Opcodes de graphe de fluence

Ces opcodes permettent d'utiliser des graphes de fluences (ou graphes de flots de données asynchrones) dans des orchestres de Csound. Les signaux s'écoulent depuis les prises de sortie (outlets) des instruments émetteurs et sont additionnés dans les prises d'entrée (inlets) des instruments récepteurs. Les signaux peuvent être de taux-k, de taux-a, de taux-f ou des tableaux de taux-a. On peut connecter n'importe quel nombre d'outlets à n'importe quel nombre d'inlets. Lorsqu'une nouvelle instance d'un instrument est créée pendant l'exécution, les connexions déclarées sont automatiquement instanciées.

Les graphes de fluence simplifient la construction de mélangeurs complexes, de chaînes de traitement du signal, etc. Ils simplifient également la réutilisation de définitions d'instrument "plug and play" et même de sous-orchestres entiers, qui peuvent être simplement insérés (`#include`) et ainsi "branchés" dans des orchestres existants.

Noter que les inlets et les outlets sont définis dans les instruments sans référence à la manière dont ils sont connectés. Les connexions sont définies dans l'en-tête de l'orchestre. C'est cette séparation qui permet d'avoir des instruments greffons.

Les inlets doivent être nommés. Les instruments peuvent être nommés ou numérotés, mais dans tous les cas chaque instrument émetteur doit être défini dans l'orchestre avant chacun de ses récepteurs. Si les instruments sont nommés, il est plus facile de connecter les outlets et les inlets d'un orchestre de niveau plus élevé aux inlets et aux outlets d'un orchestre inclu (`#include`) de niveau moins élevé.

Les opcodes de graphe de fluence comprennent : *outleta*, pour envoyer un signal de taux-a depuis n'importe quel instrument par un port nommé. *outletk*, pour envoyer un signal de taux-k depuis n'importe quel instrument par un port nommé. *outletkid*, semblable à *outletk*, mais il ne reçoit un signal de taux-k que depuis une instance identifiée d'un port. *outletf*, pour envoyer un signal de taux-f depuis n'importe quel instrument par un port nommé. *outletv*, pour envoyer un signal, tableau de taux-a, depuis n'importe quel instrument par un port nommé. *inleta*, pour recevoir un signal de taux-a à travers un port nommé. *inletk*, pour recevoir un signal de taux-k à travers un port nommé. *inletkid*, semblable à *inletk*, mais il ne transmet un signal qu'entre des opcodes inlet et outlet. *inletf*, pour recevoir un signal de taux-f à travers un port nommé. *inletv*, pour recevoir un signal, tableau de taux-a, à travers un port nommé. *connect*, pour acheminer le signal depuis un outlet nommé dans un instrument émetteur vers un inlet nommé dans un instrument récepteur. *alwayson* pour activer un instrument de façon permanente depuis l'en-tête de l'orchestre, sans l'aide d'une instruction de partition, par exemple pour l'utiliser comme processeur d'effet recevant ses entrées depuis un certain nombre d'émetteurs. *ftgenonce* pour instancier des tables de fonction depuis des définitions d'instrument, sans l'aide d'instructions-f dans la partition ou d'opcodes *ftgen* dans l'en-tête de l'orchestre.

Un scénario typique d'utilisation de ces opcodes ressemble à ceci ; un ensemble d'instruments est défini, chacun dans son propre fichier d'orchestre, et chaque instrument définit des ports d'entrée, des ports de sortie et des tables de fonction en son sein. De tels instruments sont complètement autonomes. Puis un ensemble de processeurs d'effets tels qu'égaliseurs, réverbérations, compresseurs, etc, sont également définis, chacun dans son propre fichier. Enfin un orchestre maître personnalisé inclut (`#include`) les instruments et les effets à utiliser, dirige les sorties de certains instruments dans un égaliseur et les sorties d'autres effets dans un autre égaliseur, puis achemine les sorties des deux égaliseurs dans une réverbération, la sortie de la réverbération dans un compresseur et la sortie du compresseur dans un fichier de sortie stéréo.

## Exemples

Voici un exemple des opcodes de graphe de fluence. Il utilise le fichier *signalflowgraph.csd* [examples/signalflowgraph.csd].

### Exemple 10. Exemple des opcodes de graphe de fluence.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o madsr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Michael Gogins */
; Initialize the global variables.
sr = 44100
ksmps = 100
nchnls = 2

; Connect up the instruments to create a signal flow graph.

connect "SimpleSine", "leftout", "Reverberator", "leftin"
connect "SimpleSine", "rightout", "Reverberator", "rightin"

connect "Moogy", "leftout", "Reverberator", "leftin"
connect "Moogy", "rightout", "Reverberator", "rightin"

connect "Reverberator", "leftout", "Compressor", "leftin"
connect "Reverberator", "rightout", "Compressor", "rightin"

connect "Compressor", "leftout", "Soundfile", "leftin"
connect "Compressor", "rightout", "Soundfile", "rightin"

; Turn on the "effect" units in the signal flow graph.

alwayson "Reverberator", 0.91, 12000
alwayson "Compressor"
alwayson "Soundfile"

instr SimpleSine
  ihz = cpsmidinn(p4)
  iamplitude = ampdb(p5)
  print ihz, iamplitude
  ; Use ftgenonce instead of ftgen, ftgentmp, or f statement.
  isine ftgenonce 0, 0, 4096, 10, 1
  a1 oscili iamplitude, ihz, isine
  aenv madsr 0.05, 0.1, 0.5, 0.2
  asignal = a1 * aenv
  ; Stereo audio outlet to be routed in the orchestra header.
  outleta "leftout", asignal * 0.25
  outleta "rightout", asignal * 0.75
endin

instr Moogy
  ihz = cpsmidinn(p4)
  iamplitude = ampdb(p5)
  ; Use ftgenonce instead of ftgen, ftgentmp, or f statement.
  isine ftgenonce 0, 0, 4096, 10, 1
  asignal vco iamplitude, ihz, 1, 0.5, isine
  kfco line 200, p3, 2000
  krez init 0.9
  asignal moogvcf asignal, kfco, krez, 100000
  ; Stereo audio outlet to be routed in the orchestra header.
  outleta "leftout", asignal * 0.75
  outleta "rightout", asignal * 0.25
endin

instr Reverberator
  ; Stereo input.

```

```

aleftin inleta "leftin"
arightin inleta "rightin"
idelay = p4
icutoff = p5
aleftout, arightout reverb5c aleftin, arightin, idelay, icutoff
; Stereo output.
outleta "leftout", aleftout
outleta "rightout", arightout
endin

instr Compressor
; Stereo input.
aleftin inleta "leftin"
arightin inleta "rightin"
kthreshold = 25000
icomp1 = 0.5
icomp2 = 0.763
irtime = 0.1
iftime = 0.1
aleftout dam aleftin, kthreshold, icomp1, icomp2, irtime, iftime
arightout dam arightin, kthreshold, icomp1, icomp2, irtime, iftime
; Stereo output.
outleta "leftout", aleftout
outleta "rightout", arightout
endin

instr Soundfile
; Stereo input.
aleftin inleta "leftin"
arightin inleta "rightin"
outs aleftin, arightin
endin

</CsInstruments>
<CsScore>
; Not necessary to activate "effects" or create f-tables in the score!
; Overlapping notes to create new instances of instruments.
i "SimpleSine" 1 5 60 85
i "SimpleSine" 2 5 64 80
i "Moogy" 3 5 67 75
i "Moogy" 4 5 71 70
e 1
</CsScore>
</CsSoundSynthesizer>

```

---

# Opcodes Jacko

Ces opcodes permettent l'utilisation des ports de Jack depuis les orchestres et les instruments de Csound. Les ports peuvent recevoir ou envoyer des données audio ou MIDI, et envoyer des données de note.

Les opcodes Jacko ne remplacent pas le pilote de Jack et les options de la ligne de commande de Csound pour Jack, et les opcodes Jacko ne travaillent pas avec ces derniers (d'où le nom "Jacko" au lieu de "Jack"). Les opcodes Jacko sont une facilité indépendante qui offre une plus grande flexibilité dans le routage du signal.

De plus, les opcodes Jacko peuvent travailler avec le système Jack en mode "roue libre", ce qui permet l'utilisation de synthétiseurs externes supportant Jack, comme Aeolus ou Pianoteq, pour restituer les pièces de Csound soit plus rapides, soit, ce qui est plus important, moins rapides que le temps réel. C'est très utile pour restituer des pièces complexes sans coupure au moyen d'instruments comme Aeolus, qui peut n'être accessible qu'à travers Jack.

Les opcodes Jacko comprennent : *JackoInit*, pour initialiser l'instance courante de Csound en tant que client Jack. *JackoInfo*, pour imprimer l'information sur le démon Jack, ses clients, leurs ports et leurs connexions. *JackoFreewheel*, pour activer ou désactiver le mode "roue libre" de Jack. *JackoAudioInConnect*, pour créer une connexion entre le port Jack d'une sortie audio externe et un port Jack d'entrée dans Csound. *JackoAudioOutConnect*, pour créer une connexion entre un port Jack de Csound et le port Jack d'une entrée audio externe. *JackoMidiInConnect*, pour créer une connexion depuis un port Jack MIDI externe. Les événements MIDI en provenance de Jack sont reçus par les opcodes MIDI réguliers de Csound et le système MIDI interop. *JackoMidiOutConnect*, pour créer une connexion entre un port Jack de Csound et le port Jack d'une entrée MIDI externe. *JackoOn*, pour activer ou désactiver les ports Jack de Csound. *JackoAudioIn*, pour recevoir les données audio d'un port Jack d'entrée dans Csound, qui les a lui-même reçu du port externe auquel il est connecté. *JackoAudioOut*, pour envoyer des données audio à un port Jack de sortie de Csound, qui les enverra lui-même vers le port externe auquel il est connecté. *JackoMidiOut*, pour envoyer des message de canal MIDI à un port Jack de sortie de Csound, qui les enverra lui-même vers le port externe auquel il est connecté. *JackoNoteOut*, pour envoyer une note (avec sa durée) à un port Jack de sortie de Csound, qui l'enverra lui-même vers le port externe auquel il est connecté. *JackoTransport*, pour contrôler le routage de Jack.

Un scénario d'utilisation typique des opcodes Jacko ressemblera à ceci.

## Exemples

Voici un exemple des opcodes Jacko. Il utilise le fichier *jacko.csd* [examples/jacko.csd].

### Exemple 11. Exemple des opcodes Jacko.

```
<CsoundSynthesizer>
<CsOptions>
csound -m255 -M0 -+rtmidi=null -Rwf --midi-key=4 --midi-velocity=5 -o jacko_test.wav
</CsOptions>
<CsInstruments>

;;;
;;; NOTE: this csd must be run after starting "aeolus -t".
;;;

sr          = 48000
; The control rate must be BOTH a power of 2 (for Jack)
; AND go evenly into sr. This is about the only one that works!
ksmps       = 128
nchnls      = 2
```

```

Odbfs      = 1

JackoInit  "default", "csound"

; To use ALSA midi ports, use "jackd -Xseq"
; and use "jack_lsp -A -c" or aliases from JackInfo,
; probably together with information from the sequencer,
; to figure out the damn port names.

; JackoMidiInConnect  "alsa_pcm:in-131-0-Master", "midiin"
JackoAudioInConnect "aeolus:out.L", "leftin"
JackoAudioInConnect "aeolus:out.R", "rightin"
JackoMidiOutConnect "midiout", "aeolus:Midi/in"

; Note that Jack enables audio to be output to a regular
; Csound soundfile and, at the same time, to a sound
; card in real time to the system client via Jack.

JackoAudioOutConnect "leftout", "system:playback_1"
JackoAudioOutConnect "rightout", "system:playback_2"
JackoInfo

; Turning freewheeling on seems automatically
; to turn system playback off. This is good!

JackoFreewheel 1
JackoOn

alwayson "jackin"

instr 1
; //////////////////////////////////////
ichannel = p1 - 1
itime = p2
iduration = p3
ikey = p4
ivelocity = p5
JackoNoteOut "midiout", ichannel, ikey, ivelocity
print itime, iduration, ichannel, ikey, ivelocity
endin

instr jackin
; //////////////////////////////////////
JackoTransport 3, 1.0
aleft JackoAudioIn "leftin"
aright JackoAudioIn "rightin"

; Aeolus uses MIDI controller 98 to control stops.
; Only 1 data value byte is used, not the 2 data
; bytes often used with NRPNS.
; The format for control mode is 01mm0ggg:
; mm 10 to set stops, 0, ggg group (or Division, 0 based).
; The format for stop selection is 000bbbb:
; bbbbb for button number (0 based).

; Mode to enable stops for Divison I: b1100010 (98
; [this controller VALUE is a pure coincidence]).

JackoMidiOut "midiout", 176, 0, 98, 98

; Stops: Principal 8 (0), Principal 4 (1) , Flote 8 (8) , Flote 2 (10)

JackoMidiOut "midiout", 176, 0, 98, 0
JackoMidiOut "midiout", 176, 0, 98, 1
JackoMidiOut "midiout", 176, 0, 98, 8
JackoMidiOut "midiout", 176, 0, 98, 10

```

```
; Sends audio coming in from Aeolus out  
; not only to the Jack system out (sound card),  
; but also to the output soundfile.  
    ; Note that in freewheeling mode, "leftout"  
    ; and "rightout" simply go silent.  
  
JackoAudioOut "leftout", aleft  
JackoAudioOut "rightout", aright  
outs    aright, aleft  
endin  
  
</CsInstruments>  
<CsScore>  
f 0 30  
i 1 1 30 60 60  
i 1 2 30 64 60  
i 1 3 30 71 60  
e 2  
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Par Michael Gogins, 2010



---

# Opcodes Python

## Introduction

En utilisant la famille d'opcodes Python, vous pouvez interagir avec un interpréteur Python embarqué dans Csound de cinq manières :

1. Initialiser l'interpréteur Python (les opcodes *pyinit*),
2. Exécuter une instruction (les opcodes *pyrun*),
3. Exécuter un script (les opcodes *pyexec*),
4. Invoquer un objet callable et lui passer des arguments (les opcodes *pycall*),
5. Evaluer une expression (les opcodes *pyeval*), ou
6. Changer la valeur d'un objet Python, avec la possibilité de créer un nouvel objet Python (les opcodes *pyassign*) ;

et vous pouvez faire toutes ces choses :

1. Au temps-i ou au temps-k,
2. Dans l'espace de nom global de Python, ou dans un espace de nom spécifique à une instance individuelle d'un instrument Csound (contexte local ou "l"),
3. Et vous pouvez récupérer de 0 à 8 valeurs de retour d'objets appelables qui acceptent N paramètres.

...cela signifie qu'il y a beaucoup d'opcodes concernant Python. Mais tous ces opcodes partagent le même préfixe *py*, et ils ont une structure de nom régulière :

"py" + [préfixe contextuel facultatif] + [nom d'action] + [suffixe de temps-x facultatif]

## Syntaxe de l'orchestre

Des blocs de code Python, voire des scripts entiers, peuvent être embarqués dans un orchestre Csound en utilisant les directives `{ { et } }` pour entourer le script, comme ci-dessous :

```
sr=44100
kr=4410
ksmps=10
nchnls=1
pyinit

giSinusoid ftgen 0, 0, 8192, 10, 1

pyruni {{
import random

pool = [(1 + i/10.0) ** 1.2 for i in range(100)]

def get_number_from_pool(n, p):
    if random.random() < p:
        i = int(random.random() * len(pool))
```

```
        pool[i] = n
    return random.choice(pool)
}}

instr 1
k1 oscil 1, 3, giSinusoid
k2 pycall1 "get_number_from_pool", k1 + 2, p4
    printk 0.01, k2
endin
```

## Crédits

Copyright © 2002 par Maurizio Umberto Puxeddu. Tous droits réservés.

Copyright © 2004 et 2005 par Michael Gogins, pour certaines parties.

---

# Opcodes pour le traitement d'image

Voici une liste des opcodes pour lire/écrire des fichiers d'image :

- *imagecreate*
- *imagesize*
- *imagegetpixel*
- *imagesetpixel*
- *imagesave*
- *imageload*
- *imagefree*

---

# Opcodes STK



## Note

Ces opcodes sont un composant facultatif de Csound6. On peut tester s'ils sont installés avec la commande "csound -z" qui donne la liste de tous les opcodes disponibles.

On peut compiler les opcodes STK de différentes manières. Si l'on compile tout Csound, il suffit de suivre les instructions au début du fichier Opcodes/stk/stkOpcodes.cpp.

La famille des opcodes STK comprend :

- *STKBandedWG*
- *STKBeeThree*
- *STKBlowBotl*
- *STKBlowHole*
- *STKBowed*
- *STKBrass*
- *STKClarinet*
- *STKDrummer*
- *STKFlute*
- *STKFMVoices*
- *STKHevyMetl*
- *STKMandolin*
- *STKModalBar*
- *STKMoog*
- *STKPercFlut*
- *STKPlucked*
- *STKResonate*
- *STKRhodey*
- *STKSaxofony*
- *STKShakers*
- *STKSimple*
- *STKSitar*
- *STKStifKarp*

- *STKTubeBell*
- *STKVoicForm*
- *STKWhistle*
- *STKWurley*

Pour plus d'information sur les opcodes STK, consulter le site web du Synthesis ToolKit en C++ à <https://ccrma.stanford.edu/software/stk>.

## Crédits

Cette implémentation des générateurs unitaires STK a été écrite par Perry R. Cook et Gary P. Scavone.

Les opcodes ont été ensuite adaptés comme greffon de Csound5 par Michael Gogins.

---

# Opcodes divers

Voici une liste d'opcodes qui ne rentrent dans aucune catégorie :

- *system* - appelle un programme externe via le mécanisme d'appel du système.
- *modmatrix* - opcode matrice de modulation avec optimisation pour les matrices creuses.

---

## Partie III. Référence

---

---

## Table des matières

Opcodes et opérateurs de l'orchestre .....	214
!= .....	215
#define .....	217
#include .....	221
#undef .....	224
#ifdef .....	225
#ifndef .....	226
\$NOM .....	227
% .....	230
&& .....	232
> .....	234
>= .....	236
lastcycle .....	238
< .....	240
<= .....	242
* .....	244
+ .....	246
- .....	248
/ .....	250
= .....	252
+= .....	254
== .....	256
^ .....	258
.....	260
! .....	262
Odbfs .....	264
A4 .....	267
<< .....	268
>> .....	270
& .....	271
.....	273
¬ .....	274
# .....	275
a .....	276
abs .....	278
active .....	280
adsr .....	284
adsyn .....	287
adsynt .....	289
adsynt2 .....	292
aftouch .....	295
alpass .....	297
alwayson .....	299
ampdb .....	302
ampdbfs .....	304
ampmidi .....	306
ampmidicurve .....	308
ampmidid .....	310
areson .....	312
aresonk .....	315
atone .....	317



atonek .....	319
atonex .....	321
ATSadd .....	323
ATSaddnz .....	327
ATSbufread .....	330
ATScross .....	332
ATSinfo .....	335
ATSinterpread .....	338
ATSread .....	340
ATSreadnz .....	344
ATSpartialtap .....	346
ATSinnoi .....	348
babo .....	352
balance .....	356
balance2 .....	358
bamboo .....	360
barmodel .....	362
bbcutm .....	364
bbcuts .....	369
beadsynt .....	372
beosc .....	376
betarand .....	379
bexprnd .....	382
bformenc1 .....	384
bformdec1 .....	387
binit .....	390
biquad .....	392
biquada .....	397
birnd .....	400
bpf .....	402
bpfcos .....	405
bqrez .....	408
butbp .....	410
butbr .....	411
buthp .....	412
butlp .....	413
butterbp .....	414
butterbr .....	416
butterhp .....	418
butterlp .....	420
button .....	422
buzz .....	424
c2r .....	426
cabasa .....	428
cauchy .....	430
cauchy1 .....	432
ceil .....	434
cell .....	436
cent .....	439
centroid .....	441
ceps .....	443
cepsinv .....	445
cggoto .....	447
chanctrl .....	449

changed	451
changed2	453
chani	456
chano	457
cbrt	458
chebyshevpoly	460
checkbox	463
chn	465
chnclear	467
chnexport	469
chnget	471
chnmix	474
chnparams	476
chnset	477
chuap	480
cigoto	484
ckgoto	486
clear	488
clfilt	490
clip	493
clockoff	496
clockon	498
cmp	500
cmplxprod	503
cngoto	505
comb	507
combinv	509
compress	511
compileorc	514
compilestr	516
compress2	518
compilecsd	521
connect	523
control	526
convle	527
convolve	528
copya2ftab	532
copyf2array	534
cos	536
cosseg	538
cossegb	540
cossegr	542
cosh	544
cosinv	546
cps2pch	548
cpsmidi	552
cpsmidib	554
cpsmidinn	556
cpsoct	560
cpspch	563
cpstmid	566
cpstun	568
cpstuni	571
cpsxpch	574

cpumeter .....	578
cpuprc .....	580
cross2 .....	583
crossfm .....	585
crunch .....	588
ctrl14 .....	590
ctrl21 .....	592
ctrl7 .....	594
ctrlinit .....	597
cuserrnd .....	598
dam .....	601
date .....	604
dates .....	606
db .....	608
dbamp .....	610
dbfsamp .....	612
dcblock .....	614
dcblock2 .....	616
dconv .....	618
dct .....	620
dctinv .....	622
deinterleave .....	624
delay .....	626
delay1 .....	628
delayk .....	630
delayr .....	633
delayw .....	635
deltap .....	637
deltap3 .....	640
deltapi .....	643
deltapn .....	646
deltapx .....	648
deltapxw .....	650
denorm .....	652
diff .....	654
diode_ladder .....	656
directory .....	659
diskgrain .....	661
diskin .....	664
diskin2 .....	667
dispfft .....	671
display .....	673
distort .....	675
distort1 .....	677
divz .....	679
doppler .....	681
dot .....	683
downsamp .....	684
dripwater .....	686
dssiactivate .....	688
dssiaudio .....	691
dssictls .....	693
dssiinit .....	695
dssilist .....	697

dumpk .....	699
dumpk2 .....	702
dumpk3 .....	705
dumpk4 .....	708
duserrnd .....	711
dust .....	713
dust2 .....	715
else .....	717
elseif .....	719
endif .....	721
endin .....	723
endop .....	725
envlpx .....	728
envlpxr .....	731
ephasor .....	734
eqfil .....	736
evalstr .....	738
event .....	739
event_i .....	743
exciter .....	745
exitnow .....	747
exp .....	749
expcurve .....	751
expon .....	753
exprand .....	755
exprandi .....	757
expseg .....	759
expsega .....	761
expsegb .....	763
expsegba .....	765
expsegr .....	767
faustaudio .....	769
faustcompile .....	771
faustctl .....	773
faustdsp .....	775
faustgen .....	777
faustplay .....	779
fareylen .....	781
fareyleni .....	783
ficlose .....	786
filebit .....	788
filelen .....	790
filenchnls .....	792
filepeak .....	794
filescale .....	796
filesr .....	798
filevalid .....	800
fillarray .....	802
fft .....	804
filter2 .....	806
fin .....	808
fini .....	810
fink .....	812
fiopen .....	814

flanger .....	816
flashtxt .....	818
FLbox .....	820
FLbutBank .....	825
FLbutton .....	828
FLcloseButton .....	833
FLcolor .....	836
FLcolor2 .....	838
FLcount .....	839
FLexecButton .....	842
FLgetsnap .....	846
FLgroup .....	847
FLgroupEnd .....	849
FLgroup_end .....	850
FLhide .....	851
FLhvsBox .....	852
FLhvsBoxSetValue .....	853
FLjoy .....	854
FLkeyIn .....	857
FLknob .....	859
FLlabel .....	864
FLloadsnap .....	866
FLmouse .....	867
flooper .....	869
flooper2 .....	871
floor .....	873
FLpack .....	875
FLpackEnd .....	878
FLpack_end .....	879
FLpanel .....	880
FLpanelEnd .....	884
FLpanel_end .....	885
FLprintk .....	886
FLprintk2 .....	887
FLroller .....	888
FLrun .....	891
FLsavesnap .....	892
FLscroll .....	898
FLscrollEnd .....	901
FLscroll_end .....	902
FLsetAlign .....	903
FLsetBox .....	904
FLsetColor .....	906
FLsetColor2 .....	908
FLsetFont .....	909
FLsetPosition .....	911
FLsetSize .....	912
FLsetsnap .....	913
FLsetSnapGroup .....	915
FLsetText .....	916
FLsetTextColor .....	918
FLsetTextSize .....	919
FLsetTextType .....	920
FLsetVal_i .....	923

FLsetVal .....	924
FLshow .....	925
FLslidBnk .....	926
FLslidBnk2 .....	930
FLslidBnkGetHandle .....	934
FLslidBnkSet .....	935
FLslidBnkSetk .....	936
FLslidBnk2Set .....	938
FLslidBnk2Setk .....	939
FLslider .....	942
FLtabs .....	948
FLtabsEnd .....	954
FLtabs_end .....	955
FLtext .....	956
FLupdate .....	959
fluidAllOut .....	960
fluidCCi .....	962
fluidCCk .....	964
fluidControl .....	966
fluidEngine .....	969
fluidInfo .....	972
fluidLoad .....	974
fluidNote .....	976
fluidOut .....	978
fluidProgramSelect .....	981
fluidSetInterpMethod .....	984
FLvalue .....	986
FLvkeybd .....	989
FLvslidBnk .....	990
FLvslidBnk2 .....	994
FLxyin .....	997
fmanal .....	1000
fmax .....	1002
fmb3 .....	1004
fmbell .....	1006
fmin .....	1009
fmmetal .....	1011
fmod .....	1014
fmpercfl .....	1016
fmrhode .....	1018
fmvoice .....	1021
fmwurlie .....	1023
fof .....	1026
fof2 .....	1029
fofilter .....	1035
fog .....	1037
fold .....	1040
follow .....	1042
follow2 .....	1044
foscil .....	1046
foscili .....	1048
fout .....	1050
fouti .....	1054
foutir .....	1056

foutk .....	1058
fprintks .....	1060
fprints .....	1066
frac .....	1068
fractalnoise .....	1070
framebuffer .....	1072
freeverb .....	1074
ftaudio .....	1076
ftchnls .....	1079
ftconv .....	1081
ftcps .....	1084
ftexists .....	1086
ftfree .....	1088
ftgen .....	1090
ftgenonce .....	1093
ftgentmp .....	1095
ftlen .....	1097
ftload .....	1099
ftloadk .....	1100
ftlptim .....	1101
ftmorf .....	1103
ftom .....	1105
ftprint .....	1107
ftsamplbank .....	1109
ftsave .....	1111
ftsavek .....	1113
ftslice .....	1114
ftsr .....	1116
gain .....	1118
gainslider .....	1120
gtf .....	1122
gauss .....	1123
gaussi .....	1125
gausstrig .....	1127
gbuzz .....	1130
genarray .....	1132
genarray_i .....	1134
gendy .....	1136
gendyc .....	1140
gendyx .....	1143
getcfg .....	1147
getcol .....	1149
getftargs .....	1151
getrow .....	1153
getrowlin .....	1156
getseed .....	1159
gogobel .....	1160
goto .....	1162
grain .....	1164
grain2 .....	1166
grain3 .....	1171
granule .....	1176
guiro .....	1180
harmon .....	1182

harmon2 .....	1184
hdf5read .....	1187
hdf5write .....	1189
hilbert .....	1191
hilbert2 .....	1196
hrtfearly .....	1198
hrtfmove .....	1202
hrtfmove2 .....	1205
hrtfreverb .....	1208
hrtfstat .....	1211
hsboscil .....	1214
hvs1 .....	1217
hvs2 .....	1221
hvs3 .....	1227
hypot .....	1230
i .....	1232
if .....	1233
fftinv .....	1238
igoto .....	1240
ihold .....	1242
imagecreate .....	1244
imagefree .....	1246
imagegetpixel .....	1248
imageload .....	1251
imagesave .....	1253
imagesetpixel .....	1255
imagesize .....	1257
in .....	1259
in32 .....	1261
inch .....	1262
inh .....	1264
init .....	1265
initc14 .....	1268
initc21 .....	1269
initc7 .....	1270
inleta .....	1272
inletk .....	1275
inletkid .....	1277
inletf .....	1278
inletv .....	1279
ino .....	1280
inq .....	1281
inrg .....	1283
ins .....	1284
insremot .....	1286
insglobal .....	1289
instr .....	1291
int .....	1293
integ .....	1295
interleave .....	1297
interp .....	1299
invalue .....	1302
inx .....	1304
inz .....	1305



JackoAudioIn .....	1306
JackoAudioInConnect .....	1307
JackoAudioOut .....	1308
JackoAudioOutConnect .....	1309
JackoFreewheel .....	1310
JackoInfo .....	1311
JackoInit .....	1313
JackoMidiInConnect .....	1315
JackoMidiOutConnect .....	1316
JackoMidiOut .....	1317
JackoNoteOut .....	1318
JackoOn .....	1319
JackoTransport .....	1320
jacktransport .....	1321
jitter .....	1323
jitter2 .....	1325
joystick .....	1327
jspline .....	1330
k .....	1332
K35_hpf .....	1333
K35_lpf .....	1337
kgoto .....	1341
kr .....	1343
ksmps .....	1344
lenarray .....	1345
lfo .....	1347
limit .....	1349
limit1 .....	1351
line .....	1353
linen .....	1355
linenr .....	1357
lineto .....	1360
linlin .....	1362
lincos .....	1364
link_beat_force .....	1366
link_beat_get .....	1367
link_beat_request .....	1368
link_create .....	1369
link_enable .....	1371
link_is_enabled .....	1373
link_metro .....	1375
link_peers .....	1377
link_tempo_get .....	1379
link_tempo_set .....	1381
linrand .....	1383
linseg .....	1385
linsegb .....	1387
linsegr .....	1389
liveconv .....	1391
locsend .....	1394
locsig .....	1397
log .....	1400
log10 .....	1402
log2 .....	1404

logbtwo .....	1406
logcurve .....	1408
loop_ge .....	1410
loop_gt .....	1411
loop_le .....	1412
loop_lt .....	1415
loopseg .....	1418
loopsepg .....	1421
looptseg .....	1423
loopxseg .....	1425
lorenz .....	1427
lorisread .....	1430
lorismorph .....	1433
lorisplay .....	1436
loscil .....	1439
loscil3 .....	1442
loscilx .....	1445
lowpass2 .....	1448
lowres .....	1450
lowresx .....	1452
lpf18 .....	1454
lpfreson .....	1456
lphasor .....	1458
lpinterp .....	1460
lposcil .....	1461
lposcil3 .....	1463
lposcila .....	1465
lposcilsa .....	1467
lposcilsa2 .....	1469
lpread .....	1471
lpreson .....	1474
lpshold .....	1477
lpsholdp .....	1479
lpslot .....	1480
mac .....	1481
maca .....	1483
madsr .....	1485
mags .....	1489
mandel .....	1492
mandol .....	1495
maparray .....	1497
marimba .....	1500
massign .....	1503
max .....	1506
maxabs .....	1508
maxabsaccum .....	1510
maxaccum .....	1512
maxalloc .....	1514
max_k .....	1516
maxarray .....	1518
mclock .....	1520
mdelay .....	1522
median .....	1524
mediank .....	1526

metro	1528
metro2	1530
mfb	1533
midglobal	1535
midiarp	1536
midic14	1538
midic21	1540
midic7	1542
midichannelaftertouch	1544
midichn	1546
midicontrolchange	1549
midictrl	1551
mididefault	1553
midiin	1555
midifilestatus	1558
midinoteoff	1559
midinoteoncps	1561
midinoteonkey	1563
midinoteonoct	1565
midinoteonpch	1567
midion2	1569
midion	1571
midiout	1574
midiout_i	1576
midipitchbend	1578
midipolyaftertouch	1581
midiprogramchange	1584
miditempo	1586
midremot	1588
min	1591
minabs	1593
minabsaccum	1595
minaccum	1597
mincer	1599
minarray	1601
mirror	1603
MixerSetLevel	1605
MixerSetLevel_i	1608
MixerGetLevel	1609
MixerSend	1611
MixerReceive	1613
MixerClear	1615
mode	1617
modmatrix	1620
monitor	1625
moog	1627
moogladder	1629
moogladder2	1631
moogvcf	1633
moogvcf2	1635
moscil	1637
mp3in	1639
mp3len	1641
mp3scal	1643

mpulse .....	1645
mrtmsg .....	1647
mtof .....	1648
mton .....	1650
multitap .....	1652
mute .....	1654
mvchpf .....	1656
mvclpf1 .....	1658
mvclpf2 .....	1660
mvclpf3 .....	1662
mvclpf4 .....	1664
mxadsr .....	1666
nchnls .....	1669
nchnls_hw .....	1671
nchnls_i .....	1672
nestedap .....	1674
nlfilt .....	1677
nlfilt2 .....	1680
noise .....	1683
noteoff .....	1686
noteon .....	1687
noteondur2 .....	1688
noteondur .....	1690
notnum .....	1692
nreverb .....	1694
nrpn .....	1697
nsamp .....	1699
nstance .....	1701
ntof .....	1704
ntom .....	1706
nstrnum .....	1708
nstrstr .....	1709
ntrpol .....	1710
octave .....	1712
octcps .....	1714
octmidi .....	1717
octmidib .....	1719
octmidinn .....	1721
octpch .....	1724
olabuffer .....	1727
opcode .....	1729
oscbnk .....	1735
oscil1 .....	1741
oscil1i .....	1743
oscil3 .....	1745
oscil .....	1747
oscili .....	1749
oscilikt .....	1751
osciliktp .....	1753
oscilikts .....	1755
osciln .....	1757
oscils .....	1759
oscilx .....	1761
OSCbundle .....	1762

OSCcount .....	1764
OSCinit .....	1766
OSCinitM .....	1768
OSClisten .....	1770
OSCrow .....	1774
OSCsend .....	1776
out32 .....	1778
out .....	1779
outc .....	1781
outch .....	1783
outh .....	1786
outiat .....	1787
outic14 .....	1789
outic .....	1791
outipat .....	1793
outipb .....	1794
outipc .....	1796
outkat .....	1798
outkc14 .....	1800
outkc .....	1801
outkpat .....	1803
outkpb .....	1804
outkpc .....	1806
outleta .....	1809
outletf .....	1811
outletk .....	1812
outletkid .....	1814
outletv .....	1815
outo .....	1816
outq1 .....	1817
outq2 .....	1819
outq3 .....	1821
outq4 .....	1823
outq .....	1825
outrg .....	1827
outs1 .....	1829
outs2 .....	1831
outs .....	1833
outvalue .....	1835
outx .....	1837
outz .....	1838
p5gconnect .....	1839
p5gdata .....	1841
p .....	1843
pan2 .....	1845
pan .....	1847
pareq .....	1849
partials .....	1852
partikkel .....	1854
partikkelget .....	1865
partikkelset .....	1868
partikkelsync .....	1871
passign .....	1875
paulstretch .....	1877

pcauchy .....	1879
pchbend .....	1881
pchmidi .....	1883
pchmidib .....	1885
pchmidinn .....	1887
pchoct .....	1890
pctom .....	1893
pconvolve .....	1895
pcount .....	1898
pdclip .....	1901
pdhalf .....	1904
pdhalfy .....	1907
peak .....	1911
pgmassign .....	1913
phaser1 .....	1917
phaser2 .....	1920
phasor .....	1924
phasorbnk .....	1926
phs .....	1928
pindex .....	1931
pinker .....	1935
pinkish .....	1936
pitch .....	1939
pitchamdf .....	1942
planet .....	1945
platerev .....	1948
plltrack .....	1950
pluck .....	1952
poisson .....	1954
pol2rect .....	1958
polyaft .....	1961
polynomial .....	1963
port .....	1966
portk .....	1968
poscil3 .....	1970
poscil .....	1973
pow .....	1975
powershape .....	1977
powoftwo .....	1979
prealloc .....	1981
prepiano .....	1983
print .....	1986
printf .....	1988
printk2 .....	1990
printk .....	1992
printks .....	1994
printks2 .....	1997
prints .....	2000
printarray .....	2002
product .....	2005
product .....	2007
pset .....	2008
ptable .....	2010
ptablei .....	2012

ptable3 .....	2015
ptablew .....	2016
ptrack .....	2019
puts .....	2021
pvadd .....	2023
pvbufread .....	2027
pvcross .....	2029
pvinterp .....	2032
pvoc .....	2035
pvread .....	2038
pvsadsyn .....	2040
pvsanal .....	2043
pvsarp .....	2046
pvsbandp .....	2049
pvsbandr .....	2051
pvsbin .....	2053
pvsblur .....	2055
pvsbuffer .....	2057
pvsbufread .....	2058
pvsbufread2 .....	2061
pvscale .....	2063
pvscent .....	2065
pvsceps .....	2067
pvcross .....	2069
pvsdemix .....	2071
pvsdiskin .....	2073
pvsdisp .....	2075
pvsfilter .....	2077
pvsfread .....	2080
pvsfreeze .....	2082
pvsftr .....	2084
pvsftw .....	2086
pvsfwrite .....	2089
pvsgain .....	2091
pvshift .....	2093
pvsifd .....	2095
pvsinfo .....	2097
pvsinit .....	2099
pvsin .....	2100
pvslock .....	2101
pvsmaska .....	2103
pvmix .....	2105
pvmorph .....	2107
pvsMOOTH .....	2110
pvsout .....	2112
pvsosc .....	2113
pvspitch .....	2116
pvstanal .....	2119
pvstencil .....	2121
pvstrace .....	2123
pvsvoc .....	2125
pvsynth .....	2127
pvsWarp .....	2129
pvs2tab .....	2131

pyassign Opcodes .....	2133
pycall Opcodes .....	2134
pyeval Opcodes .....	2138
pyexec Opcodes .....	2139
pyinit Opcodes .....	2142
pyrun Opcodes .....	2143
pwd .....	2145
qinf .....	2147
qnan .....	2149
r2c .....	2151
rand .....	2153
randh .....	2155
randi .....	2158
random .....	2160
randomh .....	2162
randomi .....	2165
rbjeq .....	2168
readclock .....	2171
readf .....	2173
readfi .....	2175
readk .....	2177
readk2 .....	2180
readk3 .....	2183
readk4 .....	2186
readscore .....	2189
readscratch .....	2191
rect2pol .....	2193
reinit .....	2195
release .....	2197
remoteport .....	2198
remove .....	2199
repluck .....	2200
reshapearray .....	2202
reson .....	2204
resonk .....	2206
resonr .....	2208
resonx .....	2212
resonxk .....	2214
resony .....	2216
resonz .....	2218
resyn .....	2222
return .....	2224
reverb .....	2225
reverb2 .....	2227
reverb3 .....	2228
rewindscore .....	2230
rezzy .....	2232
rfft .....	2234
rifft .....	2237
rigoto .....	2239
rireturn .....	2240
rms .....	2242
rnd .....	2244
rnd31 .....	2246



round .....	2252
rspline .....	2254
rtclock .....	2256
S .....	2258
s16b14 .....	2259
s32b14 .....	2261
samphold .....	2263
sandpaper .....	2265
scale .....	2267
scalearray .....	2269
scanhammer .....	2271
scans .....	2272
scantable .....	2276
scanu .....	2278
schedkwhen .....	2280
schedkwhennamed .....	2283
schedule .....	2286
schedulek .....	2289
schedwhen .....	2290
scoreline .....	2292
scoreline_i .....	2294
sc_lag .....	2296
sc_lagud .....	2298
sc_phasor .....	2300
sc_trig .....	2302
seed .....	2304
sekere .....	2306
select .....	2308
semitone .....	2310
sense .....	2312
sensekey .....	2313
serialBegin .....	2317
serialEnd .....	2319
serialFlush .....	2320
serialPrint .....	2321
serialRead .....	2322
serialWrite_i .....	2324
serialWrite .....	2325
seqtime2 .....	2327
seqtime .....	2330
setcol .....	2333
setctrl .....	2335
setksmps .....	2338
setrow .....	2340
setscorepos .....	2343
sfilist .....	2345
sfinstr3 .....	2347
sfinstr3m .....	2350
sfinstr .....	2353
sfinstrm .....	2356
sfload .....	2358
sflooper .....	2361
sfpassign .....	2364
sfplay3 .....	2367

sfplay3m .....	2370
sfplay .....	2373
sfplaym .....	2375
sfplist .....	2378
sfpreset .....	2380
shaker .....	2383
shiftin .....	2385
shiftout .....	2387
signum .....	2389
sin .....	2391
sinh .....	2393
sininv .....	2395
sinsyn .....	2397
sleighbells .....	2399
slicearray .....	2401
slider16 .....	2403
slider16f .....	2405
slider16table .....	2407
slider16tablef .....	2409
slider32 .....	2411
slider32f .....	2413
slider32table .....	2415
slider32tablef .....	2417
slider64 .....	2419
slider64f .....	2421
slider64table .....	2423
slider64tablef .....	2425
slider8 .....	2427
slider8f .....	2429
slider8table .....	2431
slider8tablef .....	2433
sliderKawai .....	2435
sndloop .....	2436
sndwarp .....	2438
sndwarpst .....	2442
sockrecv .....	2446
socksend .....	2448
sorta .....	2450
sortd .....	2451
soundin .....	2452
space .....	2455
spat3d .....	2460
spat3di .....	2469
spat3dt .....	2473
spdist .....	2478
specaddm .....	2482
specdiff .....	2483
specdisp .....	2484
specfilt .....	2485
spechist .....	2486
specptrk .....	2487
specscal .....	2489
specsum .....	2490
spectrum .....	2491

splitrig .....	2493
sprintf .....	2495
sprintfk .....	2497
spsend .....	2499
sqrt .....	2502
squnewave .....	2504
sr .....	2509
statevar .....	2511
stix .....	2513
STKBandedWG .....	2515
STKBeeThree .....	2517
STKBlowBotl .....	2519
STKBlowHole .....	2521
STKBowed .....	2523
STKBrass .....	2525
STKClarinet .....	2527
STKDrummer .....	2529
STKFlute .....	2531
STKFMVoices .....	2533
STKHevyMetl .....	2535
STKMandolin .....	2537
STKModalBar .....	2539
STKMoog .....	2541
STKPercFlut .....	2543
STKPlucked .....	2545
STKResonate .....	2547
STKRhodey .....	2549
STKSaxofony .....	2551
STKShakers .....	2553
STKSimple .....	2555
STKSitar .....	2557
STKStifKarp .....	2559
STKTubeBell .....	2561
STKVoicForm .....	2563
STKWhistle .....	2565
STKWurley .....	2567
strchar .....	2569
strchark .....	2571
strcpy .....	2572
strcpyk .....	2573
strcat .....	2575
strcatk .....	2577
strcmp .....	2579
strcmpk .....	2581
streson .....	2582
strfromurl .....	2584
strget .....	2586
strindex .....	2588
strindexk .....	2589
strlen .....	2591
strlenk .....	2592
strlower .....	2593
strlowerk .....	2595
strrindex .....	2596

strrindexk .....	2598
strset .....	2599
strstrip .....	2601
strsub .....	2603
strsubk .....	2605
strtod .....	2606
strtodk .....	2607
strtol .....	2608
strtolk .....	2609
strupper .....	2610
strupperk .....	2611
subinstr .....	2612
subinstrinit .....	2615
sum .....	2616
sumarray .....	2618
svfilter .....	2620
syncgrain .....	2623
syncloop .....	2626
syncphasor .....	2628
system .....	2632
tab .....	2634
tabifd .....	2636
table .....	2638
table3 .....	2640
tablecopy .....	2641
tablefilter .....	2644
tablefilteri .....	2646
tablegpw .....	2648
tablei .....	2649
tableicopy .....	2652
tableigpw .....	2653
tableikt .....	2654
tableimix .....	2657
tablekt .....	2659
tablemix .....	2662
tableng .....	2664
tablera .....	2666
tableseg .....	2669
tablesuffle .....	2671
tablew .....	2674
tablewa .....	2677
tablewkt .....	2680
tablexkt .....	2682
tablexseg .....	2685
tabmorph .....	2687
tabmorpha .....	2690
tabmorphak .....	2693
tabmorphi .....	2696
tabplay .....	2699
tabrec .....	2700
tabrowlin .....	2702
tabsum .....	2705
tab2array .....	2707
tab2pvs .....	2709

tambourine .....	2711
tan .....	2713
tanh .....	2715
taninv .....	2717
taninv2 .....	2719
tbvcf .....	2721
tempest .....	2724
tempo .....	2727
temposcal .....	2729
tempoval .....	2731
tigoto .....	2733
timedseq .....	2735
timeinstk .....	2738
timeinsts .....	2740
timek .....	2742
times .....	2744
timeout .....	2747
tival .....	2749
tlineto .....	2751
tone .....	2753
tonek .....	2755
tonex .....	2757
trandom .....	2759
tradsyn .....	2761
transeg .....	2763
transegb .....	2765
transegr .....	2767
trcross .....	2769
trfilter .....	2771
trhighest .....	2773
trigger .....	2775
trigseq .....	2777
trim .....	2781
trirand .....	2783
trlowest .....	2785
trmix .....	2787
trscale .....	2789
trshift .....	2791
trsplitt .....	2793
turnoff .....	2795
turnoff2 .....	2797
turnon .....	2800
tvconv .....	2801
unirand .....	2803
until .....	2806
unwrap .....	2808
upsamp .....	2811
urandom .....	2813
urd .....	2816
vactrol .....	2819
vadd .....	2821
vadd_i .....	2824
vaddv .....	2826
vaddv_i .....	2829

vaget .....	2831
valpass .....	2833
vaset .....	2836
vbap .....	2838
vbapmove .....	2841
vbapg .....	2844
vbapgmove .....	2847
vbap16 .....	2850
vbap16move .....	2852
vbap4 .....	2854
vbap4move .....	2857
vbap8 .....	2860
vbap8move .....	2862
vbaplsinit .....	2865
vbapz .....	2868
vbapzmove .....	2870
vcella .....	2872
vco .....	2875
vco2 .....	2878
vco2ft .....	2882
vco2ift .....	2884
vco2init .....	2886
vcomb .....	2889
vcopy .....	2892
vcopy_i .....	2895
vdelay .....	2897
vdelay3 .....	2899
vdelayx .....	2901
vdelayxq .....	2903
vdelayxs .....	2905
vdelayxw .....	2907
vdelayxwq .....	2909
vdelayxws .....	2911
vdelayk .....	2913
vdivv .....	2914
vdivv_i .....	2917
vecdelay .....	2919
veloc .....	2920
vexp .....	2922
vexp_i .....	2925
vexpseg .....	2927
vexpv .....	2929
vexpv_i .....	2932
vibes .....	2934
vibr .....	2936
vibrato .....	2938
vincr .....	2940
vlimit .....	2943
vlinseg .....	2944
vlowres .....	2946
vmap .....	2948
vmirror .....	2950
vmult .....	2951
vmult_i .....	2955

vmultv .....	2958
vmultv_i .....	2961
voice .....	2963
vosim .....	2966
vphaseseg .....	2971
vport .....	2973
vpow .....	2974
vpow_i .....	2978
vpowv .....	2981
vpowv_i .....	2984
vpvoc .....	2986
vrandh .....	2989
vrandi .....	2992
vsubv .....	2995
vsubv_i .....	2998
vtable1k .....	3000
vtablei .....	3002
vtablek .....	3004
vtablea .....	3006
vtablewi .....	3008
vtablewk .....	3009
vtablewa .....	3011
vtabi .....	3013
vtabk .....	3015
vtaba .....	3017
vtabwi .....	3019
vtabwk .....	3020
vtabwa .....	3021
vwrap .....	3022
waveset .....	3023
websocket .....	3025
weibull .....	3027
wgbow .....	3030
wgbowedbar .....	3032
wgbrass .....	3034
wgclar .....	3036
wgflute .....	3038
wgpluck .....	3040
wgpluck2 .....	3043
wguide1 .....	3045
wguide2 .....	3048
while .....	3051
wiiconnect .....	3053
wiidata .....	3055
wiirange .....	3058
wiisend .....	3059
window .....	3061
wrap .....	3064
writescratch .....	3066
wterrain .....	3068
xadsr .....	3070
xin .....	3072
xout .....	3074
xscanmap .....	3076

xscansmap .....	3079
xscans .....	3080
xscanu .....	3084
xtratim .....	3088
xyin .....	3092
xyscale .....	3094
zacl .....	3096
zakinit .....	3098
zamod .....	3101
zar .....	3103
zarg .....	3105
zaw .....	3107
zawm .....	3109
zdf_1pole .....	3112
zdf_1pole_mode .....	3114
zdf_2pole .....	3116
zdf_2pole_mode .....	3119
zdf_ladder .....	3121
zfilter2 .....	3124
zir .....	3126
ziw .....	3128
ziwm .....	3130
zkcl .....	3132
zkmod .....	3134
zkr .....	3136
zkw .....	3138
zkwm .....	3140
Instructions de partition et routines GEN .....	3143
Instructions de partition .....	3143
Instruction a (ou instruction avancer) .....	3144
Instruction b .....	3146
Instruction C .....	3148
Instruction d (instruction de note) .....	3150
Instruction e .....	3152
Instruction f (ou instruction de table de fonction) .....	3154
Instruction i (instruction d'instrument ou de note) .....	3157
Instruction m (instruction de marquage) .....	3161
Instruction n .....	3163
Instruction q .....	3165
Instruction r (instruction répéter) .....	3167
Instruction s .....	3169
Instruction t (instruction de tempo) .....	3171
Instruction v .....	3173
Instruction x .....	3175
Instruction y (ou instruction graine) .....	3177
Instruction { .....	3179
Instruction } .....	3182
Routines GEN .....	3182
GEN01 .....	3186
GEN02 .....	3189
GEN03 .....	3191
GEN04 .....	3194
GEN05 .....	3197
GEN06 .....	3199



GEN07 .....	3201
GEN08 .....	3203
GEN09 .....	3205
GEN10 .....	3208
GEN11 .....	3211
GEN12 .....	3213
GEN13 .....	3216
GEN14 .....	3220
GEN15 .....	3223
GEN16 .....	3231
GEN17 .....	3234
GEN18 .....	3236
GEN19 .....	3239
GEN20 .....	3241
GEN21 .....	3244
GEN23 .....	3248
GEN24 .....	3250
GEN25 .....	3252
GEN27 .....	3254
GEN28 .....	3256
GEN30 .....	3259
GEN31 .....	3261
GEN32 .....	3263
GEN33 .....	3265
GEN34 .....	3268
GEN40 .....	3271
GEN41 .....	3273
GEN42 .....	3275
GEN43 .....	3277
GEN49 .....	3278
GEN51 .....	3280
GEN52 .....	3283
GEN53 .....	3286
GENtanh .....	3288
GENexp .....	3290
GENsone .....	3292
GENquadbezier .....	3294
GENfarey .....	3297
GENwave .....	3302
GENpadsynth .....	3305
Opcodes de l'orchestre expérimentaux et routines GEN .....	3309
Opcodes de l'orchestre expérimentaux .....	3309
cudanal .....	3310
cudasynth .....	3313
cudasliding .....	3315
Opcodes de l'orchestre et routines GEN obsolètes .....	3317
Opcodes de l'orchestre obsolètes .....	3317
abetarand .....	3318
abexprnd .....	3319
acauchy .....	3320
aexprand .....	3321
agauss .....	3322
agogobel .....	3323
alinrand .....	3324

apcauchy .....	3325
apoisson .....	3326
apow .....	3327
array .....	3328
atrirand .....	3330
aunirand .....	3331
aweibull .....	3332
bformdec .....	3333
bformenc .....	3335
clock .....	3337
hrtfer .....	3338
ibetarand .....	3340
ibexprnd .....	3341
icauchy .....	3342
ictrl14 .....	3343
ictrl21 .....	3344
ictrl7 .....	3345
iexprand .....	3346
igauss .....	3347
ilinrand .....	3348
imdic14 .....	3349
imdic21 .....	3350
imdic7 .....	3351
instimek .....	3352
instimes .....	3353
ioff .....	3354
ion .....	3355
iondur2 .....	3356
iondur .....	3357
ioutat .....	3358
ioutc14 .....	3359
ioutc .....	3360
ioutpat .....	3361
ioutpb .....	3362
ioutpc .....	3363
ipcauchy .....	3364
ipoisson .....	3365
ipow .....	3366
is16b14 .....	3367
is32b14 .....	3368
islider16 .....	3369
islider32 .....	3370
islider64 .....	3371
islider8 .....	3372
itablecopy .....	3373
itablegpw .....	3374
itablemix .....	3375
itablew .....	3376
itrirand .....	3377
iunirand .....	3378
iweibull .....	3379
kbetarand .....	3380
kbexprnd .....	3381
kcauchy .....	3382

kdump2 .....	3383
kdump3 .....	3384
kdump4 .....	3385
kdump .....	3386
kexprand .....	3387
kfilter2 .....	3388
kgauss .....	3389
klinrand .....	3390
kon .....	3391
koutat .....	3392
koutc14 .....	3393
koutc .....	3394
koutpat .....	3395
koutpb .....	3396
koutpc .....	3397
kpcauchy .....	3398
kpoisson .....	3399
kpow .....	3400
kread2 .....	3401
kread3 .....	3402
kread4 .....	3403
kread .....	3404
ktableseg .....	3405
ktirand .....	3406
kunirand .....	3407
kweibull .....	3408
sndload .....	3409
peakk .....	3411
pop .....	3412
pop_f .....	3414
push .....	3415
push_f .....	3417
soundout .....	3418
soundouts .....	3420
stack .....	3421
tb .....	3423
tableiw .....	3426
Routines GEN obsolètes .....	3429
GEN22 .....	3430
Les programmes utilitaires .....	3431
Répertoires. ....	3431
Formats des fichiers son. ....	3431
Génération d'un fichier d'analyse (ATSA, CVANAL, HETRO, LPANAL, PVANAL) .	3432
Requêtes sur un fichier (SNDINFO) .....	3443
Conversion de fichier (, HET_EXPORT, HET_IMPORT, PVLOOK, PV_EXPORT,	
PV_IMPORT, SDIF2AD, SRCONV) .....	3444
Autres utilitaires de Csound (CS, CSB64ENC, ENVEXT, EXTRACTOR, MA-	
KECSD, MIXER, SCALE, MKDB) .....	3463
Cscore .....	3477
Événements, listes et opérations .....	3477
Ecrire un programme de contrôle Cscore .....	3480
Compiler un programme Cscore .....	3485
Exemples plus avancés .....	3488
Csbeats .....	3490

---

# Opcodes et opérateurs de l'orchestre

# !=

!= — Détermine si une valeur n'est pas égale à l'autre.

## Description

Détermine si une valeur n'est pas égale à l'autre.

## Syntaxe

```
(a != b ? v1 : v2)
```

où  $a$ ,  $b$ ,  $v1$  et  $v2$  peuvent être des expressions, mais  $a$ ,  $b$  pas de taux audio.

## Exécution

Dans l'expression conditionnelle ci-dessus,  $a$  et  $b$  sont d'abord comparés. Si la relation indiquée est vraie ( $a$  n'est pas égal à  $b$ ), alors l'expression conditionnelle prend la valeur de  $v1$  ; si la relation est fausse, l'expression prend la valeur de  $v2$ .

Nota Bene : Si  $v1$  ou  $v2$  sont des expressions, elles seront évaluées avant que l'expression conditionnelle ne soit déterminée.

En termes de précedence, tous les opérateurs conditionnels (c-à-d. les opérateurs relationnels (<, etc.), et ?, et : ) sont plus faibles que les opérateurs arithmétiques et logiques (+, -, \*, /, && et //).

Ce sont des *opérateurs* pas des *opcodes*. C'est pourquoi on peut les utiliser dans les instructions de l'orchestre, mais ils ne forment pas des instructions complètes par eux-mêmes.

## Exemples

Voici un exemple de l'opérateur !=. Il utilise le fichier *notequal.csd* [examples/notequal.csd].

### Exemple 12. Exemple de l'opérateur !=.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o notequal.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
```

```

ipch  = cpspch(p4)
iprint = p5
if (iprint != 1) goto skipPrint
print ipch
asig  vco .7, ipch, 2, 0.5, 1
      outs asig, asig

skipPrint:

endin

</CsInstruments>
<CsScore>
f 1 0 65536 10 1 ;sine wave

i1 0 .5 8.00 0
i1 + .5 8.01 1 ; this note will print it's ipch value and only this one will be played
i1 + .5 8.02 2

e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1: ipch = 277.167
```

## Voir aussi

`=, >=, >, <=, <`

# #define

#define — Définit une macro.

## Description

Les macros sont des substitutions de texte qui sont faites dans l'orchestre lors de sa lecture. Le système de macro de Csound est très simple, et il utilise les caractères # et \$ pour définir et appeler les macros. Il permet d'économiser de la frappe et peut conduire à une structure cohérente dans un style consistant. Il est similaire, tout en étant indépendant, au *système de macros du langage de partition*.

*#define NOM* -- définit une macro simple. Le nom de la macro doit commencer par une lettre et peut comprendre n'importe quelle combinaison de lettres et de chiffres. La casse est significative. Cette forme est limitée dans le sens que les noms de variable sont fixes. On peut obtenir plus de flexibilité en utilisant une macro avec arguments, décrite ci-dessous.

*#define NOM(a' b' c')* -- définit une macro avec arguments. On peut l'utiliser dans des situations plus complexes. Le nom de la macro doit commencer par une lettre et peut comprendre n'importe quelle combinaison de lettres et de chiffres. Dans le texte de substitution, les arguments sont appelés sous la forme : \$A. En fait, l'implémentation définit les arguments comme des macros simples. Il peut y avoir jusqu'à 5 arguments, et les noms sont une combinaison quelconque de lettres. Souvenez-vous que la casse est significative dans les noms de macro.

## Syntaxe

```
#define NOM # texte de substitution #
```

```
#define NOM(a' b' c') # texte de substitution #
```

## Initialisation

*# texte de substitution #* -- Le texte de substitution est une chaîne de caractères (ne contenant pas de #) et peut s'étendre sur plusieurs lignes. Le texte de substitution est entouré par des caractères #, ce qui garantit qu'aucun caractère supplémentaire ne sera capturé par inadvertance.

## Exécution

Il faut prendre certaines précautions avec les macros de substitution de texte, car elles peuvent parfois produire d'étranges résultats. Elles ne tiennent compte d'aucune valeur sémantique, et ainsi les espaces sont significatifs. C'est pourquoi, au contraire du langage C, la définition délimite le texte de substitution par des caractères #. Utilisé avec discernement, ce système de macro est un concept puissant, mais il peut aussi être mal employé.

## Exemples

Voici un exemple simple de définition de macro. Il utilise le fichier *define.csd* [examples/define.csd].

### Exemple 13. Exemple simple de définition de macro.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o define.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Define the macros.
#define VOLUME #5000#
#define FREQ #440#
#define TABLE #1#

; Instrument #1
instr 1
; Use the macros.
; This will be expanded to "a1 oscil 5000, 440, 1".
a1 oscil $VOLUME, $FREQ, $TABLE

; Send it to the output.
out a1
endin

</CsInstruments>
<CsScore>

; Define Table #1 with an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie présentera des lignes comme celles-ci :

```

Macro definition for VOLUME
Macro definition for CPS
Macro definition for TABLE

```

Voici un exemple simple de définition de macro avec arguments. Il utilise le fichier *define\_args.csd* [examples/define\_args.csd].

#### Exemple 14. Exemple simple de définition de macro avec arguments.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O

```



```

; For Non-realtime ouput leave only the line below:
; -o define_args.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Define the oscillator macro.
#define OSCMACRO(VOLUME'FREQ'TABLE) #oscil $VOLUME, $FREQ, $TABLE#

; Instrument #1
instr 1
; Use the oscillator macro.
; This will be expanded to "a1 oscil 5000, 440, 1".
a1 $OSCMACRO(5000'440'1)

; Send it to the output.
out a1
endin

</CsInstruments>
<CsScore>

; Define Table #1 with an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie présentera des lignes comme celle-ci :

Macro definition for OSCMACRO

## Macros Prédéfinies de Constantes Mathématiques

A partir de Csound 5.04 des Macros de Constantes Mathématiques sont prédéfinies. Les valeurs définies sont celles que l'on trouve dans le fichier d'en-tête C math.h, et elles sont automatiquement définies au démarrage de Csound et disponibles pour une utilisation dans les orchestres.

Macro	Valeur	Equivalent à
\$M_E	2.7182818284590452354	e
\$M_LOG2E	1.4426950408889634074	log <sub>2</sub> (e)
\$M_LOG10E	0.43429448190325182765	log <sub>10</sub> (e)
\$M_LN2	0.69314718055994530942	log <sub>e</sub> (2)
\$M_LN10	2.30258509299404568402	log <sub>e</sub> (10)
\$M_PI	3.14159265358979323846	pi
\$M_PI_2	1.57079632679489661923	pi/2
\$M_PI_4	0.78539816339744830962	pi/4

Macro	Valeur	Equivalent à
\$M_1_PI	0.31830988618379067154	1/pi
\$M_2_PI	0.63661977236758134308	2/pi
\$M_2_SQRTPI	1.12837916709551257390	2/sqrt(pi)
\$M_SQRT2	1.41421356237309504880	sqrt(2)
\$M_SQRT1_2	0.70710678118654752440	1/sqrt(2)

## Voir aussi

*\$NOM, #undef*

## Crédits

Auteur : John ffitch  
 University of Bath/Codemist Ltd.  
 Bath, UK  
 Avril 1998

Exemples écrits par Kevin Conder.

Nouveau dans la version 3.48 de Csound

# #include

#include — Inclut un fichier externe pour traitement.

## Description

Inclut un fichier externe pour traitement.

## Syntaxe

```
#include « nomfichier »
```

```
#includestr « nomfichier »
```

## Exécution

#includestr est semblable à #include sauf que le nom du fichier doit être délimité par une apostrophe double et que ce nom de fichier peut subir un développement de macro.

Il est parfois commode d'organiser un orchestre sur plusieurs fichiers, par exemple avec chaque instrument dans un fichier séparé. Ce style est supporté par la fonctionnalité *#include* qui fait partie du système de macros. Une ligne contenant le texte

```
#include "nomfichier"
```

où le caractère " peut être remplacé par n'importe quel caractère approprié dans #include (mais pas dans #includestr). Pour la plupart des utilisations le symbole de l'apostrophe double sera probablement le plus commode. Le nom de fichier peut inclure un chemin complet.

L'entrée est prise à partir du fichier nommé jusqu'à son terme, puis revient à la source précédente. *Note :* les versions de Csound antérieures à la 4.19 limitaient à 20 la profondeur des fichiers inclus et des macros.

Il est également suggéré d'utiliser *#include* pour définir un ensemble de macros qui font partie du style du compositeur.

A la limite, on pourrait définir chaque instrument comme une macro, avec un numéro d'instrument en paramètre. On pourrait alors construire un orchestre entier à partir d'un certain nombre d'instructions *#include* suivies par des appels de macro.

```
#include "clarinet"  
#include "flute"  
#include "bassoon"  
$CLARINET(1)  
$FLUTE(2)  
$BASSOON(3)
```

Il faut insister sur le fait que ces changements ont lieu au niveau littéral et n'ont donc aucune incidence sémantique.

Si Csound a été construit avec la bibliothèque CURL le nom de fichier dans une instruction *#include* peut être une URL, reconnue en incluant la sous-chaîne "://" dans le nom. Cela inclura du texte via des protocoles comme http, https et ftp.

## Exemples

Voici un exemple de l'opcode `include`. Il utilise les fichiers `include.csd` [examples/include.csd], et `table1.inc` [examples/table1.inc].

### Exemple 15. Exemple de l'opcode `include`.

```
/* table1.inc */
; Table #1, a sine wave.
f 1 0 16384 10 1
/* table1.inc */
```

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o include.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a basic oscillator.
instr 1
  kamp = 10000
  kcps = 440
  ifn = 1

  a1 oscil kamp, kcps, ifn
  out a1
endin

</CsInstruments>
<CsScore>

; Include the file for Table #1.
#include "table1.inc"

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK  
Avril 1998

#includestr nouveau dans la version 6.12

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.48 de Csound

Les URLs sont disponibles depuis Csound 6.02

# #undef

#undef — Annule la définition d'une macro.

## Description

Les macros sont des substitutions de texte qui sont faites dans l'orchestre lors de sa lecture. Le système de macro de Csound est très simple, et il utilise les caractères # et \$ pour définir et appeler les macros. Il permet d'économiser de la frappe et peut conduire à une structure cohérente dans un style consistant. Il est similaire, tout en étant indépendant, au *système de macros du langage de partition*.

*#undef NOM* -- annule la définition d'un nom de macro. Si une macro n'est plus nécessaire, on peut annuler sa définition avec *#undef NOM*.

## Syntaxe

**#undef** NOM

## Exécution

Il faut prendre certaines précautions avec les macros de substitution de texte, car elles peuvent parfois produire d'étranges résultats. Elles ne tiennent compte d'aucune valeur sémantique, et ainsi les espaces sont significatifs. C'est pourquoi, au contraire du langage C, la définition délimite le texte de substitution par des caractères #. Utilisé avec discernement, ce système de macro est un concept puissant, mais il peut aussi être mal employé.

## Voir aussi

*#define*, *\$NOM*

## Crédits

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK  
Avril 1998

Nouveau dans la version 3.48 de Csound

# #ifdef

`#ifdef` — Lecture de code conditionnelle.

## Description

Si une macro est définie alors *#ifdef* peut incorporer du texte dans un orchestre jusqu'au prochain *#end*. C'est similaire, tout en étant indépendant, au *système de macros du langage de partition*.

## Syntaxe

```
#ifdef NOM
....
#else
....
#end
```

## Exécution

Noter que l'on peut imbriquer les *#ifdef*, comme dans le langage du préprocesseur C.

## Exemples

Voici un exemple simple de cette insertion conditionnelle.

### Exemple 16. Exemple simple de la forme *#ifdef*.

```
#define debug ##
instr 1
#ifdef debug
  print "calling oscil"
#end
a1  oscil 32000,440,1
out a1
endin
```

## Voir aussi

*\$NOM*, *#define*, *#ifndef*.

## Crédits

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK  
Avril 2005

Nouveau dans Csound5 (et 4.23f13)

# #ifndef

`#ifndef` — Lecture de code conditionnelle.

## Description

Si la macro spécifiée n'est pas définie alors `#ifndef` peut incorporer du texte dans un orchestre jusqu'au prochain `#end`. C'est similaire, tout en étant indépendant, au *système de macros du langage de partition*.

## Syntaxe

```
#ifndef NOM
....
#else
....
#end
```

## Exécution

Noter que l'on peut imbriquer les `#ifndef`, comme dans le langage du préprocesseur C.

## Exemples

Voici un exemple simple de ce code conditionnel.

### Exemple 17. Exemple simple de la forme `#ifndef`.

```
#define final ##
instr 1
#ifndef final
print "calling oscil"
#end
a1 oscil 32000,440,1
out a1
endin
```

## Voir aussi

`$NOM`, `#define`, `#ifdef`.

## Crédits

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK  
Avril 2005

Nouveau dans Csound5 (et 4.23f13)



# \$NOM

\$NOM — Appelle une macro définie.

## Description

Les macros sont des substitutions de texte qui sont faites dans l'orchestre lors de sa lecture. Le système de macro de Csound est très simple, et il utilise les caractères # et \$ pour définir et appeler les macros. Il permet d'économiser de la frappe et peut conduire à une structure cohérente dans un style consistant. Il est similaire, tout en étant indépendant, au *système de macros du langage de partition*.

*\$NOM \$NOM.* -- appelle une macro définie. Pour appeler une macro, on utilise son nom précédé du caractère \$. La fin du nom est marquée par le premier caractère qui n'est ni une lettre ni un chiffre. S'il est nécessaire que le nom soit suivi d'une lettre ou d'un chiffre, un point, qui sera ignoré, peut être utilisé pour terminer le nom. La chaîne, *\$NOM.*, est remplacée par le texte de substitution de la définition. Le texte de substitution peut lui-même comprendre des appels de macro.

## Syntaxe

\$NOM

## Initialisation

*# texte de substitution #* -- Le texte de substitution est une chaîne de caractères (ne contenant pas de #) et peut s'étendre sur plusieurs lignes. Le texte de substitution est entouré par des caractères #, ce qui garantit qu'aucun caractère supplémentaire ne sera capturé par inadvertance.

## Exécution

Il faut prendre certaines précautions avec les macros de substitution de texte, car elles peuvent parfois produire d'étranges résultats. Elles ne tiennent compte d'aucune valeur sémantique, et ainsi les espaces sont significatifs. C'est pourquoi, au contraire du langage C, la définition délimite le texte de substitution par des caractères #. Utilisé avec discernement, ce système de macro est un concept puissant, mais il peut aussi être mal employé.

## Exemples

Voici un exemple d'appel de macro. Il utilise le fichier *define.csd* [examples/define.csd].

### Exemple 18. Un exemple d'appel de macro.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o define.wav -W ;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Define the macros.
#define VOLUME #5000#
#define FREQ #440#
#define TABLE #1#

; Instrument #1
instr 1
; Use the macros.
; This will be expanded to "a1 oscil 5000, 440, 1".
a1 oscil $VOLUME, $FREQ, $TABLE

; Send it to the output.
out a1
endin

</CsInstruments>
<CsScore>

; Define Table #1 with an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie présentera des lignes comme celles-ci :

```

Macro definition for VOLUME
Macro definition for CPS
Macro definition for TABLE

```

Voici un exemple d'appel de macro avec arguments. Il utilise le fichier *define\_args.csd* [exemples/define\_args.csd].

### Exemple 19. Un exemple d'appel de macro avec arguments.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o define_args.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410

```

```
ksmps = 10
nchnls = 1

; Define the oscillator macro.
#define OSCMACRO(VOLUME'FREQ'TABLE) #oscil $VOLUME, $FREQ, $TABLE#

; Instrument #1
instr 1
    ; Use the oscillator macro.
    ; This will be expanded to "a1 oscil 5000, 440, 1".
    a1 $OSCMACRO(5000'440'1)

    ; Send it to the output.
    out a1
endin

</CsInstruments>
<CsScore>

; Define Table #1 with an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

La sortie présentera des lignes comme celle-ci :

Macro definition for OSCMACRO

## Voir aussi

*#define, #undef*

## Crédits

Auteur : John ffitch  
 University of Bath/Codemist Ltd.  
 Bath, UK  
 Avril 1998

Exemples écrits par Kevin Conder.

Nouveau dans la version 3.48 de Csound

# %

% — Opérateur modulo.

## Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de signe inchangé, ET logique, OU logique, addition, soustraction, multiplication et division. Notez qu'une valeur ou une expression peut être placée entre deux de ces opérateurs, lesquels peuvent la prendre comme opérande de gauche ou de droite, comme dans

$a + b * c$ .

Trois règles s'appliquent dans de tels cas :

1.  $*$  et  $/$  s'appliquent à leurs voisins plus fortement que  $+$  et  $-$ . Ainsi l'expression ci-dessus s'interprète comme

$a + (b * c)$

avec  $*$  prenant  $b$  et  $c$  puis  $+$  prenant  $a$  et  $b * c$ .

2.  $+$  et  $-$  sont prioritaires sur  $\&\&$ , qui devance lui-même  $\|$  :

$a \&\& b - c \| d$

est interprété comme

$(a \&\& (b - c)) \| d$

3. Quand deux opérateurs sont d'égale importance, les opérations ont lieu de gauche à droite :

$a - b - c$

est interprété comme

$(a - b) - c$

On peut utiliser des parenthèses pour forcer un groupement particulier.

L'opérateur  $\%$  retourne la valeur de la réduction de  $a$  par  $b$ , de telle façon que le résultat, en valeur absolue, est inférieur à la valeur absolue de  $b$ , par soustraction répétée. C'est l'équivalent de la fonction modulo pour les entiers. Nouveau dans la version 3.50 de Csound.

## Syntaxe

`a % b` (pas de restriction de taux)

où les arguments  $a$  et  $b$  peuvent être des expressions.

## Arguments

Les arguments de  $\%$  peuvent être des valeurs scalaires ou des tableaux (vecteurs) unidimensionnels de taux- $k$ , ou n'importe quelle combinaison des deux. Si l'un des arguments est un tableau, la valeur retournée le sera également.

## Exemples

Voici un exemple de l'opérateur %. Il utilise le fichier *modulus.csd* [examples/modulus.csd].

### Exemple 20. Exemple de l'opérateur %.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -Ma --midi-key=4 --midi-velocity-amp=5 -m0 ;;realtime audio out and midi in
;-iadc ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o %.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giScale ftgen 1, 0, -12, -2, 0, -34, -7, 10, -14, 3, -21, -3, -27, -10, 7, -17 ;12 note scale with detu

instr 1

ikey = p4
ivel = p5
indx = ikey % 12 ;work on the twelftone scale
icent tab_i indx, giScale ;load the scale
ifreqeq = cpsmidinn(ikey)
ifreq = ifreqeq * cent(icent) ;change frequency by cents from table
prints "Key %d modulus 12 = %d. ", ikey, indx
prints "Equal-tempered frequency of this key = %f,", ifreqeq
prints " but here with cent deviation %d = %f%n", icent, ifreq
asig vco2 ivel*.5, ifreq
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 0 60 ;run for 60 seconds

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

-, +, &&, //, \*, /, ^

## Crédits

L'arithmétique sur les vecteurs est nouvelle dans le version 6.00

# &&

&& — Opérateur ET logique.

## Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de signe inchangé, ET logique, OU logique, addition, soustraction, multiplication et division. Notez qu'une valeur ou une expression peut être placée entre deux de ces opérateurs, lesquels peuvent la prendre comme opérande de gauche ou de droite, comme dans

$a + b * c$ .

Trois règles s'appliquent dans de tels cas :

1.  $*$  et  $/$  s'appliquent à leurs voisins plus fortement que  $+$  et  $-$ . Ainsi l'expression ci-dessus s'interprète comme

$a + (b * c)$

avec  $*$  prenant  $b$  et  $c$  puis  $+$  prenant  $a$  et  $b * c$ .

2.  $+$  et  $-$  sont prioritaires sur  $&&$ , qui devance lui-même  $\parallel$  :

$a \&\& b - c \parallel d$

est interprété comme

$(a \&\& (b - c)) \parallel d$

3. Quand deux opérateurs sont d'égale importance, les opérations ont lieu de gauche à droite :

$a - b - c$

est interprété comme

$(a - b) - c$

On peut utiliser des parenthèses pour forcer un groupement particulier.

## Syntaxe

`a && b` (ET logique ; pas de taux audio)

où les arguments *a* et *b* peuvent être des expressions.

## Exemples

Voici un exemple de l'opérateur logique AND. Il utilise le fichier *opand.csd* [examples/opand.csd].

### Exemple 21. Exemple de l'opcode opand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o opand.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kval    randomh 0, 1.2, 20 ;choose between 0 and 1.2
if kval >0 && kval<=.5 then ;3 possible outcomes
    kval = 1
elseif kval >.5 && kval<=1 then
    kval =2
elseif kval >1 then
    kval =3
endif

printk2 kval    ;print value when it changes
asig    poscil .7, 440*kval, 1
        outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 16384 10 1

i 1 0 5
e
</CsScore>
</CsoundSynthesizer>

```

La sortie contiendra des lignes comme celles-ci :

```

i1      0.00000
i1      3.00000
i1      1.00000
i1      3.00000
i1      1.00000
i1      2.00000
i1      3.00000
.....

```

## Voir aussi

-, +, //, \*, /, ^, %

## &gt;

> — Détermine si une valeur est supérieure à l'autre.

## Description

Détermine si une valeur est supérieure à l'autre.

## Syntaxe

```
(a > b ? v1 : v2)
```

où  $a$ ,  $b$ ,  $v1$  et  $v2$  peuvent être des expressions, mais  $a$ ,  $b$  pas de taux audio.

## Exécution

Dans l'expression conditionnelle ci-dessus,  $a$  et  $b$  sont d'abord comparés. Si la relation indiquée est vraie ( $a$  supérieur à  $b$ ), alors l'expression conditionnelle prend la valeur de  $v1$  ; si la relation est fausse, l'expression prend la valeur de  $v2$ .

Nota Bene : Si  $v1$  ou  $v2$  sont des expressions, elles seront évaluées avant que l'expression conditionnelle ne soit déterminée.

En termes de précedence, tous les opérateurs conditionnels (c-à-d. les opérateurs relationnels (<, etc.), et ?, et : ) sont plus faibles que les opérateurs arithmétiques et logiques (+, -, \*, /, && et //).

Ce sont des *opérateurs* pas des *opcodes*. C'est pourquoi on peut les utiliser dans les instructions de l'orchestre, mais ils ne forment pas des instructions complètes par eux-mêmes.

## Exemples

Voici un exemple de l'opérateur >. Il utilise le fichier *greaterthan.csd* [examples/greaterthan.csd].

### Exemple 22. Exemple de l'opérateur >.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o greaterthan.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1
```



```

; Instrument #1.
instr 1
; Get the 4th p-field from the score.
k1 = p4

; Is it greater than 3? (1 = true, 0 = false)
k2 = (p4 > 3 ? 1 : 0)

; Print the values of k1 and k2.
printks "k1 = %f, k2 = %f\\n", 1, k1, k2
endin

</CsInstruments>
<CsScore>

; Call Instrument #1 with a p4 = 2.
i 1 0 0.5 2
; Call Instrument #1 with a p4 = 3.
i 1 1 0.5 3
; Call Instrument #1 with a p4 = 4.
i 1 2 0.5 4
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme :

```

k1 = 2.000000, k2 = 0.000000
k1 = 3.000000, k2 = 0.000000
k1 = 4.000000, k2 = 1.000000

```

## Voir aussi

`==`, `>=`, `<=`, `<`, `!=`

## Crédits

Exemple écrit par Kevin Conder.

## >=

>= — Détermine si une valeur est supérieure ou égale à l'autre.

## Description

Détermine si une valeur est supérieure ou égale à l'autre.

## Syntaxe

```
(a >= b ? v1 : v2)
```

où  $a$ ,  $b$ ,  $v1$  et  $v2$  peuvent être des expressions, mais  $a$ ,  $b$  pas de taux audio.

## Exécution

Dans l'expression conditionnelle ci-dessus,  $a$  et  $b$  sont d'abord comparés. Si la relation indiquée est vraie ( $a$  supérieur ou égal à  $b$ ), alors l'expression conditionnelle prend la valeur de  $v1$  ; si la relation est fausse, l'expression prend la valeur de  $v2$ .

Nota Bene : Si  $v1$  ou  $v2$  sont des expressions, elles seront évaluées avant que l'expression conditionnelle ne soit déterminée.

En terme de précedence, tous les opérateurs conditionnels (c-à-d. les opérateurs relationnels (<, etc.), et ?, et : ) sont plus faibles que les opérateurs arithmétiques et logiques (+, -, \*, /, && et //).

Ce sont des *opérateurs* pas des *opcodes*. C'est pourquoi on peut les utiliser dans les instructions de l'orchestre, mais ils ne forment pas des instructions complètes par eux-mêmes.

## Exemples

Voici un exemple de l'opérateur >=. Il utilise le fichier *greaterequal.csd* [examples/greaterequal.csd].

### Exemple 23. Exemple de l'opérateur >=.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o greaterequal.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1
```

```

; Instrument #1.
instr 1
; Get the 4th p-field from the score.
k1 = p4

; Is it greater than or equal to 3? (1 = true, 0 = false)
k2 = (p4 >= 3 ? 1 : 0)

; Print the values of k1 and k2.
printks "k1 = %f, k2 = %f\\n", 1, k1, k2
endin

</CsInstruments>
<CsScore>

; Call Instrument #1 with a p4 = 2.
i 1 0 0.5 2
; Call Instrument #1 with a p4 = 3.
i 1 1 0.5 3
; Call Instrument #1 with a p4 = 4.
i 1 2 0.5 4
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme :

```

k1 = 2.000000, k2 = 0.000000
k1 = 3.000000, k2 = 1.000000
k1 = 4.000000, k2 = 1.000000

```

## Voir aussi

`=, >, <=, <, !=`

## Crédits

Exemple écrit par Kevin Conder.

# lastcycle

lastcycle — Indique si un évènement est dans son dernier cycle d'exécution.

## Description

Fournit un moyen de savoir si un évènement est dans son dernier cycle d'exécution. Qu'un évènement ait une durée finie ( $p3 > 0$ ) ou qu'une note soit rallongée via un opcode de prolongement de la durée (comme « linsegr » ou « xtratim »), cet opcode retourne 1 si l'évènement est dans son dernier cycle-k. La seule situation dans laquelle « lastcycle » ne détecte pas qu'un évènement ne continuera pas durant un autre cycle est lorsque la note est arrêtée par un autre évènement (en utilisant « turnoff2 ») sans avoir la possibilité de passer par l'étape de relâchement (en utilisant « turnoff2 instrnum, imode, 0 »).

## Syntaxe

kflag **lastcycle**

## Exécution

*kflag* -- indique si la note est dans son « dernier cycle ». (1 si c'est le dernier cycle, 0 sinon)

Cet opcode est utile pour faire des actions de nettoyage, en signalant aux autres évènements que cet évènement se termine, etc. Il ne fonctionne que durant l'exécution.

## Exemples

Voici un exemple de l'opcode lastcycle. Il utilise le fichier *lastcycle.csd* [examples/lastcycle.csd].

### Exemple 24. Exemple de l'opcode lastcycle.

```
<CsoundSynthesizer>

<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  Silent  MIDI in
-odac
-d
</CsOptions>

<CsInstruments>

sr = 44100
ksmps = 64
nchnls = 2
0dbfs = 1

instr 1
  aenv linsegr 0, 0.1, 1, 0.1, 0
  asig = oscili(0.1, 1000)
  asig += oscili(0.1, 1012)
  asig *= aenv
  if lastcycle() == 1 then
    schedulek p1, 0, p3
  endif
endif
```

```
    outs asig, asig
  endin

</CsInstruments>

<CsScore>

  i 1 0 0.5
  f 0 3600

</CsScore>

</CsoundSynthesizer>
```

## Voir aussi

*xtratim, release, linsegr*

## Crédits

Par : Eduardo Moguillansky 2020

Nouveau dans la version 6.14 de Csound (2020).

## &lt;

< — Détermine si une valeur est inférieure à l'autre.

## Description

Détermine si une valeur est inférieure à l'autre.

## Syntaxe

`( a < b ? v1 : v2 )`

où  $a$ ,  $b$ ,  $v1$  et  $v2$  peuvent être des expressions, mais  $a$ ,  $b$  pas de taux audio.

## Exécution

Dans l'expression conditionnelle ci-dessus,  $a$  et  $b$  sont d'abord comparés. Si la relation indiquée est vraie ( $a$  inférieur à  $b$ ), alors l'expression conditionnelle prend la valeur de  $v1$  ; si la relation est fausse, l'expression prend la valeur de  $v2$ .

Nota Bene : Si  $v1$  ou  $v2$  sont des expressions, elles seront évaluées avant que l'expression conditionnelle ne soit déterminée.

En terme de précedence, tous les opérateurs conditionnels (c-à-d. les opérateurs relationnels (<, etc.), et ?, et : ) sont plus faibles que les opérateurs arithmétiques et logiques (+, -, \*, /, && et //).

Ce sont des *opérateurs* pas des *opcodes*. C'est pourquoi on peut les utiliser dans les instructions de l'orchestre, mais ils ne forment pas des instructions complètes par eux-mêmes.

## Exemples

Voici un exemple de l'opérateur <. Il utilise le fichier *lessthan.csd* [examples/lessthan.csd].

### Exemple 25. Exemple de l'opérateur <.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o <.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
```

```

ipch = p4
ipitch = (ipch < 15 ? cpspch(ipch) : ipch) ;if p4 is lower then 15, it assumes p4 to be pitch-class
print ipitch ;and not meant to be a frequency in Hertz
asig poscil .5, ipitch , 1
    outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1 ;sine wave

i1 0 3 8.00 ;pitch class
i1 4 3 800 ;frequency

e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme :

```

ipitch = 261.626
ipitch = 800.00

```

## Voir aussi

`=, >=, >, <=, !=`

## <=

<= — Détermine si une valeur est inférieure ou égale à l'autre.

## Description

Détermine si une valeur est inférieure ou égale à l'autre.

## Syntaxe

```
(a <= b ? v1 : v2)
```

où *a*, *b*, *v1* et *v2* peuvent être des expressions, mais *a*, *b* pas de taux audio.

## Exécution

Dans l'expression conditionnelle ci-dessus, *a* et *b* sont d'abord comparés. Si la relation indiquée est vraie (*a* inférieur ou égal à *b*), alors l'expression conditionnelle prend la valeur de *v1* ; si la relation est fausse, l'expression prend la valeur de *v2*.

Nota Bene : Si *v1* ou *v2* sont des expressions, elles seront évaluées avant que l'expression conditionnelle ne soit déterminée.

En termes de précedence, tous les opérateurs conditionnels (c-à-d. les opérateurs relationnels (<, etc.), et ?, et : ) sont plus faibles que les opérateurs arithmétiques et logiques (+, -, \*, /, && et //).

Ce sont des *opérateurs* pas des *opcodes*. C'est pourquoi on peut les utiliser dans les instructions de l'orchestre, mais ils ne forment pas des instructions complètes par eux-mêmes.

## Exemples

Voici un exemple de l'opérateur <=. Il utilise le fichier *lessequal.csd* [examples/lessequal.csd].

### Exemple 26. Exemple de l'opérateur <=.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o <=.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
```



```
kval    randomh 0, 1.2, 20 ;choose between 0 and 1.2

if kval >0 && kval<=.5 then ;3 possible outcomes
    kv1 = 1
elseif kval >.5 && kval<=1 then
    kv1 =2
elseif kval >1 then
    kv1 =3
endif

printks "random number = %f, result = %f\n", .1, kval, kv1
asig    poscil .7, 440*kv1, 1
        outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 16384 10 1

i 1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
random number = 1.035781, result = 3.000000
random number = 0.134037, result = 1.000000
random number = 0.130742, result = 1.000000
random number = 1.002550, result = 3.000000
random number = 0.370565, result = 1.000000
random number = 0.655759, result = 2.000000
random number = 0.676154, result = 2.000000
.....
```

## Voir aussi

`==, >=, >, <, !=`

## \*

\* — Opérateur de multiplication

## Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de conservation de signe, le ET et le OU logiques, l'addition, la soustraction, la multiplication et la division. A noter qu'une valeur ou une expression peut se trouver entre deux de ces opérateurs, pour lesquels elle sera l'argument de gauche ou l'argument de droite comme dans

$a + b * c$ .

Dans de tels cas trois règles s'appliquent :

1. \* et / s'appliquent à leurs voisins plus fortement que + et -. Ainsi l'expression ci-dessus est interprétée comme

$a + (b * c)$

avec \* s'appliquant à b et à c et ensuite + s'appliquant à a et à  $b * c$ .

2. + et - sont prioritaires par rapport à &&, qui est lui-même prioritaire par rapport à ||:

$a \&\& b - c || d$

est interprété comme

$(a \&\& (b - c)) || d$

3. Lorsque deux opérateurs ont le même rang de priorité, les opérations s'enchaînent de gauche à droite :

$a - b - c$

est interprété comme

$(a - b) - c$

On peut utiliser des parenthèses comme ci-dessus pour forcer un groupement particulier.

## Syntaxe

`a * b` (pas de restriction de taux)

où les arguments *a* et *b* peuvent être des expressions.

## Arguments

Les arguments de \* peuvent être des valeurs scalaires ou des tableaux (vecteurs) unidimensionnels de taux-k, ou n'importe quelle combinaison des deux. Si l'un des arguments est un tableau, la valeur retournée le sera également.

## Exemples

Voici un exemple de l'opérateur \*. Il utilise le fichier *multiplies.csd* [examples/multiplies.csd].

## Exemple 27. Exemple de l'opérateur \*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o multiplies.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kenv      expon 0.01, p3, 1
aout      poscil 0.8*kenv, 440, 1 ;multiply amplitude from 0 to 1 * 0.8
printks   "base amplitude * rising expon output = 0.8 * %f\n", .1, kenv
outs      aout, aout
endin

</CsInstruments>
<CsScore>

f 1 0 16384 10 1 ; sine wave

i 1 0 2

e

</CsScore>
</CsoundSynthesizer>
```

La sortie contiendra des lignes comme celles-ci :

```
base amplitude * rising expon output = 0.8 * 0.010000
base amplitude * rising expon output = 0.8 * 0.012589
.....
base amplitude * rising expon output = 0.8 * 0.794328
base amplitude * rising expon output = 0.8 * 0.998466
```

## Voir aussi

-, +, &&, //, /, ^, %

## Crédits

L'arithmétique sur les vecteurs est nouvelle dans la version 6.00

# **+**

+ — Opérateur d'addition

## **Description**

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de conservation de signe, le ET et le OU logiques, l'addition, la soustraction, la multiplication et la division. A noter qu'une valeur ou une expression peut se trouver entre deux de ces opérateurs, pour lesquels elle sera l'argument de gauche ou l'argument de droite comme dans

$a + b * c.$

Dans de tels cas trois règles s'appliquent :

1.  $*$  et  $/$  s'appliquent à leurs voisins plus fortement que  $+$  et  $-$ . Ainsi l'expression ci-dessus est interprétée comme

$a + (b * c)$

avec  $*$  s'appliquant à  $b$  et à  $c$  et ensuite  $+$  s'appliquant à  $a$  et à  $b * c$ .

2.  $+$  et  $-$  sont prioritaires par rapport à  $\&\&$ , qui est lui-même prioritaire par rapport à  $\|$ :

$a \&\& b - c \| d$

est interprété comme

$(a \&\& (b - c)) \| d$

3. Lorsque deux opérateurs ont le même rang de priorité, les opérations s'enchaînent de gauche à droite :

$a - b - c$

est interprété comme

$(a - b) - c$

On peut utiliser des parenthèses comme ci-dessus pour forcer un groupement particulier.

## **Syntaxe**

$+a$  (pas de restriction de taux)

$a + b$  (pas de restriction de taux)

où les arguments  $a$  et  $b$  peuvent être des expressions.

## **Arguments**

Les arguments de  $+$  peuvent être des valeurs scalaires ou des tableaux (vecteurs) unidimensionnels de taux- $k$ , ou n'importe quelle combinaison des deux. Si l'un des arguments est un tableau, la valeur retournée le sera également.

## Exemples

Voici un exemple de l'opérateur +. Il utilise le fichier *adds.csd* [examples/adds.csd].

### Exemple 28. Exemple de l'opérateur +.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o adds.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
; add unipolar square to oscil
kamp = p4
kcps = 1
itype = 3

klfo lfo kamp, kcps, itype
printk2 klfo
asig oscil 0.7, 440+klfo, 1
outs asig, asig

endin

</CsInstruments>
<CsScore>
;sine wave.
f 1 0 32768 10 1

i 1 0 2 1 ;adds 1 Hz to frequency
i 1 + 2 10 ;adds 10 Hz to frequency
i 1 + 2 220 ;adds 220 Hz to frequency

e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

-, &&, //, \*, /, ^, %

## Crédits

L'arithmétique sur les vecteurs est nouvelle dans le version 6.00

■

- — Opérateur de soustraction.

## Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de conservation de signe, le ET et le OU logiques, l'addition, la soustraction, la multiplication et la division. A noter qu'une valeur ou une expression peut se trouver entre deux de ces opérateurs, pour lesquels elle sera l'argument de gauche ou l'argument de droite comme dans

$a + b * c$ .

Dans de tels cas trois règles s'appliquent :

1.  $*$  et  $/$  s'appliquent à leurs voisins plus fortement que  $+$  et  $-$ . Ainsi l'expression ci-dessus est interprétée comme

$a + (b * c)$

avec  $*$  s'appliquant à  $b$  et à  $c$  et ensuite  $+$  s'appliquant à  $a$  et à  $b * c$ .

2.  $+$  et  $-$  sont prioritaires par rapport à  $\&\&$ , qui est lui-même prioritaire par rapport à  $\|$ :

$a \&\& b - c \| d$

est interprété comme

$(a \&\& (b - c)) \| d$

3. Lorsque deux opérateurs ont le même rang de priorité, les opérations s'enchaînent de gauche à droite :

$a - b - c$

est interprété comme

$(a - b) - c$

On peut utiliser des parenthèses comme ci-dessus pour forcer un groupement particulier.

## Syntaxe

$\#a$  (pas de restriction de taux)

$a \# b$  (pas de restriction de taux)

où les arguments  $a$  and  $b$  peuvent être des expressions.

## Arguments

Les arguments de  $\_$  peuvent être des valeurs scalaires ou des tableaux (vecteurs) unidimensionnels de taux- $k$ , ou n'importe quelle combinaison des deux. Si l'un des arguments est un tableau, la valeur retournée le sera également.

## Exemples

Voici un exemple de l'opérateur `_`. Il utilise le fichier *subtracts.csd* [examples/subtracts.csd].

### Exemple 29. Exemple de l'opérateur `_`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      ;;realtime audio I/O
; For Non-realtime ouput leave only the line below:
; -o -.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

idur = p3
iatt = p4
idec = p5
isus = p3-iatt-idec      ;calculate sustain time from subtracting attack and decay
printf_i "sustain time= note duration - attack - decay --> %.1f-%.1f-%.1f = %.1f\n", 1, idur, iatt, idec

kenv expseg 0.01, iatt, 1, isus, 1, idec, 0.01 ;envelope
asig poscil 1*kenv, 200, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 4096 10 1 ;sine wave
;      attack decay
i 1 0 3 .1      .2
i 1 4 3 .5      1.5
i 1 8 5 4       .5

e
</CsScore>
</CsoundSynthesizer>
```

La sortie contiendra des lignes comme celles-ci :

```
sustain time= note duration - attack - decay --> 3.0-0.1-0.2 = 2.7
sustain time= note duration - attack - decay --> 3.0-0.5-1.5 = 1.0
sustain time= note duration - attack - decay --> 5.0-4.0-0.5 = 0.5
```

## Voir aussi

`+`, `&&`, `//`, `*`, `/`, `^`, `%`

## Crédits

L'arithmétique sur les vecteurs est nouvelle dans la version 6.00

# /

/ — Opérateur de division.

## Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de conservation de signe, le ET et le OU logiques, l'addition, la soustraction, la multiplication et la division. A noter qu'une valeur ou une expression peut se trouver entre deux de ces opérateurs, pour lesquels elle sera l'argument de gauche ou l'argument de droite comme dans

$a + b * c$ .

Dans de tels cas trois règles s'appliquent :

1.  $*$  et  $/$  s'appliquent à leurs voisins plus fortement que  $+$  et  $-$ . Ainsi l'expression ci-dessus est interprétée comme

$a + (b * c)$

avec  $*$  s'appliquant à  $b$  et à  $c$  et ensuite  $+$  s'appliquant à  $a$  et à  $b * c$ .

2.  $+$  et  $-$  sont prioritaires par rapport à  $\&\&$ , qui est lui-même prioritaire par rapport à  $\|$ :

$a \&\& b - c \| d$

est interprété comme

$(a \&\& (b - c)) \| d$

3. Lorsque deux opérateurs ont le même rang de priorité, les opérations s'enchaînent de gauche à droite :

$a - b - c$

est interprété comme

$(a - b) - c$

On peut utiliser des parenthèses comme ci-dessus pour forcer un groupement particulier.

## Syntaxe

`a / b` (pas de restriction de taux)

où les arguments *a* et *b* peuvent être des expressions.

## Arguments

Les arguments de `/` peuvent être des valeurs scalaires ou des tableaux (vecteurs) unidimensionnels de taux-*k*, ou n'importe quelle combinaison des deux. Si l'un des arguments est un tableau, la valeur retournée le sera également.

## Exemples

Voici un exemple de l'opérateur `/`. Il utilise le fichier *divides.csd* [exemples/divides.csd].



### Exemple 30. Exemple de l'opérateur /.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o /.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

idiv = 1/p3 * p4
ktrm oscil 1, idiv, 1      ;use oscil as an envelope
printf "retrigger rate per note duration = %f\n",1, idiv
kndx line 5, p3, 1        ;vary index of FM
asig foscil ktrm, 200, 1, 1.4, kndx, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 4096 10 1 ;sine wave

i 1 0 3 10
i 1 4 3 15
i 1 8 3 2

e
</CsScore>
</CsoundSynthesizer>
```

La sortie contiendra des lignes comme celles-ci :

```
retrigger rate per note duration = 3.333333
retrigger rate per note duration = 5.000000
retrigger rate per note duration = 0.666667
```

## Voir aussi

-, +, &&, //, \*, ^, %

## Crédits

L'arithmétique sur les vecteurs est nouvelle dans la version 6.00

**=**

= — Réalise une simple affectation.

## Syntaxe

```
ares = xarg
ires = iarg
kres = karg
ires, ... = iarg, ...
kres, ... = karg, ...
table [ kval ] = karg
```

## Description

Réalise une simple affectation.

## Initialisation

= (simple affectation) - Met la valeur de l'expression *iarg* (*karg*, *xarg*) dans le résultat nommé. On peut ainsi garder en mémoire le résultat d'une évaluation pour une utilisation ultérieure.

A partir de la version 5.13 les versions de taux-i et de taux-k de l'affectation peuvent prendre un certain nombre de sorties, et un nombre égal ou inférieur d'entrées. S'il y a moins d'entrées, la dernière valeur est répétée le nombre de fois nécessaires.

A partir de la version 5.14 on peut affecter des valeurs aux éléments d'un vecteur en utilisant la notation des crochets.

## Exemples

Voici un exemple de l'opérateur d'affectation. Il utilise le fichier *assign.csd* [examples/assign.csd].

### Exemple 31. Exemple de l'opérateur d'affectation.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime output leave only the line below:
; -o assign.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
```

```

nchnls = 2

instr 1
; Assign a value to the variable i1.
i1 = 1234

; Print the value of the i1 variable.
print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme :

```
instr 1: i1 = 1234.000
```

## Voir aussi

*divz, init, passign, tival*

## Crédits

Exemple écrit par Kevin Conder.

Extension aux valeurs multiples par

Auteur : John ffitch  
 Université de Bath, et Codemist Ltd.  
 Bath, UK  
 Février 2010

Nouveau dans la version 5.13

## **+=**

+= — Addition et affectation.

## Syntaxe

```
ares += xarg  
ires += iarg  
kres += karg  
table [ kval] += karg
```

## Description

Exécute une addition et son affectation.

## Initialisation

+= - Ajoute la valeur de l'expression *iarg* (*karg*, *xarg*) au résultat nommé. On peut ainsi sauvegarder l'évaluation d'un résultat pour un usage ultérieur.

## Exemples

Voici un exemple de l'opcode `plusbecomes`. Il utilise le fichier *reverb.csd* [examples/reverb.csd].

### Exemple 32. Exemple de l'opcode `plusbecomes`.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac    ;;realtime audio out  
;-iadc    ;;uncomment -iadc if realtime audio input is needed too  
; For Non-realtime ouput leave only the line below:  
; -o reverb.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 32  
0dbfs = 1  
nchnls = 2  
  
ga1 init 0  
  
instr 1  
  
asig poscil .2, cpspch(p4), 1  
outs asig, asig  
  
ga1 += asig    ;add direct signal to global reverb
```

```

    endin

    instr 99 ;(highest instr number executed last)

    arev reverb gal, 1.5
        outs arev, arev

    gal = 0 ;clear
    endin

</CsInstruments>
<CsScore>
f 1 0 128 10 1 ;sine

i 1 0 0.1 7.00 ;short sounds
i 1 1 0.1 8.02
i 1 2 0.1 8.04
i 1 3 0.1 8.06

i 99 0 6 ;reverb runs for 6 seconds
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*assign,*

## Crédits

Auteur : John ffitch  
 Université de Bath et Codemist Ltd.  
 Bath, UK  
 2013

Nouveau dans la version 6.00

## ==

== — Teste l'égalité de deux valeurs.

## Description

Teste l'égalité de deux valeurs.

## Syntaxe

```
(a == b ? v1 : v2)
```

où *a*, *b*, *v1* et *v2* peuvent être des expressions, mais *a*, *b* pas de taux audio.

## Exécution

Dans l'expression conditionnelle ci-dessus, *a* et *b* sont d'abord comparés. Si la relation indiquée est vraie (*a* égal à *b*), alors l'expression conditionnelle prend la valeur de *v1* ; si la relation est fausse, l'expression prend la valeur de *v2*. (Par commodité, un seul "=" fonctionnera comme "= =".)

Nota Bene : Si *v1* ou *v2* sont des expressions, elles seront évaluées avant que l'expression conditionnelle ne soit déterminée.

En termes de précedence, tous les opérateurs conditionnels (c-à-d. les opérateurs relationnels (<, etc.), et ?, et : ) sont plus faibles que les opérateurs arithmétiques et logiques (+, -, \*, /, && et //).

Ce sont des *opérateurs* pas des *opcodes*. C'est pourquoi on peut les utiliser dans les instructions de l'orchestre, mais ils ne forment pas des instructions complètes par eux-mêmes.

## Exemples

Voici un exemple de l'opérateur ==. Il utilise le fichier *equals.csd* [examples/equals.csd].

### Exemple 33. Exemple de l'opérateur ==.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o equals.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
```

```

ienv = p4      ;choose envelope in score

if (ienv == 0) kthen
  kenv adsr 0.05, 0.05, 0.95, 0.05 ;sustained envelope
elseif (ienv == 1) kthen
  kenv adsr 0.5, 1, 0.5, 0.5 ;triangular envelope
elseif (ienv == 2) kthen
  kenv adsr 1, 1, 1, 0 ;ramp up
endif

aout vco2 .1, 110, 10
aout = aout * kenv
outs aout, aout

endin

</CsInstruments>
<CsScore>

i1 0 2 0
i1 3 2 1
i1 6 2 2

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

>=, >, <=, <, !=

## ^

^ — Opérateur d'élévation à une puissance.

## Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de conservation de signe, le ET et le OU logiques, l'addition, la soustraction, la multiplication et la division. A noter qu'une valeur ou une expression peut se trouver entre deux de ces opérateurs, pour lesquels elle sera l'argument de gauche ou l'argument de droite comme dans

$a + b * c$ .

Dans de tels cas trois règles s'appliquent :

1.  $*$  et  $/$  s'appliquent à leurs voisins plus fortement que  $+$  et  $-$ . Ainsi l'expression ci-dessus est interprétée comme

$a + (b * c)$

avec  $*$  s'appliquant à  $b$  et à  $c$  et ensuite  $+$  s'appliquant à  $a$  et à  $b * c$ .

2.  $+$  et  $-$  sont prioritaires par rapport à  $\&\&$ , qui est lui-même prioritaire par rapport à  $\|$ :

$a \&\& b - c \| d$

est interprété comme

$(a \&\& (b - c)) \| d$

3. Lorsque deux opérateurs ont le même rang de priorité, les opérations s'enchaînent de gauche à droite :

$a - b - c$

est interprété comme

$(a - b) - c$

On peut utiliser des parenthèses comme ci-dessus pour forcer un groupement particulier.

L'opérateur  $^$  élève  $a$  à la puissance  $b$ .  $b$  ne doit pas être de taux audio. A utiliser avec précaution car les règles de précedence peuvent ne pas fonctionner correctement. Voir *pow*. (Nouveau dans la version 3.493 de Csound.)

## Syntaxe

$a \wedge b$  ( $b$  pas de taux audio)

où les arguments  $a$  et  $b$  peuvent être des expressions.

## Exemples

Voici un exemple de l'opérateur  $^$ . Il utilise le fichier *raises.csd* [examples/raises.csd].



### Exemple 34. Exemple de l'opérateur ^.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o ^.wav   ; output to audio file
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; Lo-Fi sound

kpow = 10      ;exponent
kbase line 1, p3, 2.2 ;vary the base
kQuantize = kbase^kpow
kQuantize = kQuantize*0.5 ;half the number of steps for each side of a bipolar signal
printk2 kQuantize
asig diskin2 "fox.wav", 1, 0, 1 ;loop the fox
asig = round(asig * kQuantize) / kQuantize ;quantize and scale audio signal
outs asig, asig

endin
</CsInstruments>
<CsScore>

i1 0 19.2

e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
i1      0.50000
i1      0.50021
i1      0.50042
.....
i1 1327.49320
i1 1327.74465
```

## Voir aussi

-, +, &&, //, \*, /, %

# ||

|| — Opérateur OU logique.

## Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de signe inchangé, ET logique, OU logique, addition, soustraction, multiplication et division. Notez qu'une valeur ou une expression peut être placée entre deux de ces opérateurs, lesquels peuvent la prendre comme opérande de gauche ou de droite, comme dans

$a + b * c$ .

Trois règles s'appliquent dans de tels cas :

1.  $*$  et  $/$  s'appliquent à leurs voisins plus fortement que  $+$  et  $-$ . Ainsi l'expression ci-dessus s'interprète comme

$a + (b * c)$

avec  $*$  prenant  $b$  et  $c$  puis  $+$  prenant  $a$  et  $b * c$ .

2.  $+$  et  $-$  sont prioritaires sur  $\&\&$ , qui devance lui-même  $||$  :

$a \&\& b - c || d$

est interprété comme

$(a \&\& (b - c)) || d$

3. Quand deux opérateurs sont d'égale importance, les opérations ont lieu de gauche à droite :

$a - b - c$

est interprété comme

$(a - b) - c$

On peut utiliser des parenthèses pour forcer un groupement particulier.

## Syntaxe

`a || b` (OU logique ; pas de taux audio)

où les arguments *a* et *b* peuvent être des expressions.

## Exemples

Voici un exemple de l'opérateur `|`. Il utilise le fichier *logicOR.csd* [exemples/logicOR.csd].

### Exemple 35. Exemple de l'opérateur `|`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0    ;;realtime audio out and virtual midi keyboard
-iadc    ;;uncomment -iadc if realtime audio input is needed too
</CsOptions>
<CsInstruments>
;after a UDO from Rory Walsh
sr = 44100
ksmps = 32
nchnls = 2

instr 1 ;displays notes, midi channel and control number information

kstatus, kchan, kdata1, kdata2 midiin
k1 changed kstatus
k2 changed kchan
k3 changed kdata1
k4 changed kdata2
if((k1==1)|| (k2==1)|| (k3==1)|| (k4==1)) then
printks "Value:%d ChanNo:%d CtrlNo:%d\n" , 0, kdata2, kchan, kdata1
endif

endin
</CsInstruments>
<CsScore>

i1 0 60 ;print values for 60 seconds

e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

Value:127 ChanNo:11 CtrlNo:62
Value:127 ChanNo:11 CtrlNo:60
Value:127 ChanNo:11 CtrlNo:60
Value:127 ChanNo:11 CtrlNo:60
Value:127 ChanNo:11 CtrlNo:60
....
Value:0 ChanNo:11 CtrlNo:62
Value:0 ChanNo:11 CtrlNo:62
Value:0 ChanNo:11 CtrlNo:62
Value:0 ChanNo:11 CtrlNo:62
....
Value:77 ChanNo:11 CtrlNo:23
Value:77 ChanNo:11 CtrlNo:23
Value:76 ChanNo:11 CtrlNo:23
Value:76 ChanNo:11 CtrlNo:23
....

```

## Voir aussi

-, +, &&, \*, /, ^, %

# !

! — Opérateur NON logique.

## Description

Les opérateurs arithmétiques réalisent les opérations de changement de signe (négation), de signe inchangé, ET logique, OU logique, addition, soustraction, multiplication et division. Notez qu'une valeur ou une expression peut être placée entre deux de ces opérateurs, lesquels peuvent la prendre comme opérande de gauche ou de droite, comme dans

$a + b * c$ .

Trois règles s'appliquent dans de tels cas :

1.  $*$  et  $/$  s'appliquent à leurs voisins plus fortement que  $+$  et  $-$ . Ainsi l'expression ci-dessus s'interprète comme

$a + (b * c)$

avec  $*$  prenant  $b$  et  $c$  puis  $+$  prenant  $a$  et  $b * c$ .

2.  $+$  et  $-$  sont prioritaires sur  $\&\&$ , qui devance lui-même  $\parallel$  :

$a \&\& b - c \parallel d$

est interprété comme

$(a \&\& (b - c)) \parallel d$

3. Quand deux opérateurs sont d'égale importance, les opérations ont lieu de gauche à droite :

$a - b - c$

est interprété comme

$(a - b) - c$

On peut utiliser des parenthèses pour forcer un groupement particulier.

## Syntaxe

`! a (NON logique ; pas de taux audio)`

où l'argument *a* peuvent être une expression.

## Exemples

Voici un exemple de l'opérateur NON logique. Il utilise le fichier *opnot.csd* [examples/opnot.csd].

### Exemple 36. Exemple de l'opcode opnot.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o opand.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kval    randomh 0, 1.2, 20 ;choose between 0 and 1.2
if !(kval >0 && kval<=.5) then ;3 possible outcomes
    kval = 1
elseif !(kval >.5 && kval<=1) then
    kval =2
elseif !(kval >1) then
    kval =3
endif

printk2 kval ;print value when it changes
asig    poscil .7, 440*kval, 1
        outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 16384 10 1

i 1 0 5
e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

i1      1.00000
i1      2.00000
i1      1.00000
i1      2.00000
i1      1.00000
i1      2.00000
i1      1.00000
.....

```

## Voir aussi

-, +, //, &&, \*, /, ^, %

# Odbfs

Odbfs — Fixe la valeur des 0 décibels à amplitude maximale.

## Description

Fixe la valeur des 0 décibels à amplitude maximale.

## Syntaxe

```
Odbfs = iarg
```

```
Odbfs
```

## Initialisation

*iarg* -- la valeur des 0 décibels à amplitude maximale.

## Exécution

La valeur par défaut est 32767, si bien que tous les orchestres existants *devraient* fonctionner.

Les valeurs d'amplitude dans Csound sont toujours relatives à une valeur "*Odbfs*" représentant l'amplitude maximale possible sans écrêtage. A l'origine cette valeur valait toujours 32767 dans Csound, correspondant à l'étendue bipolaire d'un fichier son sur 16 bit ou d'un codec AN/NA sur 16 bit. Cela reste l'amplitude maximale *par défaut* dans Csound, pour des raisons de compatibilité descendante. La valeur *Odbfs* permet à Csound de produire des valeurs mises à l'échelle de n'importe quel format de sortie, que ce soit en entiers sur 16 bit ou 24 bit, en flottants sur 32 bit, et même en entiers sur 32 bit.

On peut définir Odbfs dans l'en-tête, pour fixer l'amplitude de référence utilisée par Csound, mais on peut aussi l'utiliser comme variable dans un instrument comme ceci :

```
ipeak = odbfs
```

```
asig oscil odbfs, freq, 1  
out  asig * 0.3 * odbfs
```

L'opcode *Odbfs* a pour but le codage relatif à une valeur Odbfs (et l'usage beaucoup plus fréquent des opcodes *ampdbfs()* !), plutôt que l'utilisation de valeurs d'échantillon explicites. L'utilisation de Odbfs=1 est conforme aux pratiques de l'industrie, car l'intervalle allant de -1 à 1 est utilisé dans la plupart des formats de greffon commerciaux et dans la plupart des autres systèmes de synthèse comme Pure Data.

Les nombres en virgule flottante écrits dans un fichier, lorsque *Odbfs* = 1, ne seront effectivement pas transposés du tout en amplitude. Ainsi les nombres dans le fichier sont exactement ce que l'orchestre dit qu'ils sont.

Pour plus de détails sur les valeurs d'amplitude dans Csound, voir la section *Valeurs d'amplitude dans Csound*.

## Exemples

Voici un exemple de l'opcode `Odbfs`. Il utilise le fichier `Odbfs.csd` [examples/Odbfs.csd].

### Exemple 37. Exemple de l'opcode `Odbfs`.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o Odbfs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 10
nchnls = 2

; Set the Odbfs to 1.
Odbfs = 1

instr 1 ; from linear amplitude (0-1 range)
print p4
a1 oscil p4, 440, 1
outs a1, a1
endin

instr 2 ; from linear amplitude (0-32767 range)
iamp = p4 / 32767
print iamp
a1 oscil iamp, 440, 1
outs a1, a1
endin

instr 3 ; from dB FS
iamp = ampdbfs(p4)
print iamp
a1 oscil iamp, 440, 1
outs a1, a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

i 1 0 1 1
i 1 + 1 0.5
i 1 + 1 0.1
s
i 2 0 1 32767
i 2 + 1 [32767/2]
i 2 + 1 [3276.7]
s
i 3 0 1 0
i 3 + 1 -6
```

```
i 3 + 1 -20  
e  
  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*ampdbfs()*

## Crédits

Auteur : Richard Dobson  
Mai 2002

Nouveau dans la version 4.10



# A4

A4 — Fixe la fréquence de base pour le la du diapason.

## Description

Ces instructions sont des *affectations* de valeurs globales réalisées au début d'un orchestre, avant que tout bloc d'instrument ne soit défini. Leur fonction est de fixer certaines *variables* dont le nom est un mot réservé et qui sont nécessaires à l'exécution. Une fois fixés, ces mots réservés peuvent être utilisés dans des expressions n'importe où dans l'orchestre.

## Syntaxe

**A4** = iarg

## Initialisation

A4 = (facultatif) -- fixe la fréquence de référence pour la hauteur A4 (la du diapason) à *iarg* Hz. La valeur par défaut est 440.

De plus, toute *variable globale* peut être initialisée par une *instruction de la période d'initialisation* n'importe où avant la première *instruction instr.* Toutes les affectations ci-dessus sont exécutées dans l'instrument 0 (passe-i seulement) au début de l'exécution réelle.

A partir de la version 6.08 de Csound version 6.08, A4 peut être utilisée. Elle affecte le comportement des opcodes *cpspch*, *cpsoct*, *cps2pch*, *cpsxpch* et *cpsmidinn*.

## Voir aussi

*sr*, *kr*, *ksmps*, *nchnls*, *nchnls\_i*, *0dbfs*, *cpspch*, *cpsoct*, *cpsmidinn*, *cps2pch*, *cpsxpch*.

## Crédits

Auteur : John ffitich  
Septembre 2016

Nouveau dans la version 6.08 de Csound.



<< — Opérateur de décalage binaire à gauche.

## Description

Les opérateurs de décalage binaire décalent les bit vers la gauche ou vers la droite du nombre de bit donné.

La précédence de ces opérateurs est moins élevée que celle des opérateurs arithmétiques, mais plus élevées que celle des comparaisons.

On peut utiliser des parenthèses comme ci-dessus pour forcer des groupements particuliers.

## Syntaxe

`a << b` (décalage binaire à gauche)

où les arguments *a* et *b* peuvent être des expressions.

## Exemples

Voici un exemple de l'opcode bitshift. Il utilise le fichier *bitshift.csd* [examples/bitshift.csd].

### Exemple 38. Exemple de l'opcode bitshift à gauche.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
;-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o bitshift.wav -W --nosound ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2

instr 1 ;bit shift right
ival = p4>>p5
printf_i "%i>>%i = %i\n", 1, p4, p5, ival
endin

instr 2 ;bit shift left
ival = p4<<p5
printf_i "%i<<%i = %i\n", 1, p4, p5, ival
endin

</CsInstruments>
<CsScore>
i 1 0 0.1 2 1
i 1 + . 3 1
i 1 + . 7 2
```

```

i 1 + . 16 1
i 1 + . 16 2
i 1 + . 16 3

i 2 5 0.1 1 1
i 2 + . 1 2
i 2 + . 1 3
i 2 + . 1 4
i 2 + . 2 1
i 2 + . 2 2
i 2 + . 2 3
i 2 + . 3 2
e
</CsScore>
</CsoundSynthesizer>

```

L'exemple ci-dessus produira la sortie suivante :

```

2>>1 = 1
B 0.000 .. 0.100 T 0.100 TT 0.100 M: 0.0 0.0
3>>1 = 1
B 0.100 .. 0.200 T 0.200 TT 0.200 M: 0.0 0.0
7>>2 = 1
B 0.200 .. 0.300 T 0.300 TT 0.300 M: 0.0 0.0
16>>1 = 8
B 0.300 .. 0.400 T 0.400 TT 0.400 M: 0.0 0.0
16>>2 = 4
B 0.400 .. 0.500 T 0.500 TT 0.500 M: 0.0 0.0
16>>3 = 2
B 0.500 .. 5.000 T 5.000 TT 5.000 M: 0.0 0.0
new alloc for instr 2:
1<<1 = 2
B 5.000 .. 5.100 T 5.100 TT 5.100 M: 0.0 0.0
1<<2 = 4
B 5.100 .. 5.200 T 5.200 TT 5.200 M: 0.0 0.0
1<<3 = 8
B 5.200 .. 5.300 T 5.300 TT 5.300 M: 0.0 0.0
1<<4 = 16
B 5.300 .. 5.400 T 5.400 TT 5.400 M: 0.0 0.0
2<<1 = 4
B 5.400 .. 5.500 T 5.500 TT 5.500 M: 0.0 0.0
2<<2 = 8
B 5.500 .. 5.600 T 5.600 TT 5.600 M: 0.0 0.0
2<<3 = 16
B 5.600 .. 5.700 T 5.700 TT 5.700 M: 0.0 0.0
3<<2 = 12

```

## Voir aussi

>>, &, / #

**>>**

>> — Opérateur de décalage binaire à droite.

## Description

Les opérateurs de décalage binaire décalent les bit vers la gauche ou vers la droite du nombre de bit donné.

La précedence de ces opérateurs est moins élevée que celle des opérateurs arithmétiques, mais plus élevées que celle des comparaisons.

On peut utiliser des parenthèses comme ci-dessus pour forcer des groupements particuliers.

## Syntaxe

`a >> b` (décalage binaire à droite)

où les arguments *a* et *b* peuvent être des expressions.

## Exemples

Voir l'entrée de l'opérateur << pour un exemple.

## Voir aussi

<<, &, / #

# &

& — Opérateur ET binaire.

## Description

Les opérateurs binaires effectuent le ET binaire, le OU binaire, la négation binaire et la non-équivalence binaire.

## Syntaxe

`a & b` (ET binaire)

où les arguments *a* et *b* peuvent être des expressions. Ils sont convertis à la valeur entière la plus proche selon la précision de la machine et l'opération est ensuite effectuée.

## Exécution

La priorité de ces opérateurs est inférieure à celle des opérateurs arithmétiques, mais supérieure à celle des comparaisons.

On peut utiliser des parenthèses pour forcer des groupements particuliers.

## Exemples

Voici un exemple des opérateurs binaires ET et OU. Il utilise le fichier *bitwise.csd* [examples/bitwise.csd].

### Exemple 39. Exemple des opérateurs binaires.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1
  irestOr = p4 | p5
  irestAnd = p4 & p5
  prints "%i | %i = %i\\n", p4, p5, irestOr
  prints "%i & %i = %i\\n", p4, p5, irestAnd
endin

instr 2 ; decimal to binary converter
  Sbinary = ""
  inumbits = 8
  icount init inumbits - 1

pass:
```

```

    ivalue = 2 ^ icount
    if (p4 & ivalue >= ivalue) then
        Sdigit = "1"
    else
        Sdigit = "0"
    endif
    Sbinary strcat Sbinary, Sdigit

loop_ge icount, 1, 0, pass

Stext sprintf "%i is %s in binary\\n", p4, Sbinary
prints Stext
endin

</CsInstruments>
<CsScore>
i 1 0 0.1 1 2
i 1 + . 1 3
i 1 + . 2 4
i 1 + . 3 10

i 2 2 0.1 12
i 2 + . 9
i 2 + . 15
i 2 + . 49

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

/, #, ▯

**|**

| — Opérateur OU binaire.

## Description

Les opérateurs binaires effectuent le ET binaire, le OU binaire, la négation binaire et la non-équivalence binaire.

## Syntaxe

$a \mid b$  (OU binaire)

où les arguments  $a$  et  $b$  peuvent être des expressions. Ils sont convertis à la valeur entière la plus proche selon la précision de la machine et l'opération est ensuite effectuée.

## Exécution

La priorité de ces opérateurs est inférieure à celle des opérateurs arithmétiques, mais supérieure à celle des comparaisons.

On peut utiliser des parenthèses pour forcer des groupements particuliers.

Pour un exemple d'utilisation, voir l'entrée pour  $\&$

## Voir aussi

$\&$ ,  $\#$ ,  $\neg$



$\neg$  — Opérateur NON binaire.

## Description

Les opérateurs binaires effectuent le ET binaire, le OU binaire, la négation binaire et la non-équivalence binaire.

La priorité de ces opérateurs est inférieure à celle des opérateurs arithmétiques, mais supérieure à celle des comparaisons.

On peut utiliser des parenthèses pour forcer des groupements particuliers.

## Syntaxe

$\sim a$  (NON binaire)

où l'argument  $a$  peut être une expression. Il est converti dans la valeur entière la plus proche selon la précision de la machine et l'opération est ensuite effectuée.

## Voir aussi

$\&$ ,  $/$   $\#$



# #

# — Opérateur NON-EQUIVALENCE binaire.

## Description

Les opérateurs binaires effectuent le ET binaire, le OU binaire, la négation binaire et la non-équivalence binaire.

La priorité de ces opérateurs est inférieure à celle des opérateurs arithmétiques, mais supérieure à celle des comparaisons.

On peut utiliser des parenthèses pour forcer des groupements particuliers.

## Syntaxe

`a # b` (NON-EQUIVALENCE binaire)

où les arguments *a* et *b* peuvent être des expressions. Ils sont convertis dans la valeur entière la plus proche selon la précision de la machine et l'opération est ensuite effectuée.

## Voir aussi

`&`, `/` `¬`

# a

a — Convertit un paramètre de taux-k en une valeur de taux-a avec interpolation.

## Description

Convertit un paramètre de taux-k en une valeur de taux-a avec interpolation.

## Syntaxe

**a**(x) (arguments de taux-k seulement)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode a. Il utilise le fichier *opa.csd* [examples/opa.csd].

### Exemple 40. Exemple de l'opcode a.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o a.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; hear the difference between instr.1 and 2
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;sine wave at k-rate

ksig oscil 0.8, 440, 1
; k-rate to the audio-rate conversion
asig = a(ksig)
outs asig, asig

endin

instr 2 ;sine wave at a-rate

asig oscil 0.8, 440, 1
outs asig, asig

endin
```

```
</CsInstruments>
<CsScore>
; sine wave.
f 1 0 16384 10 1

i 1 0 2
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*i, k*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.21

# abs

abs — Retourne une valeur absolue.

## Description

Retourne la valeur absolue de  $x$ .

## Syntaxe

**abs**( $x$ ) (pas de restriction de taux)

**abs**( $k/i[]$ ) ( $k$ - ou  $i$ -tableau)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode abs. Il utilise le fichier *abs.csd* [examples/abs.csd].

### Exemple 41. Exemple de l'opcode abs.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1
ix = p4
iabs = abs(ix)
print iabs

endin

</CsInstruments>
<CsScore>

i 1 0 1 0
i 1 + 1 -.15
i 1 + 1 -13
e

</CsScore>
```

```
</CsoundSynthesizer>
```

Sa sortie inclura des lignes comme :

```
instr 1: iabs = 0.000  
instr 1: iabs = 0.150  
instr 1: iabs = 13.000
```

## Voir aussi

*exp, frac, int, log, log10, i, sqrt*

# active

`active` — Retourne le nombre d'instances actives d'un instrument.

## Description

Retourne le nombre d'instances actives d'un instrument, avec une option pour ignorer les instances en phase de relâchement.

## Syntaxe

```
ir active insnum [,iopt [,inorel]]  
ir active Sinsname [,iopt [,inorel]]  
kres active kinsnum [,iopt [,inorel]]
```

## Initialisation

*insnum* -- numéro ou nom de l'instrument concerné

*Sinsname* -- nom de l'instrument

*iopt* (facultatif, 0 par défaut) -- par défaut, l'opcode retourne le nombre d'instances couramment actives. Si ce paramètre est différent de zéro, l'opcode retourne le nombre d'instances activées depuis le début de l'exécution.

*inorel* -- s'il est différent de zéro, les instruments en phase de relâchement sont ignorés (vaut zéro par défaut). N'est valide que si *iopts* vaut zéro.

## Exécution

*kinsnum* -- numéro ou nom de l'instrument concerné

*active* retourne le nombre d'instances actives de l'instrument numéro *insnum/kinsnum* (ou de l'instrument nommé *Sinsname*). A partir de Csound 4.17 la sortie est mise à jour au taux-k (si l'argument d'entrée est de taux-k), pour permettre un comptage dynamique des instances de l'instrument.

A partir de Csound 5.17 si le numéro de l'instrument passé vaut zéro, tous les instruments sont comptés.

## Exemples

Voici un exemple simple de l'opcode *active*. Il utilise le fichier *active.csd* [examples/active.csd].

### Exemple 42. Exemple simple de l'opcode *active*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
; Audio out  Audio in  
-odac          -iadc      ;;RT audio I/O
```

```

; For Non-realtime ouput leave only the line below:
; -o active.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a noisy waveform.
instr 1
; Generate a really noisy waveform.
anoisy rand 44100
; Turn down its amplitude.
aoutput gain anoisy, 2500
; Send it to the output.
out aoutput
endin

; Instrument #2 - counts active instruments.
instr 2
; Count the active instances of Instrument #1.
icount active 1
; Print the number of active instances.
print icount
endin

</CsInstruments>
<CsScore>

; Start the first instance of Instrument #1 at 0:00 seconds.
i 1 0.0 3.0

; Start the second instance of Instrument #1 at 0:015 seconds.
i 1 1.5 1.5

; Play Instrument #2 at 0:01 seconds, when we have only
; one active instance of Instrument #1.
i 2 1.0 0.1

; Play Instrument #2 at 0:02 seconds, when we have
; two active instances of Instrument #1.
i 2 2.0 0.1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme :

```

instr 2: icount = 1.000
instr 2: icount = 2.000

```

Voici un exemple plus avancé de l'opcode active. Il affiche le résultat de l'opcode active au taux-k. Il utilise le fichier *active\_k.csd* [examples/active\_k.csd].

### Exemple 43. Exemple de l'opcode active au taux-k.

```

<CsoundSynthesizer>
<CsOptions>

```

```

; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o active_k.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a noisy waveform.
instr 1
; Generate a really noisy waveform.
anoisy rand 44100
; Turn down its amplitude.
aoutput gain anoisy, 2500
; Send it to the output.
out aoutput
endin

; Instrument #2 - counts active instruments at k-rate.
instr 2
; Count the active instances of Instrument #1.
kcount active 1
; Print the number of active instances.
printk2 kcount
endin

</CsInstruments>
<CsScore>

; Start the first instance of Instrument #1 at 0:00 seconds.
i 1 0.0 3.0

; Start the second instance of Instrument #1 at 0:015 seconds.
i 1 1.5 1.5

; Play Instrument #2 at 0:01 seconds, when we have only
; one active instance of Instrument #1.
i 2 1.0 0.1

; Play Instrument #2 at 0:02 seconds, when we have
; two active instances of Instrument #1.
i 2 2.0 0.1
e

</CsScore>
</CsSoundSynthesizer>

```

Sa sortie contiendra des lignes comme :

```

i2      1.00000
i2      2.00000

```

Voici un autre exemple de l'opcode active, qui utilise le nombre d'instances pour calculer le gain. Il utilise le fichier *active\_scale.csd* [examples/active\_scale.csd].

#### Exemple 44. Exemple de l'opcode active au taux-k.



```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o atone.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr= 44100
ksmps = 64
nchnls = 1
0dbfs = 1

;by Victor Lazzarini 2008

instr 1
kscal active 1
kamp port 1/kscal, 0.01
asig oscili kamp, p4, 1
kenv linseg 0, 0.1,1,p3-0.2,1,0.1, 0

      out asig*kenv
endin

</CsInstruments>
<CsScore>
f1 0 16384 10 1

i1 0 10 440
i1 1 3 220
i1 2 5 350
i1 4 3 700
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch  
 University of Bath/Codemist Ltd.  
 Bath, UK  
 Juillet 1999

Exemples écrits par Kevin Conder.

Nouveau dans la version 3.57 de Csound ; instruments nommés ajoutés dans la version 5.13

Nouvelle option pour toutes les instances activées dans la version 5.13

Le compte de tous les instruments est nouveau dans la version 5.17

Option *no release* ajoutée dans la version 5.19

## adsr

`adsr` — Calcule l'enveloppe ADSR classique à l'aide de segments linéaires.

## Description

Calcule l'enveloppe ADSR classique à l'aide de segments linéaires.

## Syntaxe

```
ares adsr iatt, idec, islev, irel [, idel]
```

```
kres adsr iatt, idec, islev, irel [, idel]
```

## Initialisation

*iatt* -- durée de l'attaque (attack)

*idec* -- durée de la première chute (decay)

*islev* -- niveau d'entretien (sustain)

*irel* -- durée de la chute (release)

*idel* -- délai de niveau zéro avant le démarrage de l'enveloppe

## Exécution

L'enveloppe générée évolue dans l'intervalle de 0 à 1 et peut nécessiter un changement d'échelle par la suite, en fonction de l'amplitude demandée. Si l'on utilise *Odbfs* = 1, il sera probablement nécessaire de diminuer l'amplitude de l'enveloppe car plusieurs notes simultanées peuvent provoquer un écrêtage. Si l'on utilise pas *Odbfs*, une mise à l'échelle à une grande amplitude (par exemple 32000) sera peut-être nécessaire.

Voici une description de l'enveloppe :

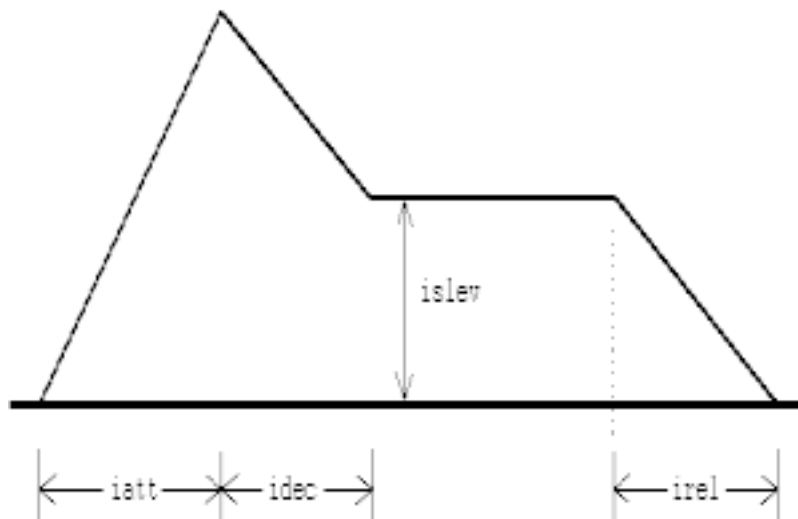


Image d'une enveloppe ADSR.

La longueur de la période d'entretien est calculée à partir de la longueur de la note. C'est pourquoi *adsr* n'est pas adapté au traitement des événements MIDI. L'opcode *madsr* utilise le mécanisme de *linsegr*, et peut donc être utilisé dans les applications MIDI.

*adsr* est nouveau dans la version 3.49 de Csound.

## Exemples

Voici un exemple de l'opcode *adsr*. Il utilise le fichier *adsr.csd* [examples/adsr.csd].

### Exemple 45. Exemple de l'opcode *adsr*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o adsr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

iatt = p5
idec = p6
islev = p7
irel = p8

kenv adsr iatt, idec, islev, irel
kcps = cpspch(p4) ;frequency

asig vco2 kenv * 0.8, kcps
outs asig, asig

endin

</CsInstruments>
<CsScore>

i 1 0 2 7.00 .0001 1 .5 .001 ; short attack
i 1 3 2 7.02 1 .5 .5 .001 ; long attack
i 1 6 2 6.09 .0001 1 .5 .7 ; long release

e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*madsr*, *mxadsr*, *xadsr*

## Crédits

Auteur : John ffitch

Nouveau dans la version 3.49

# adsyn

**adsyn** — La sortie est la somme d'un ensemble de sinusoïdes contrôlées individuellement, jouées par un banc d'oscillateurs.

## Description

La sortie est la somme d'un ensemble de sinusoïdes contrôlées individuellement, jouées par un banc d'oscillateurs.

## Syntaxe

```
ares adsyn kamod, kfmod, ksmod, ifilcod
```

## Initialisation

*ifilcod* -- entier ou chaîne de caractères dénotant un fichier de contrôle issu de l'analyse d'un signal audio. Un entier dénote le suffixe d'un fichier *adsyn.m* ou *pvoc.m* ; une chaîne de caractères (entre doubles apostrophes) donne un nom de fichier, optionnellement un nom de chemin complet. S'il ne s'agit pas d'un chemin complet, le fichier est d'abord recherché dans le répertoire courant, puis dans celui qui est indiqué par la variable d'environnement *SADIR* (si elle est définie). Le fichier de contrôle *adsyn* contient les valeurs des points charnière des enveloppes d'amplitude et de fréquence, tandis que le fichier de contrôle *pvoc* contient des données similaires organisées pour une resynthèse par tfr. L'utilisation de la mémoire dépend de la taille des fichiers impliqués, qui sont lus et maintenus entièrement en mémoire durant le calcul tout en étant partagés par les appels multiples (voir aussi *lpread*).

## Exécution

*kamod* -- facteur d'amplitude des partiels additionnés.

*kfmod* -- facteur de fréquence des partiels additionnés. C'est un facteur de transposition au taux de contrôle : une valeur de 1 signifie pas de transposition, 1,5 transpose d'un quinte juste ascendante, et 0,5 d'une octave descendante.

*ksmod* -- facteur de vitesse des partiels additionnés.

*adsyn* synthétise des timbres dynamiques complexes par la méthode de synthèse additive. N'importe quel nombre de sinusoïdes, contrôlées individuellement en fréquence et en amplitude, peuvent être additionnées par une unité arithmétique très rapide pour produire un résultat de grande qualité.

Les composantes sinusoïdales sont décrites dans un fichier de contrôle qui contient des pistes d'amplitude et de fréquence définies par des points charnière. Les pistes sont des séquences de nombres entiers sur 16 bit :

-1, date, amp, date, amp,...  
-2, date, fréq, date, fréq,...

telles que celles qui sont produites par l'analyse d'un fichier audio au moyen d'un filtre hétérodyne. (Pour des détails, voir *hetro*.) Les valeurs instantanées d'amplitude et de fréquence sont utilisées par un oscillateur interne en virgule fixe qui additionne chaque partiel actif dans un signal de sortie accumulé. Bien qu'il y

ait une limite pratique (limite supprimée dans la version 3.47) du nombre de partiels mis à contribution, il n'y a aucune restriction quant à leur comportement dans le temps. Un son quelconque que l'on peut décrire en termes d'évolution de sinusoides sera synthétisable par *adsyn* seul.

On peut aussi modifier un son décrit par un fichier de contrôle *adsyn* pendant la resynthèse. Les signaux *kamod*, *kfmod* et *ksmod* modifieront l'amplitude, la fréquence et la vitesse des partiels. Ce sont des facteurs multiplicatifs, avec *kfmod* modifiant la fréquence et *ksmod* modifiant la *vitesse* avec laquelle les segments en millisecondes définis par les points charnière sont parcourus. Ainsi, 0,7, 1,5 et 2 produiront un son plus doux, plus haut d'une quinte juste, mais deux fois moins long. Les valeurs 1, 1, 1 laisseront le son inchangé. Chacune de ces entrées peut être un signal de contrôle.

## Exemples

Voici un exemple de l'opcode *adsyn*. Il utilise les fichiers *adsyn.csd* [examples/adsyn.csd] et *kickroll.het* [examples/kickroll.het]. Le fichier « kickroll.het » a été créé en utilisant l'utilitaire *hetro* avec le fichier audio *kickroll.wav* [examples/kickroll.wav].

### Exemple 46. Exemple de l'opcode *adsyn*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o adsyn.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
; analyze the file "kickroll.wav" first
kamod = 1
kfmod = p4
ksmod = p5

asig adsyn kamod, kfmod, ksmod, "kickroll.het"
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 4 1 .2
i 1 + 1 2 1
i 1 + 1 .3 1.5
e

</CsScore>
</CsoundSynthesizer>
```

# adsynt

adsynt — Réalise une synthèse additive avec un nombre arbitraire de partiels, pas nécessairement harmoniques.

## Description

Réalise une synthèse additive avec un nombre arbitraire de partiels, pas nécessairement harmoniques.

## Syntaxe

```
ares adsynt kamp, kcps, iwfn, ifreqfn, iampfn, icnt [, iphs]
```

## Initialisation

*iwfn* -- table contenant une forme d'onde, normalement une sinus. Les valeurs de la table ne sont pas interpolées pour des raisons de performance, si bien que des tables plus grandes apportent une meilleure qualité.

*ifreqfn* -- table contenant les valeurs de fréquence de chaque partiel. *ifreqfn* peut contenir les valeurs de fréquence initiales de chaque partiel, mais elle est habituellement utilisée pour générer des paramètres pendant l'exécution avec *tablew*. Les fréquences doivent être relatives à *kcps*. La taille doit être au moins égale à *icnt*.

*iampfn* -- table contenant les valeurs d'amplitude de chaque partiel. *iampfn* peut contenir les valeurs d'amplitude initiales de chaque partiel, mais elle est habituellement utilisée pour générer des paramètres pendant l'exécution avec *tablew*. Les amplitudes doivent être relatives à *kamp*. La taille doit être au moins égale à *icnt*.

*icnt* -- nombre de partiels à générer.

*iphs* -- phase initiale de chaque oscillateur, si *iphs* = -1, l'initialisation est ignorée. Si *iphs* > 1, toutes les phases seront initialisées avec une valeur aléatoire.

## Exécution

*kamp* -- amplitude de la note.

*kcps* -- fréquence de base de la note. Les fréquences des partiels seront relatives à *kcps*.

La fréquence et l'amplitude de chaque partiel sont données dans les deux tables fournies. Le but de cet opcode est de faire générer par un instrument les paramètres de synthèse au taux-k et de les écrire dans des tables globales avec l'opcode *tablew*.

## Exemples

Voici un exemple de l'opcode *adsynt*. Il utilise le fichier *adsynt.csd* [examples/adsynt.csd]. Ces deux instruments réalisent une synthèse additive. La sortie de chacun d'entre eux sonne comme un bol tibétain. Le premier est statique, car ses paramètres ne sont générés que pendant l'initialisation. Dans le second, les paramètres changent de façon continue.

## Exemple 47. Exemple de l'opcode adsynt.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o adsynt.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
; Generate a sinewave table.
giwave ftgen 1, 0, 1024, 10, 1
; Generate two empty tables for adsynt.
gifrqs ftgen 2, 0, 32, 7, 0, 32, 0
; A table for frequency and amp parameters.
giamps ftgen 3, 0, 32, 7, 0, 32, 0

; Generates parameters at init time
instr 1
; Generate 10 voices.
icnt = 10
; Init loop index.
index = 0

; Loop only executed at init time.
loop:
; Define non-harmonic partials.
ifreq pow index + 1, 1.5
; Define amplitudes.
iamp = 1 / (index+1)
; Write to tables.
tableiw ifreq, index, gifrqs
; Used by adsynt.
tableiw iamp, index, giamps

index = index + 1
; Do loop/
if (index < icnt) igoto loop

asig adsynt 0.3, 150, giwave, gifrqs, giamps, icnt
outs asig, asig
endin

; Generates parameters every k-cycle.
instr 2
; Generate 10 voices.
icnt = 10
; Reset loop index.
kindex = 0

; Loop executed every k-cycle.
loop:
; Generate lfo for frequencies.
kspeed pow kindex + 1, 1.6
; Individual phase for each voice.
kphas phasorbnk kspeed * 0.7, kindex, icnt
klfo table kphas, giwave, 1
```



```

; Arbitrary parameter twiddling...
kdepth pow 1.4, kindex
kfreq pow kindex + 1, 1.5
kfreq = kfreq + klfo*0.006*kdepth

; Write freqs to table for adsynt.
tablew kfreq, kindex, gifrqs

; Generate lfo for amplitudes.
kspeed pow kindex + 1, 0.8
; Individual phase for each voice.
kphas phasorbnk kspeed*0.13, kindex, icnt, 2
klfo table kphas, giwave, 1
; Arbitrary parameter twiddling...
kamp pow 1 / (kindex + 1), 0.4
kamp = kamp * (0.3+0.35*(klfo+1))

; Write amps to table for adsynt.
tablew kamp, kindex, giamps

kindex = kindex + 1
; Do loop.
if (kindex < icnt) kgoto loop

asig adsynt 0.25, 150, giwave, gifrqs, giamps, icnt
outs asig, asig
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 2.5 seconds.
i 1 0 2.5
; Play Instrument #2 for 2.5 seconds.
i 2 3 2.5
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Peter Neubäcker  
Munich, Allemagne  
Août 1999

Nouveau dans la version 3.58 de Csound

# adsynt2

adsynt2 — Réalise une synthèse additive avec un nombre arbitraire de partiels - pas nécessairement harmoniques - avec interpolation.

## Description

Réalise une synthèse additive avec un nombre arbitraire de partiels, pas nécessairement harmoniques. (Voir *adsynt*)

## Syntaxe

```
ar adsynt2 kamp, kcps, iwfn, ifreqfn, iampfn, icnt [, iphs]
```

## Initialisation

*iwfn* -- table contenant une forme d'onde, normalement une sinus. Les valeurs de la table ne sont pas interpolées pour des raisons de performance, si bien que des tables plus grandes apportent une meilleure qualité.

*ifreqfn* -- table contenant les valeurs de fréquence de chaque partiel. *ifreqfn* peut contenir les valeurs de fréquence initiales de chaque partiel, mais elle est habituellement utilisée pour générer des paramètres pendant l'exécution avec *tablew*. Les fréquences doivent être relatives à *kcps*. La taille doit être au moins égale à *icnt*.

*iampfn* -- table contenant les valeurs d'amplitude de chaque partiel. *iampfn* peut contenir les valeurs d'amplitude initiales de chaque partiel, mais elle est habituellement utilisée pour générer des paramètres pendant l'exécution avec *tablew*. Les amplitudes doivent être relatives à *kamp*. La taille doit être au moins égale à *icnt*.

*icnt* -- nombre de partiels à générer.

*iphs* -- phase initiale de chaque oscillateur, si *iphs* = -1, l'initialisation est ignorée. Si *iphs* > 1, toutes les phases seront initialisées avec une valeur aléatoire.

## Exécution

*kamp* -- amplitude de la note.

*kcps* -- fréquence de base de la note. Les fréquences des partiels seront relatives à *kcps*.

La fréquence et l'amplitude de chaque partiel sont données dans les deux tables fournies. Le but de cet opcode est de faire générer par un instrument les paramètres de synthèse au taux-k et de les écrire dans des tables globales avec l'opcode *tablew*.

*adsynt2* est identique à *adsynt* (by Peter Neubäcker), sauf qu'il réalise une interpolation linéaire pour les enveloppes d'amplitude de chaque partiel. Il est un peu plus lent que *adsynt*, mais l'interpolation améliore grandement la qualité du son dans les transitoires rapides des enveloppes d'amplitude lorsque  $kr < sr$  (c'est-à-dire quand  $ksmps > 1$ ). Il n'y a pas d'interpolation pour les enveloppes de hauteur, car dans ce cas la dégradation de la qualité sonore n'est pas aussi évidente même avec de grandes valeurs de *ksmps*. Il n'est pas recommandé quand  $kr = sr$  ; dans ce cas, *adsynt* est meilleur (car plus rapide).

## Exemples

Voici un exemple de l'opcode `adsynt2`. Il utilise le fichier `adsynt2.csd` [examples/adsynt2.csd]. Ces deux instruments réalisent une synthèse additive. Leurs sorties respectives sonnent comme un bol tibétain. La première est statique car les paramètres ne sont fixés qu'à l'initialisation. Dans la seconde, les paramètres changent de manière continue.

### Exemple 48. Exemple de l'opcode `adsynt2`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o adsynt2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
; Generate a sinewave table.
giwave ftgen 1, 0, 1024, 10, 1
; Generate two empty tables for adsynt2.
gifrgs ftgen 2, 0, 32, 7, 0, 32, 0
; A table for frequency and amp parameters.
giamps ftgen 3, 0, 32, 7, 0, 32, 0

; Generates parameters at init time
instr 1
; Generate 10 voices.
icnt = 10
; Init loop index.
index = 0

; Loop only executed at init time.
loop:
; Define non-harmonic partials.
ifreq pow index + 1, 1.5
; Define amplitudes.
iamp = 1 / (index+1)
; Write to tables.
tableiw ifreq, index, gifrgs
; Used by adsynt2.
tableiw iamp, index, giamps

index = index + 1
; Do loop/
if (index < icnt) igoto loop

asig adsynt2 0.4, 150, giwave, gifrgs, giamps, icnt
outs asig, asig
endin

; Generates parameters every k-cycle.
instr 2
; Generate 10 voices.
icnt = 10
```

```

; Reset loop index.
kindex = 0

; Loop executed every k-cycle.
loop:
; Generate lfo for frequencies.
kspeed pow kindex + 1, 1.6
; Individual phase for each voice.
kphas phasorbnk kspeed * 0.7, kindex, icnt
klfo table kphas, giwave, 1
; Arbitrary parameter twiddling...
kdepth pow 1.4, kindex
kfreq pow kindex + 1, 1.5
kfreq = kfreq + klfo*0.006*kdepth

; Write freqs to table for adsynt2.
tablew kfreq, kindex, gifrqs

; Generate lfo for amplitudes.
kspeed pow kindex + 1, 0.8
; Individual phase for each voice.
kphas phasorbnk kspeed*0.13, kindex, icnt, 2
klfo table kphas, giwave, 1
; Arbitrary parameter twiddling...
kamp pow 1 / (kindex + 1), 0.4
kamp = kamp * (0.3+0.35*(klfo+1))

; Write amps to table for adsynt2.
tablew kamp, kindex, giamps

kindex = kindex + 1
; Do loop.
if (kindex < icnt) kgoto loop

asig adsynt2 0.25, 150, giwave, gifrqs, giamps, icnt
outs asig, asig
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 2.5 seconds.
i 1 0 2.5
; Play Instrument #2 for 2.5 seconds.
i 2 3 2.5
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Disponible auparavant seulement dans CsoundAV)

# aftouch

aftouch — Reçoit la valeur d'after-touch actuelle de ce canal.

## Description

Reçoit la valeur d'after-touch actuelle de ce canal.

## Syntaxe

```
kaft aftouch [imin] [, imax]
```

## Initialisation

*imin* (facultatif, par défaut 0) -- limite minimale des valeurs obtenues.

*imax* (facultatif, par défaut 127) -- limite maximale des valeurs obtenues.

## Exécution

Reçoit la valeur d'after-touch actuelle de ce canal.

## Exemples

Voici un exemple de l'opcode aftouch. Il utilise le fichier *aftouch.csd* [examples/aftouch.csd].

### Exemple 49. Exemple de l'opcode aftouch.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  No messages  MIDI in
-odac          -d          -M0   ;;RT audio out with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kaft aftouch 0, 1
printk2 kaft

;aftertouch from music keyboard used for volume control
asig oscil 0.7 * kaft, 220, 1
outs asig, asig

endin
```

```
</CsInstruments>
<CsScore>
; sine wave.
f 1 0 16384 10 1

i 1 0 30
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*ampmidi, cpsmidi, cpsmidib, midictrl, notnum, octmidi, octmidib, pchbend, pchmidi, pchmidib, veloc*

## Crédits

Auteurs : Barry L. Vercoe - Mike Berry  
MIT - Mills  
Mai 1997

# alpass

alpass — Réverbère un signal en entrée avec une réponse en fréquence plate.

## Description

Réverbère un signal en entrée avec une réponse en fréquence plate.

## Syntaxe

```
ares alpass asig, xrvt, ilpt [, iskip] [, insmps]
```

## Initialisation

*ilpt* -- durée de boucle en secondes, déterminant la « densité d'échos » de la réverbération. Celle-ci caractérise à son tour la « couleur » du filtre *en peigne* dont la courbe de réponse en fréquence contiendra *ilpt* \* *sr/2* pics régulièrement espacés entre 0 et *sr/2* (la fréquence de Nyquist). La durée de boucle peut être aussi grande que le permet la mémoire disponible. L'espace requis pour une boucle de *n* secondes est de  $4n*sr$  octets. L'espace pour le retard est alloué et retourné comme dans *delay*.

*iskip* (facultatif, 0 par défaut) -- état initial de l'espace de données de la boucle de retard (cf. *reson*). La valeur par défaut est 0.

*insmps* (facultatif, 0 par défaut) -- valeur du retard, en nombre d'échantillons.

## Exécution

*xrvt* -- la durée de réverbération (définie comme le temps en secondes pris par un signal pour décroître à 1/1000 ou 60 dB de son amplitude originale).

Le filtre répète l'entrée avec une densité d'écho déterminée par la durée de boucle *ilpt*. Le taux d'atténuation est indépendant et il est déterminé par *xrvt*, la durée de réverbération (définie comme le temps en secondes pris par un signal pour décroître à 1/1000 ou 60 dB de son amplitude originale). La sortie apparaît sans retard.

## Exemples

Voici un exemple de l'opcode alpass. Il utilise le fichier *alpass.csd* [examples/alpass.csd].

### Exemple 50. Exemple de l'opcode alpass.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o alpass.wav -W ;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

sr = 44100
ksmps = 100
nchnls = 2
0dbfs = 1

gamix init 0

instr 1

acps    expon p4, p3, p5
asig vco 0.6, acps, 1
outs asig, asig

gamix = gamix + asig

endin

instr 99

arvt1 line 3.5*1.5, p3, 6
arvt2 line 3.5, p3, 4
ilpt = 0.1
aleft alpass gamix, arvt1, ilpt
aright alpass gamix, arvt2, ilpt*2
outs aleft, aright

gamix = 0 ; clear mixer

endin

</CsInstruments>
<CsScore>
f1 0 4096 10 1
i 1 0 3 20 2000

i 99 0 8
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*comb, reverb, valpass, vcomb*

## Crédits

Paramètres de taux audio introduits dans la version 6.02

Novembre 2013.



# alwayson

`alwayson` — Active les instruments indiqués dans l'en-tête de l'orchestre sans le recours à une instruction `i`.

## Description

Opcodes du greffon `signalflowgraph`.

Active les instruments indiqués dans l'en-tête de l'orchestre sans le recours à une instruction `i`. Les instruments doivent être activés dans le même ordre que celui de leur définition.

L'opcode `alwayson` est conçu pour simplifier la définition d'orchestres réutilisables avec traitement du signal ou enchainement d'effets et réseaux.

## Syntaxe

```
alwayson Tinstrument [p4, ..., pn]
```

## Initialisation

*Tinstrument* -- Nom sous forme de chaîne de caractères de la définition d'instrument à activer.

[*p4*, ..., *pn*] -- *p*-champs optionnels à passer à l'instrument, dans le même ordre et avec les mêmes types que si c'était une instruction `i`.

Lors de l'activation de l'instrument, *p1* est son numéro, *p2* vaut 0 et *p3* vaut -1. Les *p*-champs à partir de *p4* peuvent être envoyés à l'instrument de manière optionnelle.

## Exemples

Voici un exemple de l'opcode `alwayson`. Il utilise le fichier `alwayson.csd` [examples/alwayson.csd].

### Exemple 51. Exemple de l'opcode `alwayson`.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
;-Wfo alwayson.wav
</CsOptions>
<CsInstruments>

; Initialize the global variables.

sr      = 44100
ksmps   = 32
nchnls  = 2

; Connect up instruments and effects to create the signal flow graph.

connect "SimpleSine", "leftout", "Reverberator", "leftin"
connect "SimpleSine", "rightout", "Reverberator", "rightin"

connect "Moogy", "leftout", "Reverberator", "leftin"
connect "Moogy", "rightout", "Reverberator", "rightin"
```

```

connect "Reverberator", "leftout", "Compressor", "leftin"
connect "Reverberator", "rightout", "Compressor", "rightin"

connect "Compressor", "leftout", "Soundfile", "leftin"
connect "Compressor", "rightout", "Soundfile", "rightin"

; Turn on the "effect" units in the signal flow graph.

alwayson "Reverberator", 0.91, 12000
alwayson "Compressor"
alwayson "Soundfile"

; Define instruments and effects in order of signal flow.

instr SimpleSine
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ; Default values:  p1 p2 p3 p4 p5 p6 p7 p8 p9 p10
    pset 0, 0, 10, 0, 0, 0, 0.5
iattack = 0.015
idecay = 0.07
isustain = p3
irelease = 0.3
p3 = iattack + idecay + isustain + irelease
adamping linsegr 0.0, iattack, 1.0, idecay + isustain, 1.0, irelease, 0.0
iHz = cpsmidinn(p4)
    ; Rescale MIDI velocity range to a musically usable range of dB.
iamplitude = ampdb(p5 / 127 * 15.0 + 60.0)
    ; Use ftgenonce instead of ftgen, ftgentmp, or f statement.
icosine ftgenonce 0, 0, 65537, 11, 1
aoscili oscili iamplitude, iHz, icosine
aadsr madsr iattack, idecay, 0.6, irelease
asignal = aoscili * aadsr
aleft, aright pan2 asignal, p7
    ; Stereo audio output to be routed in the orchestra header.
outleta "leftout", aleft
outleta "rightout", aright
endin

instr Moogy
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ; Default values:  p1 p2 p3 p4 p5 p6 p7 p8 p9 p10
    pset 0, 0, 10, 0, 0, 0, 0.5
iattack = 0.003
isustain = p3
irelease = 0.05
p3 = iattack + isustain + irelease
adamping linsegr 0.0, iattack, 1.0, isustain, 1.0, irelease, 0.0
iHz = cpsmidinn(p4)
    ; Rescale MIDI velocity range to a musically usable range of dB.
iamplitude = ampdb(p5 / 127 * 20.0 + 60.0)
print iHz, iamplitude
    ; Use ftgenonce instead of ftgen, ftgentmp, or f statement.
isine ftgenonce 0, 0, 65537, 10, 1
asignal vco iamplitude, iHz, 1, 0.5, isine
kfco line 2000, p3, 200
krez = 0.8
asignal moogvcf asignal, kfco, krez, 100000
asignal = asignal * adamping
aleft, aright pan2 asignal, p7
    ; Stereo audio output to be routed in the orchestra header.
outleta "leftout", aleft
outleta "rightout", aright
endin
instr Reverberator
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ; Stereo input.

```

```

aleftin  inleta      "leftin"
arightin inleta      "rightin"
idelay   =           p4
icutoff  =           p5
aleft, aright reverbsc aleftin, arightin, idelay, icutoff
; Stereo output.
outleta  "leftout", aleft
outleta  "rightout", aright
endin

instr Compressor
; Stereo input.
aleftin  inleta      "leftin"
arightin inleta      "rightin"
kthreshold = 25000
icompl    = 0.5
icomp2    = 0.763
irtime    = 0.1
ifetime   = 0.1
aleftout  dam        aleftin, kthreshold, icomp1, icomp2, irtime, iftime
arightout dam        arightin, kthreshold, icomp1, icomp2, irtime, iftime
; Stereo output.
outleta  "leftout", aleftout
outleta  "rightout", arightout
endin

instr Soundfile
; Stereo input.
aleftin  inleta      "leftin"
arightin inleta      "rightin"
outs     aleftin, arightin
endin

</CsInstruments>
<CsScore>

; It is not necessary to activate "effects" or create f-tables in the score!
; Overlapping notes create new instances of instruments with proper connections.

i "SimpleSine" 1 5 60 85
i "SimpleSine" 2 5 64 80
i "Moogy" 3 5 67 75
i "Moogy" 4 5 71 70
; 1 extra second after the performance
e 1

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Par Michael Gogins, 2009

# ampdb

ampdb — Retourne l'amplitude équivalente à la valeur  $x$  donnée en décibel.

## Description

Retourne l'amplitude équivalente à la valeur  $x$  donnée en décibel.

- 60 dB = 1000
- 66 dB = 1995.262
- 72 dB = 3891.07
- 78 dB = 7943.279
- 84 dB = 15848.926
- 90 dB = 31622.764

## Syntaxe

**ampdb**( $x$ ) (pas de restriction de taux)

## Exemples

Voici un exemple de l'opcode ampdb. Il utilise le fichier *ampdb.csd* [examples/ampdb.csd].

### Exemple 52. Exemple de l'opcode ampdb.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ampdb.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1

idb = p4
iamp = ampdb(idb)
asig oscil iamp, 220
print iamp
outs asig, asig
endin
```

```
</CsInstruments>
<CsScore>
i 1 0 1 50
i 1 + 1 90
i 1 + 1 68
i 1 + 1 80

e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1: iamp = 316.228
instr 1: iamp = 31622.763
instr 1: iamp = 2511.886
instr 1: iamp = 9999.996
```

## Voir aussi

*ampdbfs, db, dbamp, dbfsamp*

# ampdbfs

ampdbfs — Retourne l'amplitude équivalente (sur une échelle d'entiers signés sur 16 bit) à la valeur  $x$  de l'amplitude maximale (dB FS).

## Description

Retourne l'amplitude équivalente à la valeur  $x$  de l'amplitude maximale (dB FS). Les valeurs logarithmiques de l'échelle en décibels sont converties en valeurs linéaires entières sur 16 bit allant de -32768 à +32767.

## Syntaxe

**ampdbfs**( $x$ ) (pas de restriction de taux)

## Exemples

Voici un exemple de l'opcode ampdbfs. Il utilise le fichier *ampdbfs.csd* [examples/ampdbfs.csd].

### Exemple 53. Exemple de l'opcode ampdbfs.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ampdbfs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1

idb = p4
iamp = ampdbfs(idb)
asig oscil iamp, 220, 1
  print iamp
  outs asig, asig
endin

</CsInstruments>
<CsScore>
;sine wave.
f 1 0 16384 10 1

i 1 0 1 -1
i 1 + 1 -5
i 1 + 1 -6
```

```
i 1 + 1 -20  
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1: iamp = 29204.511  
instr 1: iamp = 18426.801  
instr 1: iamp = 16422.904  
instr 1: iamp = 3276.800
```

## Voir aussi

*ampdb, dbamp, dbfsamp, 0dbfs*

Nouveau dans la version 4.10 de Csound.

# ampmidi

ampmidi — Retourne la vélocité de l'évènement MIDI en cours.

## Description

Retourne la vélocité de l'évènement MIDI en cours.

## Syntaxe

```
iamp ampmidi iscal [, ifn]
```

## Initialisation

*iscal* -- facteur de pondération de taux-i

*ifn* (facultatif, par défaut 0) -- numéro d'une table de fonction contenant un tableau de conversion normalisé, grâce auquel la valeur entrante est interprétée. La valeur par défaut est 0, ce qui signifie pas de conversion.

## Exécution

Reçoit la vélocité de l'évènement MIDI en cours, la modifie éventuellement grâce à une table de conversion normalisée, et retourne une valeur d'amplitude dans l'intervalle 0 - *iscal*.

## Exemples

Voici un exemple de l'opcode `ampmidi`. Il utilise le fichier *ampmidi.csd* [examples/ampmidi.csd].

### Exemple 54. Exemple de l'opcode `ampmidi`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o ampmidi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;expects MIDI note inputs on channel 1

iamp ampmidi 1 ; scale amplitude between 0 and 1
asig oscil iamp, 220, 1
print iamp
outs asig, asig
```



```
    endin

    </CsInstruments>
    <CsScore>
    ;Dummy f-table for 1 minute
    f 0 60
    ;sine wave.
    f 1 0 16384 10 1

    e

    </CsScore>
    </CsoundSynthesizer>
```

## Voir aussi

*aftouch, cpsmidi, cpsmidib, midictrl, notnum, octmidi, octmidib, pchbend, pchmidi, pchmidib, veloc*

## Crédits

Auteurs : Barry L. Vercoe - Mike Berry  
MIT - Mills  
Mai 1997

# ampmidicurve

ampmidicurve — Associe à une vélocité MIDI en entrée un facteur de gain en sortie de valeur maximale 1, en modifiant ce gain de sortie par un intervalle dynamique et un exposant de mise en forme.

## Description

Opcode du greffon ampmidid.

Associe à une vélocité MIDI en entrée un facteur de gain en sortie de valeur maximale 1, en modifiant ce gain de sortie par un intervalle dynamique et un exposant de mise en forme. Le gain minimum en sortie vaut 1 moins l'intervalle dynamique. Un exposant de mise en forme valant 1 donne une réponse linéaire ; une augmentation de l'exposant produit un coude décroissant progressivement dans la courbe de réponse du gain.

## Syntaxe

igain **ampmidicurve** ivelocity, idynamicrange, iexponent

kgain **ampmidicurve** kvelocity, kdynamicrange, kexponent

## Initialisation

*imidivelocity* -- Vélocité MIDI, comprise entre 0 et 127.

*idynamicrange* -- Intervalle dynamique désiré du gain, entre 0 et 1.

*iexponent* -- Exposant appliqué pour mettre en forme la courbe de réponse du gain, 1 ou plus.

## Exécution

*kmidivelocity* -- Vélocité MIDI, comprise entre 0 et 127.

*kdynamicrange* -- Intervalle dynamique désiré du gain, entre 0 et 1.

*kexponent* -- Exposant appliqué pour mettre en forme la courbe de réponse du gain, 1 ou plus.

Associe à une vélocité MIDI en entrée un facteur de gain en sortie de valeur maximale 1, en modifiant ce gain de sortie par un intervalle dynamique et un exposant de mise en forme. Le gain minimum en sortie vaut 1 moins l'intervalle dynamique. Un exposant de mise en forme valant 1 donne une réponse linéaire ; une augmentation de l'exposant produit un coude décroissant progressivement dans la courbe de réponse du gain, selon l'équation :  $y = d * (x/127)^h + 1 - d$ , où  $y$  = le gain,  $x$  = la vélocité MIDI en entrée (de 0 à 127),  $d$  = l'intervalle dynamique (de 0 à 1), et  $h$  = l'exposant de mise en forme (1 ou plus). Cet opcode a été suggéré par Mauro Giubileo.

## Exemples

Voici un exemple de l'opcode ampmidicurve. Il utilise le fichier *ampmidicurve.csd* [exemples/ampmidi-curve.csd].

### Exemple 55. Exemple de l'opcode ampmidicurve.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
;;;RT audio out, note=p4 and velocity=p5
-odac --midi-key=4 --midi-velocity-amp=5
;-iadc    ;;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ampmidid.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

massign 0, 1 ;assign all midi to instr. 1

instr 1

isine ftgenonce 0, 0, 4096, 10, 1 ;sine wave

    ihz = cpsmidinn(p4)
    ivelocity = p5
    ; MIDI velocity to signal amplitude.
    iamplitude = ampdb(ivelocity)
    ; Gain with compressed dynamic range, soft knee.
    igain ampmidicurve ivelocity, .92, 3
    print ivelocity, iamplitude, igain
    a1 oscili 1, ihz, isine
    aenv madsr 0.05, 0.1, 0.5, 0.2
    asig = a1 * aenv * igain
    outs asig, asig

endin

</CsInstruments>
<CsScore>
;      note velocity
i 1 0 2 61 100
i 1 + 2 65 10
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*aftouch, cpsmidi, cpsmidib, midictrl, notnum, octmidi, octmidib, pchbend, pchmidi, pchmidib, veloc*

## Crédits

Auteur : Michael Gogins  
2019

# ampmidid

ampmidid — Fait correspondre musicalement la vitesse MIDI avec l'amplitude dans un intervalle dynamique spécifié en décibels.

## Description

Opcodes du greffon ampmidid.

Fait correspondre musicalement la vitesse MIDI avec l'amplitude dans un intervalle dynamique spécifié en décibels.

## Syntaxe

```
iamplitude ampmidid ivelocity, idecibels  
kamplitude ampmidid kvelocity, idecibels
```

## Initialisation

*iamplitude* -- Amplitude.

*ivelocity* -- vitesse MIDI, comprise entre 0 et 127.

*idecibels* -- Intervalle dynamique désiré en décibels.

## Exécution

*kamplitude* -- Amplitude.

*kvelocity* -- vitesse MIDI, comprise entre 0 et 127.

Fait correspondre musicalement la vitesse MIDI avec l'amplitude dans un intervalle dynamique spécifié en décibels :  $a = (m * v + b) ^ 2$ , où  $a$  = amplitude,  $v$  = vitesse MIDI,  $r = 10 ^ (R / 20)$ ,  $b = 127 / (126 * \text{sqrt}(r)) - 1 / 126$ ,  $m = (1 - b) / 127$ , et  $R$  = intervalle dynamique spécifié en décibels. Voir Roger Dannenberg, "The Interpretation of MIDI Velocity," dans Georg Essl and Ichiro Fujinaga (Eds.), Proceedings of the 2006 International Computer Music Conference, Novembre 6-11, 2006 (San Francisco : The International Computer Music Association), pp. 193-196.

## Exemples

Voici un exemple de l'opcode ampmidid. Il utilise le fichier *ampmidid.csd* [examples/ampmidid.csd].

### Exemple 56. Exemple de l'opcode ampmidid.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
;;RT audio out, note=p4 and velocity=p5
```

```

-odac --midi-key=4 --midi-velocity-amp=5
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ampmidid.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

massign 0, 1 ;assign all midi to instr. 1

instr 1

isine ftgenonce 0, 0, 4096, 10, 1 ;sine wave

    ihz = cpsmidinn(p4)
    ivelocity = p5
    idb ampmidid ivelocity, 20 ;map to dynamic range of 20 dB.
    idb = idb + 60 ;limit range to 60 to 80 decibels
    iamplitude = ampdb(idb) ;loudness in dB to signal amplitude

a1 oscili iamplitude, ihz, isine
aenv madsr 0.05, 0.1, 0.5, 0.2
asig = a1 * aenv
    outs asig, asig

endin

</CsInstruments>
<CsScore>
;      note velocity
i 1 0 2 61 100
i 1 + 2 65 10
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*aftouch, cpsmidi, cpsmidib, midictrl, notnum, octmidi, octmidib, pchbend, pchmidi, pchmidib, veloc*

## Crédits

Auteur : Michael Gogins  
2006

# areson

**areson** — Un filtre réjecteur de bande réglable (notch filter) dont les fonctions de transfert sont les complémentaires de celles de l'opcode *reson*.

## Description

Un filtre réjecteur de bande réglable dont les fonctions de transfert sont les complémentaires de celles de l'opcode *reson*.

## Syntaxe

```
ares areson asig, kcf, kbw [, iscl] [, iskip]
ares areson asig, acf, kbw [, iscl] [, iskip]
ares areson asig, kcf, abw [, iscl] [, iskip]
ares areson asig, acf, abw [, iscl] [, iskip]
```

## Initialisation

*iscl* (facultatif, par défaut 0) -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf/acf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. (Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc.) Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*ares* -- le signal de sortie au taux audio.

*asig* -- le signal d'entrée au taux audio.

*kcf/acf* -- la fréquence centrale du filtre, ou position fréquentielle de la crête de la réponse.

*kbw/abw* -- largeur de bande du filtre (la différence en Hz entre les points haut et bas à mi-puissance).

*areson* est un filtre dont les fonctions de transfert sont complémentaires de celles de *reson*. Ainsi *areson* est un filtre réjecteur de bande variable (notch filter) dont les fonctions de transfert représentent les aspects « filtrés » de leurs compléments. Cependant, l'échelle de puissance n'est pas normalisée dans *areson* mais reste le complément exact de l'unité correspondante. Ainsi les deux versions d'un signal audio filtré par des unités *reson* et *areson* correspondantes, redonneraient par addition le signal original.

Cette propriété est particulièrement utile pour contrôler le mélange de différentes sources (voir *lpreson*). On peut obtenir des courbes de réponse complexes comme celles qui présentent plusieurs valeurs maximales, en utilisant un banc de filtres adéquats en série. (La réponse résultante est le produit des différentes

réponses.) Dans une telle situation, les atténuations combinées peuvent conduire à une sérieuse perte de puissance du signal, mais celle-ci peut être compensée au moyen de *balance*.



### Avertissement

Lorsqu'on l'utilise avec *iscl* cet opcode n'est pas un filtre réjecteur de bande mais se comporte comme *reson*.

## Exemples

Voici un exemple de l'opcode *areson*. Il utilise le fichier *areson.csd* [examples/areson.csd].

### Exemple 57. Exemple de l'opcode *areson*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o areson.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; unfiltered noise

asig rand 0.5 ; white noise signal.
outs asig, asig
endin

instr 2 ; filtered noise

kcf init 1000
kbw init 100
asig rand 0.5
afil areson asig, kcf, kbw
afil balance afil,asig ; afil = very loud
outs afil, afil
endin

</CsInstruments>
<CsScore>

i 1 0 2
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*aresonk*, *atone*, *atonek*, *port*, *portk*, *reson*, *resonk*, *tone*, *tonek*

## Crédits

Paramètres de taux audio introduits dans la version 6.02

Octobre 2013.



# aresonk

**aresonk** — Un filtre réjecteur de bande réglable (notch filter) dont les fonctions de transfert sont les complémentaires de celles de l'opcode *reson*.

## Description

Un filtre réjecteur de bande réglable dont les fonctions de transfert sont les complémentaires de celles de l'opcode *reson*.

## Syntaxe

```
kres aresonk ksig, kcf, kbw [, iscl] [, iskip]
```

## Initialisation

*iscl* (facultatif, par défaut 0) -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. (Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc.) Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*kres* -- le signal de sortie au taux de contrôle.

*ksig* -- le signal d'entrée au taux de contrôle.

*kcf* -- la fréquence centrale du filtre, ou position fréquentielle de la crête de la réponse.

*kbw* -- largeur de bande du filtre (la différence en Hz entre les points haut et bas à mi-puissance).

*aresonk* est un filtre dont les fonctions de transfert sont complémentaires de celles de *resonk*. Ainsi *aresonk* est un filtre réjecteur de bande variable (notch filter) dont les fonctions de transfert représentent les aspects « filtrés » de leurs compléments. Cependant, l'échelle de puissance n'est pas normalisée dans *aresonk* mais reste le complément réel de l'unité correspondante.

## Exemples

Voici un exemple de l'opcode *aresonk*. Il utilise le fichier *aresonk.csd* [exemples/aresonk.csd].

### Exemple 58. Exemple de l'opcode *aresonk*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o aresonk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisin ftgen 0, 0, 2^10, 10, 1

instr 1

ksig randomh 400, 1800, 150
aout poscil .2, 1000+ksig, gisin
outs aout, aout
endin

instr 2

ksig randomh 400, 1800, 150
kbw line 1, p3, 600 ; vary bandwidth
ksig aresonk ksig, 800, kbw
aout poscil .2, 1000+ksig, gisin
outs aout, aout
endin

</CsInstruments>
<CsScore>
i 1 0 5
i 2 5.5 5
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*areson, atone, atonek, port, portk, reson, resonk, tone, tonek*

# atone

*atone* — Un filtre passe-haut dont les fonctions de transfert sont les complémentaires de celles de l'opcode *tone*.

## Description

Un filtre passe-haut dont les fonctions de transfert sont les complémentaires de celles de l'opcode *tone*.

## Syntaxe

```
ares atone asig, khp [, iskip]
```

## Initialisation

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*ares* -- le signal de sortie au taux audio.

*asig* -- le signal d'entrée au taux audio.

*khp* -- le point à mi-puissance de la courbe de réponse, en Hertz. La mi-puissance est définie par puissance maximale / racine de 2.

*atone* est un filtre dont les fonctions de transfert sont complémentaires de celles de *tone*. Ainsi *atone* est un filtre passe-haut dont les fonctions de transfert représentent les aspects « filtrés » de leurs compléments. Cependant, l'échelle de puissance n'est pas normalisée dans *atone* mais reste le complément réel de l'unité correspondante. Ainsi les deux versions d'un signal audio filtré par des unités *tone* et *atone* correspondantes, redonneraient par addition le signal original.

Cette propriété est particulièrement utile pour contrôler le mélange de différentes sources (voir *lpreson*). On peut obtenir des courbes de réponse complexes comme celles qui présentent plusieurs valeurs maximales, en utilisant un banc de filtres adéquats en série. (La réponse résultante est le produit des différentes réponses.) Dans une telle situation, les atténuations combinées peuvent conduire à une sérieuse perte de puissance du signal, mais celle-ci peut être restaurée au moyen de *balance*.

## Exemples

Voici un exemple de l'opcode *atone*. Il utilise le fichier *atone.csd* [examples/atone.csd].

### Exemple 59. Exemple de l'opcode *atone*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o atone.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;white noise

asig rand 1
outs asig, asig

endin

instr 2 ;filtered noise

asig rand 1
khp init 4000
asig atone asig, khp
outs asig, asig

endin

</CsInstruments>
<CsScore>

i 1 0 2
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*areson, aresonk, atonek, port, portk, reson, resonk, tone, tonek*

# atonek

*atonek* — Un filtre passe-haut dont les fonctions de transfert sont les complémentaires de celles de l'opcode *tonek*.

## Description

Un filtre passe-haut dont les fonctions de transfert sont les complémentaires de celles de l'opcode *tonek*.

## Syntaxe

```
kres atonek ksig, khp [, iskip]
```

## Initialisation

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*kres* -- le signal de sortie au taux de contrôle.

*ksig* -- le signal d'entrée au taux de contrôle.

*khp* -- le point à mi-puissance de la courbe de réponse, en Hertz. La mi-puissance est définie par puissance maximale / racine de 2.

*atonek* est un filtre dont les fonctions de transfert sont complémentaires de celles de *tonek*. Ainsi *atonek* est un filtre passe-haut dont les fonctions de transfert représentent les aspects « filtrés » de leurs compléments. Cependant, l'échelle de puissance n'est pas normalisée dans *atonek* mais reste le complément réel de l'unité correspondante.

## Exemples

Voici un exemple de l'opcode *atonek*. Il utilise le fichier *atonek.csd* [examples/atonek.csd].

### Exemple 60. Exemple de l'opcode *atonek*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o atonek.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisin ftgen 0, 0, 2^10, 10, 1

instr 1

ksig randomh 400, 1800, 150
aout poscil .2, 1000+ksig, gisin
outs aout, aout
endin

instr 2

ksig randomh 400, 1800, 150
khp line 1, p3, 400 ;vary high-pass
ksig atonek ksig, khp
aout poscil .2, 1000+ksig, gisin
outs aout, aout
endin

</CsInstruments>
<CsScore>
i 1 0 5
i 2 5.5 5
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*areson, aresonk, atone, port, portk, reson, resonk, tone, tonek*

## Crédits

Auteur : Robin Whittle  
 Australie  
 Mai 1997

# atonex

atonex — Emule une série de filtres utilisant l'opcode *atone*.

## Description

*atonex* est équivalent à un filtre constitué de plusieurs couches de filtres *atone* avec les mêmes arguments, connectés en série. L'utilisation d'une série d'un nombre important de filtres permet une pente de coupure plus raide. Ils sont plus rapides que l'équivalent obtenu à partir du même nombre d'instances d'opcodes classiques dans un orchestre Csound, car il n'y aura qu'un cycle d'initialisation et une seule passe de *k* cycles de contrôle à la fois et la boucle audio sera entièrement contenue dans la mémoire cache du processeur.

## Syntaxe

```
ares atonex asig, khp [, inumlayer] [, iskip]
ares atonex asig, ahp [, inumlayer] [, iskip]
```

## Initialisation

*inumlayer* (facultatif) -- nombre d'éléments dans la série de filtre. La valeur par défaut est 4.

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal d'entrée

*khp/ahp* -- le point à mi-puissance de la courbe de réponse. La mi-puissance est définie par puissance maximale / racine de 2.

## Exemples

Voici un exemple de l'opcode *atonex*. Il utilise le fichier *atonex.csd* [examples/atonex.csd].

### Exemple 61. Exemple de l'opcode *atonex*.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o atonex.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
```

```

nchnls = 2
0dbfs = 1

instr 1 ; unfiltered noise

asig rand 0.7 ; white noise
outs asig, asig

endin

instr 2 ; filtered noise

asig rand 0.7
khp line 100, p3, 3000
afilt atonex asig, khp, 32

; Clip the filtered signal's amplitude to 85 dB.
a1 clip afilt, 2, ampdb(85)
outs a1, a1
endin

</CsInstruments>
<CsScore>

i 1 0 2
i 2 2 2

e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*resonx, tonex*

## Crédits

Auteur : Gabriel Maldonado (adapté par John ffitich)  
 Italie

Nouveau dans la version 3.49 de Csound

Paramètres de taux audio introduits dans la version 6.02

Octobre 2013.



# ATSadd

ATSadd — Utilise les données d'un fichier d'analyse ATS pour réaliser une synthèse additive.

## Description

*ATSadd* lit depuis un fichier d'analyse ATS et utilise les données pour réaliser une synthèse additive à partir d'une batterie interne d'oscillateurs avec interpolation.

## Syntaxe

```
ar ATSadd ktimepnt, kfmod, iatsfile, ifn, ipartials[, ipartialoffset, \
    ipartialincr, igatefn]
```

## Initialisation

*iatsfile* – le numéro ATS (n dans ats.n) ou le nom entre guillemets du fichier d'analyse créé avec *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>].

*ifn* – numéro de table d'une fonction stockée contenant une onde sinus pour *ATSadd* et une onde cosinus pour *ATSaddnz* (voir les exemples ci-dessous pour plus d'information).

*ipartials* – nombre de partiels qui seront utilisés dans la resynthèse (le bruit a un maximum de 25 bandes).

*ipartialoffset* (facultatif) – le premier partiel utilisé (0 par défaut).

*ipartialincr* (facultatif) – fixe le pas d'incrémentation que ces opcodes de synthèse utilisent pour compter les composants bins à partir de *ipartialoffset* dans la resynthèse (1 par défaut).

*igatefn* (facultatif) – numéro d'une fonction stockée qui sera appliquée aux amplitudes des bins de l'analyse avant la resynthèse. Si *igatefn* est supérieur à 0 les amplitudes de chaque bin seront pondérées par *igatefn* selon un simple procédé de mise en correspondance. D'abord les amplitudes de tous les bins de toutes les trames du fichier d'analyse sont comparées pour déterminer la valeur maximale de l'amplitude. Cette valeur est ensuite utilisée pour créer des amplitudes normalisées comme indices dans la fonction stockée *igatefn*. L'amplitude maximale correspondra au dernier point de la fonction. Une amplitude de 0 correspondra au premier point de la fonction. Les valeurs comprises entre 0 et 1 correspondront aux points à l'intérieur de la table de fonction. Voir les exemples ci-dessous.

## Exécution

*ktimepnt* – Le pointeur de temps en secondes utilisé comme indice sur le fichier ATS. Est utilisé pour *ATSadd* exactement de la même manière que pour *pvoc*.

*ATSadd* et *ATSaddnz* sont basés sur *pvadd* par Richard Karpen et ils utilisent des fichier créés par ATS de Juan Pampin (*Analyse - Transformation - Synthèse* [<http://www-ccrma.stanford.edu/~juan/ATS.html>]).

*kfmod* – Un facteur de transposition du taux de contrôle : la valeur 1 implique pas de transposition, 1.5 transpose vers l'aigu d'une quinte juste et 0.5 vers le grave d'une octave. Est utilisé pour *ATSadd* exactement de la même manière que pour *pvoc*.

*ATSadd* lit depuis un fichier d'analyse ATS et utilise les données pour réaliser une synthèse additive à partir d'une batterie interne d'oscillateurs avec interpolation. L'utilisateur fournit la table d'onde (habituellement une période d'onde sinusoïdale) et il peut choisir quels partiels de l'analyse seront utilisés dans la resynthèse.

## Exemples

```
ktime line 0, p3, 2.5
asig ATSadd ktime, 1, "clarinet.ats", 1, 20, 2
```

Dans l'exemple ci-dessus, *ipartials* vaut 20 et *ipartialoffset* vaut 2. Les partiels du fichier d'analyse "clarinet.ats" allant du 3ème au 22ème seront synthétisés. *kfmod* vaut 1 et il n'y aura ainsi pas de modification de la hauteur. Comme l'enveloppe *ktimepnt* évolue de 0 à 2.5 pendant la durée de la note, le fichier d'analyse sera lu de 0 à 2.5 secondes de la durée originale de l'analyse pendant la durée de la note csound, ce qui permet de changer la durée indépendamment de la hauteur.

## Exemples

Voici un exemple complet de l'opcode ATSadd. Il utilise le fichier *ATSadd.csd* [examples/ATSadd.csd].

### Exemple 62. Exemple de l'opcode ATSadd.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc for RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ATSadd.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
dbfs = 1

instr 1 ; "beats.ats" is created by atsa

ktime line 0, p3, 2
asig ATSadd ktime, 1, "beats.ats", 1, 20, 0, 2
outs asig*3, asig*3 ;amplify

endin

</CsInstruments>
<CsScore>
;sine wave.
f 1 0 16384 10 1

i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Dans l'exemple ci-dessus nous synthétisons 20 partiels comme dans l'exemple 1 sauf que cette fois-ci nous fixons *ipartialoffset* à 0 et *ipartialincr* à 2, ce qui veut dire que nous commençons avec le premier partiel et que nous synthétisons au total 20 partiels, ignorant tous les partiels impairs (1, 3, 5, ...).

Voici un autre exemple de l'opcode ATSadd. Il utilise le fichier *ATSadd-2.csd* [examples/ATSadd-2.csd].

### Exemple 63. Exemple 2 de l'opcode ATSadd.

```

<CsoundSynthesizer>
<CsOptions>
-odac -d -ml
</CsOptions>
<CsInstruments>
;example by joachim heintz
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine      ftgen      0, 0, 1024, 10, 1

  instr AllTheTones
Sfile      =          "fox.ats"
           prints      "Resynthesizing with all the tones.\n"
iDur       ATSinfo     Sfile, 7
p3         =          iDur
iNumParts  ATSinfo     Sfile, 3
           prints      "Overall number of partials = %d\n", iNumParts
ktime      line        0, iDur, iDur
asig       ATSadd      ktime, 1, Sfile, giSine, iNumParts
           outs        asig, asig

;start next instr
           event_i      "i", "TonesInBandsOfTen", iDur+1, iDur, 0, iNumParts
  endin

  instr TonesInBandsOfTen
Sfile      =          "fox.ats"
iOffset    =          p4 ;start at this partial
iNumParts  =          p5 ;overall number of partials
           prints      "Resynthesizing with partials %d - %d.\n", iOffset+1, iOffset+10
ktime      line        0, p3, p3
asig       ATSadd      ktime, 1, Sfile, giSine, 10, iOffset
           outs        asig, asig

;start next instance until there are enough partials left
  if iOffset+20 < iNumParts then
           event_i      "i", "TonesInBandsOfTen", p3+1, p3, iOffset+10, iNumParts
        else
           event_i      "i", "End", p3, 1
  endif
  endin

  instr End
           exitnow
  endin
</CsInstruments>
<CsScore>
i "AllTheTones" 0 1
e 999
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*ATSread, ATSreadnz, ATSinfo, ATSbufread, ATScross, ATSinterpread, ATSpartialtap, ATSaddnz, ATSsinnoi*

## Crédits

Auteur : Alex Norman

Seattle, Washington  
2004

# ATSaddnz

ATSaddnz — Utilise les données d'un fichier d'analyse ATS pour réaliser une synthèse de bruit.

## Description

*ATSaddnz* lit depuis un fichier d'analyse ATS et utilise les données pour réaliser une synthèse additive en utilisant une fonction randi modifiée.

## Syntaxe

```
ar ATSaddnz ktimepnt, iatsfile, ibands[, ibandoffset, ibandincr]
```

## Initialisation

*iatsfile* – le numéro ATS (n dans ats.n) ou le nom entre guillemets du fichier d'analyse créé avec *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>].

*ibands* – nombre de bandes de bruit qui seront utilisées dans la resynthèse (le bruit comprend 25 bandes au maximum).

*ibandoffset* (facultatif) – la première bande de bruit utilisée (0 par défaut).

*ibandincr* (facultatif) – fixe le pas d'incrément que ces opcodes de synthèse utilisent pour compter les composants bins à partir de *ipartialoffset* dans la resynthèse (1 par défaut).

## Exécution

*ktimepnt* – Le pointeur de temps en secondes utilisé comme indice sur le fichier ATS. Est utilisé pour *ATSaddnz* exactement de la même manière que pour *pvoc* et *ATSadd*.

*ATSaddnz* et *ATSadd* sont basés sur *pvadd* par Richard Karpen et ils utilisent des fichiers créés par ATS de Juan Pampin (*Analyse - Transformation - Synthèse* [<http://www-ccrma.stanford.edu/~juan/ATS.html>]).

*ATSaddnz* lit aussi depuis un fichier d'analyse ATS mais il resynthétise le bruit depuis les données d'énergie du bruit contenues dans le fichier ATS. Il utilise une fonction randi modifiée pour créer du bruit à bande limitée et le module avec une onde cosinus, pour synthétiser une sélection de bandes de fréquence spécifiée par l'utilisateur. La modulation du bruit est nécessaire pour placer le bruit à bande limitée au bon endroit dans le spectre de fréquence.

## Exemples

```
ktime line      0, p3, 2.5
asig ATSaddnz ktime, "clarinet.ats", 25
```

Dans l'exemple ci-dessus nous resynthétisons les 25 bandes de bruit depuis les données contenues dans le fichier d'analyse nommé "clarinet.ats".

## Exemples

Voici un exemple complet de l'opcode *ATSaddnz*. Il utilise le fichier *ATSaddnz.csd* [exemples/*ATSaddnz.csd*].

**Exemple 64. Exemple de l'opcode ATSaddnz.**

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc for RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ATSaddnzwav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; "beats.ats" is created by atsa

ktime line      0, p3, 2
asig ATSaddnz ktime, "cage.ats", 1, 24
  outs asig*10, asig*10 ;amplify
endin

</CsInstruments>
<CsScore>

i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

Ici nous ne resynthétisons que la 25ème bande de bruit (*ibandoffset* à 24 et *ibands* à 1).

Voici un autre exemple de l'opcode ATSaddnz. Il utilise le fichier *ATSaddnz-2.csd* [exemples/ATSaddnz-2.csd].

**Exemple 65. Exemple 2 de l'opcode ATSaddnz.**

```

<CsoundSynthesizer>
<CsOptions>
-odac -d -ml
</CsOptions>
<CsInstruments>
;example by joachim heintz
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr AllTheNoise
Sfile      =      "fox.ats"
  prints    "Resynthesizing with all the noise.\n"
iDur       ATSinfo Sfile, 7
p3         =      iDur
ktime      line    0, iDur, iDur
asig       ATSaddnz ktime, Sfile, 25
  outs      asig, asig

;start next instr
event_i "i", "NoiseInBandsOfFive", iDur+1, 1, 0

```

```
    endin

    instr NoiseInBandsOfFive
    Sfile      =      "fox.ats"
    prints      "Resynthesizing with noise bands %d - %d.\n", p4, p4+5
    iDur      ATInfo  Sfile, 7
    p3      =      iDur
    ktime      line    0, iDur, iDur
    asig      ATSaddnz ktime, Sfile, 5, p4
    outs      asig, asig

    ;start next instr
    if p4 < 20 then
        event_i  "i", "NoiseInBandsOfFive", iDur+1, 1, p4+5
    endif
    endin
</CsInstruments>
<CsScore>
i "AllTheNoise" 0 1
e 25
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*ATSread, ATSreadnz, ATInfo, ATSbufread, ATScross, ATSinterpread, ATSpartialtap, ATSaddnz, ATSinnoi*

## Crédits

Auteur : Alex Norman  
Seattle, Washington  
2004

# ATSBufread

ATSBufread — lit des données depuis un fichier ATS et les stocke dans une table interne de paires de données fréquence, amplitude.

## Description

*ATSBufread* lit des données depuis un fichier ATS et les stocke dans une table interne de paires de données fréquence, amplitude.

## Syntaxe

```
ATSBufread ktimepnt, kfmod, iatsfile, ipartial[, ipartialoffset, \
ipartialincr]
```

## Initialisation

*iatsfile* – le numéro ATS (n dans ats.n) ou le nom entre guillemets du fichier d'analyse créé avec *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>].

*ipartial* – nombre de partiels qui seront utilisés dans la resynthèse (le bruit a un maximum de 25 bandes).

*ipartialoffset* (facultatif) – le premier partiel utilisé (0 par défaut).

*ipartialincr* (facultatif) – fixe le pas d'incrémentation que ces opcodes de synthèse utilisent pour compter les composants bins à partir de *ipartialoffset* dans la resynthèse (1 par défaut).

## Exécution

*ktimepnt* – Le pointeur de temps en secondes utilisé comme indice sur le fichier ATS. Est utilisé pour *ATSBufread* exactement de la même manière que pour *pvoc*.

*kfmod* – une entrée pour faire une transposition de hauteur ou une modulation de fréquence sur tous les partiels synthétisés ; si aucune modulation de fréquence ou aucun changement de hauteur ne sont désirés, il faut utiliser 1 pour cette valeur.

*ATSBufread* est basé sur *pvbufread* par Richard Karpen. *ATScross*, *ATSinterpread* et *ATSpartialtap* dépendent tous de *ATSBufread* comme *pvcross* et *pvinterp* dépendent de *pvbufread*. *ATSBufread* lit des données depuis un fichier ATS et les stocke dans une table interne de paires de données fréquence, amplitude. Les données stockées par un *ATSBufread* ne sont accessibles que par d'autres générateurs unitaires, et ainsi, à cause de l'architecture de Csound, un *ATSBufread* doit se trouver avant (mais pas nécessairement directement) tout générateur unitaire dépendant. Bien que *ATSBufread* ne produise pas de données directement, il fonctionne exactement comme *ATSadd*. Il utilise un pointeur temporel (*ktimepnt*) pour indexer les données dans la durée, *ipartial*, *ipartialoffset* et *ipartialincr* pour sélectionner les partiels à stocker dans la table et *kfmod* pour pondérer les partiels en fréquence.

## Exemples

Voici un exemple de l'opcode *ATSBufread*. Il utilise le fichier *ATSBufread.csd* [examples/ATSBufread.csd].



**Exemple 66. Exemple de l'opcode ATSbufread.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc for RT audio input is needed too
; For Non-realtime output leave only the line below:
; -o ATsbufread.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; "beats.ats" and "fox.ats" are created by atsa

ktime line 0, p3, 4
ktime2 line 0, p3, 4
kline expseg 0.001, .3, 1, p3-.3, 1
kline2 expseg 0.001, p3, 3
    ATsbufread ktime2, 1, "fox.ats", 20
aout ATScross ktime, 2, "beats.ats", 1, kline, 0.001 * (4 - kline2), 180
outs aout*2, aout*2

endin

</CsInstruments>
<CsScore>
; sine wave.
f 1 0 16384 10 1

i 1 0 4
e
</CsScore>
</CsoundSynthesizer>
```

Voir aussi les exemples de *ATScross*, *ATSinterpread* et *ATSpartialtap*

## Voir aussi

*ATsread*, *ATsreadnz*, *ATsinfo*, *ATssinnoi*, *ATScross*, *ATSinterpread*, *ATSpartialtap*, *ATSadd*, *ATSaddnz*

## Crédits

Auteur : Alex Norman  
Seattle, Washington  
2004

# ATScross

ATScross — exécute une synthèse croisée à partir de fichiers d'analyse ATS.

## Description

*ATScross* utilise les données d'un fichier d'analyse ATS et d'un *ATSbufread* pour exécuter une synthèse croisée.

## Syntaxe

```
ar ATScross ktimepnt, kfmod, iatsfile, ifn, kmylev, kbuflev, ipartials \  
    [, ipartialoffset, ipartialincr]
```

## Initialisation

*iatsfile* – entier ou chaîne de caractères dénotant un fichier de contrôle dérivé de l'analyse ATS d'un signal audio. Un entier indique le suffixe d'un fichier ATS.m ; une chaîne de caractères (entre guillemets) donne un nom de fichier, ou un nom de chemin complet. Si ce n'est pas un chemin complet, le fichier est d'abord cherché dans le répertoire courant, puis dans celui qui est donné par la variable d'environnement SADIR (si elle est définie).

*ifn* – numéro de la table d'une fonction stockée contenant une onde sinusoïdale.

*ipartials* – nombre de partiels qui seront utilisés dans la resynthèse.

*ipartialoffset* (facultatif) – le premier partiel utilisé (0 par défaut).

*ipartialincr* (facultatif) – fixe le pas d'incrémentation que ces opcodes de synthèse utilisent pour compter les composants bins à partir de *ipartialoffset* dans la resynthèse (1 par défaut).

## Exécution

*ktimepnt* – Le pointeur de temps en secondes utilisé comme indice sur le fichier ATS. Est utilisé pour *ATScross* exactement de la même manière que pour *pvoc*.

*kfmod* – une entrée pour faire une transposition de hauteur ou une modulation de fréquence sur tous les partiels synthétisés ; si aucune modulation de fréquence ou aucun changement de hauteur ne sont désirés, il faut utiliser 1 pour cette valeur.

*kmylev* - pondère le composant *ATScross* du spectre de fréquence appliqué aux partiels depuis le fichier ATS indiqué par l'opcode *ATScross*. L'information du spectre de fréquence vient du fichier ATS de *ATScross*. Une valeur de 1 (et 0 pour *kbuflev*) donne le même résultat que *ATSadd*.

*kbuflev* - pondère le composant *ATSbufread* du spectre de fréquence appliqué aux partiels depuis le fichier ATS indiqué par l'opcode *ATScross*. L'information du spectre de fréquence vient du fichier ATS *ATSbufread*. Une valeur de 1 (et 0 pour *kmylev*) donne des partiels qui ont l'information de fréquence du fichier ATS donné par l'*ATScross*, mais les amplitudes imposées par les données du fichier ATS donné par *ATSbufread*.

*ATScross* utilise les données d'un fichier d'analyse ATS (indiqué par *iatsfile*) et les données d'un *ATSbufread* pour exécuter une synthèse croisée. *ATScross* utilise *ktimepnt*, *kfmod*, *ipartials*, *ipartialoffset* et

*ipartialincr* de la même manière que *ATSadd*. *ATScross* synthétise une onde sinus pour chaque partiel sélectionné par l'utilisateur et utilise la fréquence de ce partiel (après pondération en fréquence par *kfmod*) comme indice dans la table créée par *ATSbufread*. Les valeurs intermédiaires sont obtenues par interpolation. *ATScross* utilise la somme des données d'amplitude de son fichier ATS (pondérée par *kmylev*) et les données d'amplitude fournies par *ATSbufread* (pondérées par *kbuflev*) pour mettre à l'échelle l'amplitude de chaque partiel qu'il synthétise. En fixant *kmylev* à un et *kbuflev* à zéro, *ATScross* agira exactement comme *ATSadd*. En fixant *kmylev* à zéro et *kbuflev* à un, on produira un son qui aura tous les partiels sélectionnés par l'unité *ATScross*, mais avec les amplitudes fournies par *ATSbufread*. Il n'est pas nécessaire que le pointeur de temps de l'*ATSbufread* soit le même que celui de l'*ATScross*.

## Exemples

Voici un exemple de l'opcode *ATScross*. Il utilise le fichier *ATScross.csd* [examples/ATScross.csd].

### Exemple 67. Exemple de l'opcode *ATScross*.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc for RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ATScross.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; "beats.ats" and "fox.ats" are created by atsa

ktime line 0, p3, 4
ktime2 line 0, p3, 4
kline expseg 0.001, .3, 1, p3-.3, 1
kline2 expseg 0.001, p3, 3
    ATScross ktime2, 1, "fox.ats", 20
aout ATScross ktime, 2, "beats.ats", 1, kline, 0.001 * (4 - kline2), 180
outs aout*2, aout*2

endin

</CsInstruments>
<CsScore>
; sine wave.
f 1 0 16384 10 1

i 1 0 4
e
</CsScore>
</CsoundSynthesizer>
```

Cet exemple exécute une synthèse croisée à partir des deux fichiers ATS "fox.ats" et "beats.ats". Le résultat sera un son qui débute avec le profil (en fréquence) de fox.ats et se termine avec le profil de beats.ats. Toutes les fréquences d'onde sinusoïdale viennent de beats.ats. La valeur de *kbuflev* est pondérée parce que l'énergie produite en appliquant le spectre de fréquence de fox.ats aux partiels de beats.ats est très importante. Noter également que les pointeurs de temps d'*ATSbufread* (fox.ats) et d'*ATScross* (beats.ats) n'ont pas nécessairement la même valeur, ce qui permet de lire les deux fichiers ATS à des vitesses différentes.

## Voir aussi

*ATSread, ATSreadnz, ATSinfo, ATSSinnoi, ATSbufread, ATSinterpread, ATSpartialtap, ATSadd, ATSaddnz*

## Crédits

Auteur : Alex Norman  
Seattle, Washington  
2004

# ATSinfo

ATSinfo — Lit des données de l'en-tête d'un fichier ATS.

## Description

*atsinfo* lit des données de l'en-tête d'un fichier ATS.

## Syntaxe

```
idata ATSinfo iatsfile, ilocation
```

## Initialisation

*iatsfile* – le numéro ATS (n dans ats.n) ou le nom entre guillemets du fichier d'analyse créé avec *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>].

*ilocation* – indique quel champ de l'en-tête du fichier retourner. Les données de l'en-tête donnent de l'information sur les données contenues dans le reste du fichier ATS. Les valeurs possibles pour *ilocation* sont données dans la liste suivante :

0 - Taux d'échantillonnage (Hz)

1 - Taille de trame (en échantillons)

2 - Taille de fenêtre (en échantillons)

3 - Nombre de partiels

4 - Nombre de trames

5 - Amplitude maximale

6 - Fréquence maximale (Hz)

7 - Durée (secondes)

8 - Type du fichier ATS

## Exécution

Des macros peuvent améliorer la lisibilité de votre code Csound ; je donne mes définitions de macro ci-dessous :

```
#define ATS_SAMP_RATE #0#  
#define ATS_FRAME_SZ #1#  
#define ATS_WIN_SZ #2#  
#define ATS_N_PARTIALS #3#  
#define ATS_N_FRAMES #4#  
#define ATS_AMP_MAX #5#  
#define ATS_FREQ_MAX #6#  
#define ATS_DUR #7#  
#define ATS_TYPE #8#
```

*ATInfo* peut être utile pour écrire des instruments génériques qui fonctionneront avec plusieurs fichiers ATS, même s'ils ont différentes longueurs, différents nombres de partiels, etc. L'exemple 2 est une simple application de cela.

## Exemples

voici un exemple de l'opcode *ATInfo*. Il utilise le fichier *ATInfo.csd* [examples/ATInfo.csd].

### Exemple 68. Exemple de l'opcode *ATInfo*.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1 ; "fox.ats" is created by atsa

inum_partials ATInfo "fox.ats", 3
  print inum_partials

endin

</CsInstruments>
<CsScore>
i 1 0 0
e

</CsScore>
</CsoundSynthesizer>
```

Dans l'exemple ci-dessus nous utilisons *ATInfo* pour trouver le nombre de partiels dans le fichier ATS.

## Autres exemples

1. 

```
imax_freq      ATInfo "cl.ats", $ATS_FREQ_MAX
```

Dans l'exemple ci-dessus nous obtenons la valeur de la fréquence maximale du fichier ATS "cl.ats" et nous la stockons dans `imax_freq`. Nous utilisons la macro Csound `$ATS_FREQ_MAX` (définie ci-dessus), qui est équivalente au nombre 6.

2. 

```
i_npartials    ATInfo p4, $ATS_N_PARTIALS
i_dur          ATInfo p4, $ATS_DUR
ktimepnt       line 0, p3, i_dur
aout           ATSadd ktimepnt, 1, p4, 1, i_npartials
```

Dans l'exemple ci-dessus nous utilisons *ATInfo* pour retrouver la durée et le nombre de partiels dans le fichier ATS indiqué par `p4`. Avec cette information nous synthétisons les partiels au moyen d'*ATSadd*. Comme la durée et le nombre de partiels ne sont pas codés en dur, nous pouvons utiliser ce code avec n'importe quel fichier ATS.

Voici un autre exemple de l'opcode *ATInfo*. Il utilise le fichier *ATInfo-2.csd* [examples/ATInfo-2.csd].

**Exemple 69. Exemple 2 de l'opcode ATSinfo.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n      ;;no audio out
</CsOptions>
<CsInstruments>
;example by joachim heintz
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; "fox.ats" has been created by ATSanal
Sfile = "fox.ats"
isr    ATSinfo Sfile, 0
ifs    ATSinfo Sfile, 1
iws    ATSinfo Sfile, 2
inp    ATSinfo Sfile, 3
inf    ATSinfo Sfile, 4
ima    ATSinfo Sfile, 5
imf    ATSinfo Sfile, 6
id     ATSinfo Sfile, 7
ift    ATSinfo Sfile, 8
prints {{
Sample rate =    %d Hz
Frame Size =    %d samples
Window Size =    %d samples
Number of Partial = %d
Number of Frames = %d
Maximum Amplitude = %f
Maximum Frequency = %f Hz
Duration =      %f seconds
ATS file Type = %d
}}, isr, ifs, iws, inp, inf, ima, imf, id, ift
endin
</CsInstruments>
<CsScore>
i 1 0 0
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*ATSread, ATSreadnz, ATSbufread, ATScross, ATSinterpread, ATSpartialtap, ATSadd, ATSaddnz, ATSSin-*  
*noi*

## Crédits

Auteur : Alex Norman  
Seattle, Washington  
2004

# ATSinterpread

ATSinterpread — permet de déterminer l'enveloppe de fréquence de n'importe quel *ATSbufread*.

## Description

*ATSinterpread* permet de déterminer l'enveloppe de fréquence de n'importe quel *ATSbufread*.

## Syntaxe

kamp **ATSinterpread** kfreq

## Exécution

*kfreq* - une valeur de fréquence (en Hz) utilisée par *ATSinterpread* comme indice dans la table produite par un *ATSbufread*.

*ATSinterpread* prend une valeur de fréquence (*kfreq* en Hz). Cette fréquence sert à indexer les données d'un *ATSbufread*. La valeur retournée est une amplitude obtenue de l'*ATSbufread* après interpolation. *ATSinterpread* permet de déterminer l'enveloppe de fréquence de n'importe quel *ATSbufread*. Ces données peuvent être utiles pour plusieurs raisons, dont l'une est la réalisation de la synthèse croisée entre des données provenant d'un fichier ATS et des données non ATS.

## Exemples

Voici un exemple de l'opcode *ATSinterpread*. Il utilise le fichier *ATSinterpread.csd* [exemples/ATSinterpread.csd].

### Exemple 70. Exemple de l'opcode *ATSinterpread*.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc for RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ATSinterpread.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; "beats.ats" is created by atsa

ktime line 0, p3, 1.8
  ATSbufread ktime, 1, "beats.ats", 42
kamp ATSinterpread p4
aosc oscili kamp, p4, 1
  outs aosc * 25, aosc * 25

endin
```



```
</CsInstruments>
<CsScore>
; sine wave.
f 1 0 16384 10 1

i 1 0 2 100
e

</CsScore>
</CsoundSynthesizer>
```

Cet exemple montre comment utiliser *ATSinterpread*. Ici une fréquence est fournie par la partition (p4) et cette fréquence est passée à un *ATSinterpread* (avec un *ATSbufread*) correspondant. L'*ATSinterpread* utilise cette fréquence pour retourner l'amplitude correspondante basée sur le fichier ATS donné par le *ATSbufread* (beats.ats dans ce cas). Nous utilisons ensuite cette amplitude pour pondérer une onde sinus qui est synthétisée avec la même fréquence (p4). On peut étendre ceci pour inclure plusieurs ondes sinus. De cette manière il est possible de synthétiser n'importe quelle fréquence raisonnable (comprise entre la fréquence basse et la fréquence haute du fichier ATS indiqué), et de conserver la forme (en fréquence) du fichier ATS (donné par l'*ATSbufread*).

## Voir aussi

*ATSread*, *ATSreadnz*, *ATSinfo*, *ATSsinnoi*, *ATSbufread*, *ATScross*, *ATSpartialtap*, *ATSadd*, *ATSaddnz*

## Crédits

Auteur : Alex Norman  
Seattle, Washington  
2004

# ATSread

ATSread — Lit des données depuis un fichier ATS.

## Description

*ATSread* retourne l'information d'amplitude (*kamp*) et de fréquence (*kfreq*) d'un partiel spécifié contenu dans le fichier d'analyse ATS au moment indiqué par le pointeur de temps *ktimepnt*.

## Syntaxe

```
kfreq, kamp ATSread ktimepnt, iatsfile, ipartial
```

## Initialisation

*iatsfile* – le numéro ATS (n dans ats.n) ou le nom entre guillemets du fichier d'analyse créé avec *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>].

*ipartial* – le numéro du partiel d'analyse duquel seront retournés la fréquence en Hz et l'amplitude.

## Exécution

*kfreq*, *kamp* - sorties de l'unité *ATSread*. Ces valeurs représentent la fréquence et l'amplitude d'un partiel spécifique sélectionné par *ipartial*. Les informations du partiel sont dérivées d'une analyse ATS. *ATSread* interpole la fréquence et l'amplitude entre les trames dans le fichier d'analyse ATS au taux-k. La sortie dépend des données dans le fichier d'analyse et du pointeur *ktimepnt*.

*ktimepnt* – Le pointeur de temps en secondes utilisé comme indice sur le fichier ATS. Est utilisé pour *ATSread* exactement de la même manière que pour *pvoc* et *ATSadd*.

## Exemples

Voici un exemple de l'opcode *ATSread*. Il utilise le fichier *ATSread.csd* [examples/ATSread.csd].

### Exemple 71. Exemple de l'opcode *ATSread*.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ATSread.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; "beats.ats" is created by atsa

ktime line 0, p3, 2
kfreq, kamp ATSread ktime, "beats.ats", 100
```

```

aout oscili 0.8, kfreq, 1
outs aout, aout

endin

</CsInstruments>
<CsScore>
; sine wave.
f 1 0 16384 10 1

i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

Ici nous utilisons *ATSread* pour obtenir la fréquence et l'amplitude du centième partiel du fichier d'analyse ATS 'beats.ats'. Nous utilisons ces données pour piloter un oscillateur, mais nous pourrions les utiliser pour toute autre opération qui accepte une entrée au taux-k, comme la largeur de bande et la résonnance d'un filtre, etc.

Voici un autre exemple de l'opcode *ATSread*. Il utilise le fichier *ATSread-2.csd* [exemples/ATS-read-2.csd].

## Exemple 72. Exemple 2 de l'opcode *ATSread*.

```

<CsoundSynthesizer>
<CsOptions>
-odac -d -ml28
</CsOptions>
<CsInstruments>
; example by joachim heintz
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 1024, 10, 1
gSfile = "fox.ats"
giNumParts ATSinfo gSfile, 3 ;overall number of partials
giDur ATSinfo gSfile, 7 ;duration
seed 0

instr ReadOnePartial
iPartial = p4
p3 = giDur
ktime line 0, giDur, giDur
prints "Resynthesizing partial number %d.\n", iPartial
kFq, kAmp ATSread ktime, gSfile, iPartial
kAmp port kAmp, .1 ;smooth amplitudes - still not satisfactoring
aOut poscil kAmp, kFq, giSine
aOut linen aOut, 0, p3, .01 ;anti-click
outs aOut*10, aOut*10

;start next instr: normal speed, three loops, pause between loops one second
event_i "i", "MasterRand", giDur+3, 1, 1, 3, 2
endin

instr MasterRand
;random selections of 10 partials per second, overlapping
iSpeed = p4 ;speed of reading / playing
iNumLoops = p5 ;number of loops
iPause = p6 ;length of pause between loops

```

```

        prints      "Resynthesizing random partials.\n"
p3      =          (giDur/iSpeed+iPause) * iNumLoops
;start next instr: half speed, three loops, three seconds pause between loops
        event_i    "i", "MasterArp", p3+3, 1, .5, 3, 3
;loop over duration plus pause
loop:
        timeout    0, giDur/iSpeed+iPause, play
        reinit     loop

play:
gkTime  line       0, giDur/iSpeed, giDur ;start time from 0 in each loop
kTrig   metro      10 ;10 new partials per second
;call subinstrument if trigger and no pause
if kTrig == 1 && gkTime < giDur then
kPart   random     1, giNumParts+.999
        event      "i", "PlayRand", 0, 1, int(kPart)
endif

        endin

        instr MasterArp
;arpeggio-like reading and playing of partials
iSpeed  =          p4 ;speed of reading / playing
iNumLoops =        p5 ;number of loops
iPause  =          p6 ;length of pause between loops
        prints     "Arpeggiating partials.\n"
p3      =          (giDur/iSpeed+iPause) * iNumLoops
loop:
        timeout    0, giDur/iSpeed+iPause, play
        reinit     loop

play:
gkTime  line       0, giDur/iSpeed, giDur
kArp    linseg     1, (giDur/iSpeed)/2, giNumParts, (giDur/iSpeed)/2, 1 ;arp up and down
kTrig   metro      10 ;10 new partials per second
if kTrig == 1 && gkTime < giDur then
        event      "i", "PlayArp", 0, 5, int(kArp)
endif

;exit csound when finished
        event_i    "i", "End", p3+5, 1

        endin

        instr PlayRand
iPartial =          p4
kFq,kAmp ATsread   gkTime, gSfile, iPartial
kamp     port      kAmp, .15 ;smooth amplitudes
aOut     poscil    kAmp, kFq, giSine
aOut     linen     aOut, .01, p3, .01
        outs      aOut, aOut

        endin

        instr PlayArp
kCount   init      1 ;k-cycle
iPartial =          p4
kFq,kAmp ATsread   gkTime, gSfile, iPartial
if kCount == 1 then ;get freq from first k-cycle
kModFq   =          kFq
;avoid to go with 0 Hz as this blocks the mode filter
if kModFq == 0 then
        turnoff
endif
endif
iVol     random    -42, -12 ;db
iOffset  random    .01, .1 ;no too regularly ...
aImp     mpulse    ampdb(iVol), p3, iOffset
iQ       random    500, 5000
aOut     mode      aImp, kModFq, iQ
aOut     linen     aOut, 0, p3, p3/3

```

```
        outs      aOut, aOut
kCount    =      2
    endin

    instr End
        exitnow
    endin
</CsInstruments>
<CsScore>
i "ReadOnePartial" 0 1 10
e 999
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*AThreadnz, ATInfo, ATBufread, ATScross, ATInterpread, ATPartialtap, ATSadd, ATSaddnz, ATSin-*  
*noi*

## Crédits

Auteur : Alex Norman  
Seattle, Washington  
2004

# ATSreadnz

ATSreadnz — lit des données depuis un fichier ATS.

## Description

*ATSreadnz* retourne l'énergie (*kenergy*) d'une bande de bruit spécifiée par l'utilisateur (1-25 bandes) à la date indiquée par le pointeur de temps *ktimepnt*.

## Syntaxe

```
kenergy ATSreadnz ktimepnt, iatsfile, iband
```

## Initialisation

*iatsfile* – le numéro ATS (n dans ats.n) ou le nom entre guillemets du fichier d'analyse créé avec *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>].

*iband* – le numéro de la bande de bruit dont il faut retourner les données d'énergie.

## Exécution

*kenergy* est la sortie contenant l'énergie interpolée linéairement de la bande de bruit indiquée par *iband*. La sortie dépend des données dans le fichier d'analyse et de *ktimepnt*.

*ktimepnt* – Le pointeur de temps en secondes utilisé comme indice sur le fichier ATS. Est utilisé pour *ATSreadnz* exactement de la même manière que pour *pvoc* et *ATSadd*.

*ATSsaddnz* lit depuis un fichier ATS et resynthétise le bruit à partir des données d'énergie du bruit contenues dans le fichier ATS. Il utilise une fonction *randi* modifiée pour créer du bruit à bande limitée et le module avec une table d'onde fournie par l'utilisateur (une période d'une onde cosinus), pour synthétiser une sélection spécifiée par l'utilisateur de bandes de fréquence. Il est nécessaire de moduler le bruit pour placer le bruit à bande limitée au bon endroit dans le spectre de fréquence.

Une analyse ATS diffère d'une analyse *pvanal* du fait qu'ATS trace les partiels et calcule l'énergie du bruit dans le son étant analysé. Pour plus d'information sur l'analyse ATS, lire la description de Juan Pampin sur la page web *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>].

## Exemples

```
ktime   line      2.5, p3, 0
kenergy ATSreadnz ktime, "clarinet.ats", 5
```

Ici nous extrayons la bande d'énergie 5 du bruit du fichier d'analyse ATS 'clarinet.ats'. Nous lisons à l'envers depuis 2.5 secondes vers le début du fichier d'analyse. On peut l'utiliser pour synthétiser du bruit comme cela :

```
anoise randi      sqrt(kenergy), 55
aout   oscili    40000000000000000000000000000000, 455, 2
aout   =          aout * anoise
```

La table de fonction 2 utilisée dans l'oscillateur est une onde cosinus, qui est nécessaire pour déplacer le bruit à bande limitée au bon endroit dans le spectre de fréquence. La fonction *randi* crée une bande de fréquence centrée sur 0 Hz qui a une largeur de bande d'environ 110 Hz ; en la multipliant par une onde cosinus on la déplace pour qu'elle soit centrée à 455 Hz, qui est la fréquence centrale de la 5ème bande critique de bruit. Ce n'est qu'un exemple pour synthétiser du bruit qui serait mieux réalisé avec *ATSaddnz*, à moins que l'on ne désire utiliser son propre algorithme de synthèse de bruit. Peut-être peut-on utiliser l'énergie du bruit pour autre chose comme appliquer une petite quantité de tremblement à des partiels spécifiques ou contrôler quelque chose sans aucun rapport avec le son source ?

Voici un autre exemple de l'opcode *ATSreadnz*. Il utilise le fichier *ATSreadnz.csd* [exemples/ATS-readnz.csd].

### Exemple 73. Un autre exemple de l'opcode *ATSreadnz*.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ATSreadnz.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; "beats.ats" is created by atsa

ktime line 0, p3, 2
kenergy ATSreadnz ktime, "beats.ats", 2
anoise randi kenergy, 500
aout oscili 0.005, 455, 1
aout = aout * anoise
outs aout, aout
endin

</CsInstruments>
<CsScore>
; cosine wave
f 1 0 16384 11 1 1

i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*ATSread*, *ATSinfo*, *ATSbufread*, *ATScross*, *ATSinterpread*, *ATSpartialtap*, *ATSadd*, *ATSaddnz*, *ATSinnoi*

## Crédits

Auteur : Alex Norman  
 Seattle, Washington  
 2004

# ATSpartialtap

ATSpartialtap — retourne une paire fréquence, amplitude à partir d'un opcode *ATSBufread*.

## Description

*ATSpartialtap* prend un numéro de partiel et retourne une paire fréquence, amplitude. Les données de fréquence et d'amplitude proviennent d'un opcode *ATSBufread*.

## Syntaxe

```
kfrq, kamp ATSpartialtap ipartialnum
```

## Initialisation

*ipartialnum* - indique le partiel que l'opcode *ATSpartialtap* doit lire à partir d'un *ATSBufread*.

## Exécution

*kfrq* - retourne la valeur de fréquence du partiel demandé.

*kamp* - retourne la valeur d'amplitude du partiel demandé.

*ATSpartialtap* prend un numéro de partiel et retourne une paire fréquence, amplitude. Les données de fréquence et d'amplitude proviennent d'un opcode *ATSBufread* C'est une version restreinte d'*ATSread*, car chaque opcode *ATSread* a son propre pointeur de temps indépendant et *ATSpartialtap* est restreint aux données données par un *ATSBufread*. Cette simplicité est son point fort.

## Exemples

Voici un exemple de l'opcode *ATSpartialtap*. Il utilise le fichier *ATSpartialtap.csd* [exemples/ATSpartialtap.csd].

### Exemple 74. Exemple de l'opcode ATSpartialtap.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc for RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ATSpartialtap.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; "beats.ats" is created by atsa

ktime line 0, p3, 2
```



```

    ATSbufread ktime, 1, "beats.ats", 30
    kfreq1, kam1 ATSpartialtap 5
    kfreq2, kam2 ATSpartialtap 20
    kfreq3, kam3 ATSpartialtap 30

    aout1 oscil kam1, kfreq1, 1
    aout2 oscil kam2, kfreq2, 1
    aout3 oscil kam3, kfreq3, 1
    aout = (aout1+aout2+aout3)*10 ; amplify some more
    outs aout, aout

    endin

</CsInstruments>
<CsScore>
; sine wave.
f 1 0 16384 10 1
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

Cet exemple utilise un *ATSpartialtap* et un *ATSbufread* pour lire les partiels 5, 20 et 30 de 'beats.ats'. On pourrait utiliser ces amplitudes et ces fréquences pour resynthétiser ces partiels ou pour faire quelque chose de tout à fait différent.

## Voir aussi

*ATSread, ATSreadnz, ATSinfo, ATSinnoi, ATSbufread, ATScross, ATSinterpread, ATSadd, ATSaddnz*

## Crédits

Auteur : Alex Norman  
 Seattle, Washington  
 2004

# ATSsinnoi

ATSsinnoi — Utilise les données d'un fichier d'analyse ATS pour réaliser une resynthèse.

## Description

*ATSsinnoi* lit les données d'un fichier ATS et utilise cette information pour synthétiser à la fois des sinusoïdes et du bruit.

## Syntaxe

```
ar ATSsinnoi ktimepnt, ksinlev, knzlev, kfmod, iatsfile, ipartials \  
    [, ipartialoffset, ipartialincr]
```

## Initialisation

*iatsfile* – le numéro ATS (n dans ats.n) ou le nom entre guillemets du fichier d'analyse créé avec *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>].

*ipartials* – nombre de partiels qui seront utilisés dans la resynthèse (le bruit a un maximum de 25 bandes).

*ipartialoffset* (optional) – (facultatif) – le premier partiel utilisé (0 par défaut).

*ipartialincr* (optional) – (facultatif) – fixe le pas d'incrémentation que ces opcodes de synthèse utilisent pour compter les composants bins à partir de *ipartialoffset* dans la resynthèse (1 par défaut).

## Exécution

*ktimepnt* – Le pointeur de temps en secondes utilisé comme indice sur le fichier ATS. Est utilisé pour *ATSsinnoi* exactement de la même manière que pour *pvoc*.

*ksinlev* - contrôle le niveau des sinus dans le générateur unitaire *ATSsinnoi*. Une valeur de 1 donne des ondes sinus à plein volume.

*knzlev* - contrôle le niveau des composants du bruit dans le générateur unitaire *ATSsinnoi*. Une valeur de 1 donne du bruit à plein volume.

*kfmod* – une entrée pour faire une transposition de hauteur ou une modulation de fréquence sur tous les partiels synthétisés ; si aucune modulation de fréquence ou aucun changement de hauteur ne sont désirés, il faut utiliser 1 pour cette valeur.

*ATSsinnoi* lit les données d'un fichier ATS et utilise cette information pour synthétiser à la fois des sinusoïdes et du bruit. L'énergie du bruit pour chaque bande est distribuée également entre les partiels qui tombent dans cette bande. Chaque partiel est ensuite synthétisé, avec sa composante de bruit. Chaque composante de bruit est ensuite modulée par le partiel correspondant pour être placée au bon endroit dans le spectre de fréquence. Les niveaux du bruit et des partiels sont contrôlables individuellement. Voir la page web *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>] pour plus d'information sur la synthèse sinnoi. Une analyse ATS diffère d'une analyse pvanal du fait qu'ATS trace les partiels et calcule l'énergie du bruit dans le son étant analysé. Pour plus d'information sur l'analyse ATS, lire la description de Juan Pampin sur la page web *ATS* [<http://www-ccrma.stanford.edu/~juan/ATS.html>].

## Exemples

```
ktime    line        0, p3, 2.5
```

```
asig      ATSSinnoi ktime, 1, 1, 1, "beats.ats", 42
```

Nous synthétisons ici à la fois le bruit et les ondes sinus (les 42 partiels) contenus dans "beats.ats". Les volumes relatifs du bruit et des partiels sont inchangés (chacun est fixé à 1).

Voici un autre exemple de l'opcode ATSSinnoi. Il utilise le fichier *ATSSinnoi.csd* [examples/ATSSinnoi.csd].

### Exemple 75. Exemple de l'opcode ATSSinnoi.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ATSSinnoi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; "beats.ats" is created by atsa

ktime line 0, p3, 2
knzfade expon 0.001, p3, 2
aout ATSSinnoi ktime, 1, knzfade, 1, "beats.ats", 150
outs aout*2, aout*2 ;amplify some more
endin

</CsInstruments>
<CsScore>

i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Cet exemple reprend le précédent mais en utilisant une enveloppe pour contrôler *knzlev* (le niveau de bruit). Cela donne le son de "beats.wav" dont la composante de bruit apparaît progressivement durant la durée de la note.

## Voir aussi

*ATSread, ATSreadnz, ATSinfo, ATShufread, ATScross, ATSinterpread, ATSpartialtap, ATSadd, ATSaddnz*

Voici un autre exemple de l'opcode ATSSinnoi. Il utilise le fichier *ATSSinnoi-2.csd* [examples/ATSSinnoi-2.csd].

### Exemple 76. Exemple 2 de l'opcode ATSSinnoi.

```
<CsoundSynthesizer>
<CsOptions>
-odac -d -m128
</CsOptions>
<CsInstruments>
;example by joachim heintz
```

```

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine    ftgen    0, 0, 1024, 10, 1
gSfile    =        "fox.ats"
giNumParts ATSinno gSfile, 3 ;overall number of partials
giDur     ATSinno  gSfile, 7 ;duration
seed      0

instr PlayList
event_i "i", "PlayAll", 0, 1, 1, 0, .5 ;sine only, half speed
event_i "i", "PlayAll", giDur*2+1, 1, 0, 1, .5 ;noise only
event_i "i", "PlayAll", giDur*4+2, 1, .5, .5, .5 ;half sine, half noise
endin

instr PlayAll
iSinAmnt = p4 ;sine amount (0-1)
iNzAmnt  = p5 ;noise amount (0-1)
iSpeed   = p6 ;speed
p3       = giDur/iSpeed
ktime    line 0, giDur/iSpeed, giDur
prints   "Resynthesizing all partials with tone = %.1f and noise = %.1f.\n", iSinAmnt, iNzAmnt
aOut     ATSinnoi ktime, iSinAmnt, iNzAmnt, 1, gSfile, giNumParts
outs     aOut, aOut
endin

instr PlayBand
iOffset  = p4 ;offset in partials
iSpeed   = p5 ;speed
p3       = giDur/iSpeed
ktime    line 0, giDur/iSpeed, giDur
prints   "Resynthesizing partials %d to %d with related noise.\n", iOffset+1, iOffset+10
aOut     ATSinnoi ktime, 1, 1, 1, gSfile, 10, iOffset, 1
outs     aOut, aOut
;call itself again
if iOffset < giNumParts - 20 then
    event_i "i", "PlayBand", giDur/iSpeed+1, 1, iOffset+10, iSpeed
endif
endin

instr PlayWeighted
;sine amount, noise amount and speed are varying
kSinAmnt randomi 0, 1, 1, 3
kNzAmnt  = 1-kSinAmnt
kSpeed   randomi .01, .3, 1, 3
async    init 0
atime, aEnd syncphasor kSpeed/giDur, async
kTrig    metro 100
kEnd     max_k aEnd, kTrig, 1 ;1 if phasor signal crosses zero
ktime    downsamp atime
aOut     ATSinnoi ktime*giDur, kSinAmnt, kNzAmnt, 1, gSfile, giNumParts
outs     aOut, aOut
;exit if file is at the end
if kEnd == 1 then
    event "i", "End", 0, 1
endif
endin

instr End
exitnow
endin

</CsInstruments>

```

```
<CsScore>  
i "PlayList" 0 1  
i "PlayBand" 20 1 0 .5  
i "PlayWeighted" 110 100  
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : Alex Norman  
Seattle, Washington  
2004

# babo

babo — Une réverbération par modèle physique.

## Description

*babo* est une abréviation pour *ball-within-the-box* (balle dans la boîte). C'est un modèle physique de réverbération basé sur le papier de Davide Rocchesso "The Ball within the Box: a sound-processing metaphor", Computer Music Journal, Vol 19, N.4, pp.45-47, Hiver 1995.

La géométrie du résonateur peut être définie, de même que certaines caractéristiques de la réponse, la position de l'auditeur dans le résonateur et la position de la source sonore.

## Syntaxe

```
a1, a2 babo asig, ksrcx, ksrcy, ksrcz, irx, iry, irz [, idiff] [, ifno]
```

## Initialisation

*irx, iry, irz* -- les coordonnées géométriques du résonateur (longueur des côtés en mètres).

*idiff* -- est le coefficient de diffusion sur les murs, qui contrôle l'importance de la diffusion (0-1, où 0 = pas de diffusion, 1 = diffusion maximale - vaut 1 par défaut).

*ifno* -- fonction des valeurs pour expert : un numéro de fonction contenant tous les paramètres additionnels du résonateur. C'est normalement une fonction de type GEN2 en mode non normalisé. Les paramètres sont :

- *decay* -- décroissance principale du résonateur (0.99 par défaut)
- *hydecay* -- décroissance des hautes fréquences du résonateur (0.1 par défaut)
- *rcvx, rcvy, rcvz* -- coordonnées de la position de l'auditeur (en mètres ; 0,0,0 est le centre du résonateur)
- *rdistance* -- la distance en mètres entre deux récepteurs (vos oreilles, par exemple - 0.3 par défaut)
- *direct* -- l'atténuation du signal direct (0-1, 0.5 par défaut)
- *early\_diff* -- le coefficient d'atténuation des premières réflexions (0-1, 0.8 par défaut)

## Exécution

*asig* -- le signal en entrée

*ksrcx, ksrcy, ksrcz* -- les coordonnées virtuelles de la source sonore (le signal en entrée). Elles peuvent changer au taux-k et fournissent toutes les variations nécessaires en terme de réponse du résonateur.

## Exemples

Voici un exemple simple de l'opcode babo. Il utilise les fichiers *babo.csd* [examples/babo.csd] et *beats.wav* [examples/beats.wav].

## Exemple 77. Un exemple simple de l'opcode babo.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime output leave only the line below:
; -o babo.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Nicola Bernardini */

sr = 44100
ksmps = 32
nchnls = 2

; minimal babo instrument
;
instr 1
    ix      = p4 ; x position of source
    iy      = p5 ; y position of source
    iz      = p6 ; z position of source
    ixsize  = p7 ; width of the resonator
    iysize  = p8 ; depth of the resonator
    izsize  = p9 ; height of the resonator

    ainput soundin "beats.wav"

    al,ar babo    ainput*0.7, ix, iy, iz, ixsize, iysize, izsize

    outs     al,ar

endin

</CsInstruments>
<CsScore>

/* Written by Nicola Bernardini */
; simple babo usage:
;
;p4      : x position of source
;p5      : y position of source
;p6      : z position of source
;p7      : width of the resonator
;p8      : depth of the resonator
;p9      : height of the resonator
;
i 1 0 20 6 4 3 14.39 11.86 10
;          ^^^^^^  ^^^^^^^^^^^^^^^
;          |||/|||  ++++++optimal room dims according to
;          |||/|||  Milner and Bernard JASA 85(2), 1989
;          ++++++source position
e

</CsScore>
</CsoundSynthesizer>
```

Voici un exemple avancé de l'opcode babo. Il utilise les fichiers *babo\_expert.csd* [examples/babo\_expert.csd] et *beats.wav* [examples/beats.wav].

## Exemple 78. Un exemple avancé de l'opcode babo.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o babo_expert.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Nicola Bernardini */

sr = 44100
ksmps = 32
nchnls = 2

; full blown babo instrument with movement
;
instr 2
  ixstart = p4 ; start x position of source (left-right)
  ixend = p7 ; end x position of source
  iystart = p5 ; start y position of source (front-back)
  iyend = p8 ; end y position of source
  izstart = p6 ; start z position of source (up-down)
  izend = p9 ; end z position of source
  ixsize = p10 ; width of the resonator
  iysize = p11 ; depth of the resonator
  izsize = p12 ; height of the resonator
  idiff = p13 ; diffusion coefficient
  iexpert = p14 ; power user values stored in this function

  ainput soundin "beats.wav"
  ksource_x line ixstart, p3, ixend
  ksource_y line iystart, p3, iyend
  ksource_z line izstart, p3, izend

  al,ar babo ainput*0.7, ksource_x, ksource_y, ksource_z, ixsize, iysize, izsize, idiff, iexpert

  outs al,ar

endin

</CsInstruments>
<CsScore>

/* Written by Nicola Bernardini */
; full blown instrument
;p4 : start x position of source (left-right)
;p5 : end x position of source
;p6 : start y position of source (front-back)
;p7 : end y position of source
;p8 : start z position of source (up-down)
;p9 : end z position of source
;p10 : width of the resonator
;p11 : depth of the resonator
;p12 : height of the resonator
;p13 : diffusion coefficient
;p14 : power user values stored in this function

; decay hidecay rx ry rz rdistance direct early_diff
f1 0 8 -2 0.95 0.95 0 0 0 0.3 0.5 0.8 ; brighter
f2 0 8 -2 0.95 0.5 0 0 0 0.3 0.5 0.8 ; default (to be set as)
f3 0 8 -2 0.95 0.01 0 0 0 0.3 0.5 0.8 ; darker

```



## Crédits

Nouveau dans la version 4.09 de Csound

# balance

*balance* — Ajuste un signal audio selon les valeurs d'un autre.

## Description

La valeur efficace de *asig* peut être interrogée, fixée ou ajustée pour s'adapter à celle d'un signal de comparaison.

## Syntaxe

```
ares balance asig, acomp [, ihp] [, iskip]
```

## Initialisation

*ihp* (facultatif) -- point à mi-puissance (en Hz) d'un filtre passe-bas interne spécial. La valeur par défaut est 10.

*iskip* (facultatif, 0 par défaut) -- disposition initiale de l'espace de données interne (voir *reson*). La valeur par défaut est 0.

## Exécution

*asig* -- signal audio en entrée

*acomp* -- le signal de comparaison

*balance* restitue une version de *asig*, dont l'amplitude a été modifiée de façon à ce que sa valeur efficace soit égale à celle d'un signal de comparaison *acomp*. Ainsi un signal qui a subi une perte de puissance (par exemple en traversant un banc de filtres) peut être restauré en l'ajustant, par exemple, à sa propre source. Il faut noter que *gain* et *balance* n'effectuent que des modifications d'amplitude, les signaux de sortie ne subissant aucune autre altération.

## Exemples

Voici un exemple de l'opcode *balance*. Il utilise le fichier *balance.csd* [examples/balance.csd].

### Exemple 79. Exemple de l'opcode *balance*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o balance.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
; Generate a band-limited pulse train.
asrc buzz 0.9, 440, sr/440, 1

; Send the source signal through 2 filters.
a1 reson asrc, 1000, 100
a2 reson a1, 3000, 500

; Balance the filtered signal with the source.
afin balance a2, asrc
outs afin, afin

endin

</CsInstruments>
<CsScore>
;sine wave.
f 1 0 16384 10 1

i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*balance2, gain, rms*

# balance2

balance2 — Ajuste un signal audio selon les valeurs d'un autre.

## Description

La valeur efficace de *asig* peut être interrogée, fixée ou ajustée pour s'adapter à celle d'un signal de comparaison.

## Syntaxe

```
ares balance2 asig, acomp [, ihp] [, iskip]
```

## Initialisation

*ihp* (facultatif) -- point à mi-puissance (en Hz) d'un filtre passe-bas interne spécial. La valeur par défaut est 10.

*iskip* (facultatif, 0 par défaut) -- disposition initiale de l'espace de données interne (voir *reson*). La valeur par défaut est 0.

## Exécution

*asig* -- signal audio en entrée

*acomp* -- le signal de comparaison

*balance2* restitue une version de *asig*, dont l'amplitude a été modifiée de façon à ce que sa valeur efficace soit égale à celle d'un signal de comparaison *acomp*. Ainsi un signal qui a subi une perte de puissance (par exemple en traversant un banc de filtres) peut être restauré en l'ajustant, par exemple, à sa propre source. Il faut noter que *gain* et *balance2* n'effectuent que des modifications d'amplitude, les signaux de sortie ne subissant aucune autre altération.

Noter que *balance2* est semblable à *balance* sauf que le gain est recalculé pour chaque échantillon plutôt que d'interpoler des valeurs de taux-k.

## Exemples

Voici un exemple de l'opcode *balance2*. Il utilise le fichier *balance2.csd* [examples/balance2.csd].

### Exemple 80. Exemple de l'opcode *balance2*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o balance.wav -W ;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

  sr = 44100
  ksmpr = 32
  nchnls = 2
  odbfs = 1

  instr 1
    ; Generate a band-limited pulse train.
    asrc buzz 0.9, 440, sr/440, 1

    ; Send the source signal through 2 filters.
    a1 reson asrc, 1000, 100
    a2 reson a1, 3000, 500

    ; Balance the filtered signal with the source.
    afin balance2 a2, asrc
        outs afin, afin

  endin

</CsInstruments>
<CsScore>
  ;sine wave.
  f 1 0 16384 10 1

  i 1 0 2
  e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*balance, gain, rms*

## Crédits

Auteur : Victor Lazzarini;  
Mars 2018

Nouveau dans Csound version 6.11

# bamboo

bamboo — Modèle semi-physique d'un son de bambou.

## Description

*bamboo* est un modèle semi-physique d'un son de bambou. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntaxe

```
ares bamboo kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \  
    [, ifreq1] [, ifreq2]
```

## Initialisation

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 1,25.

*idamp* (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

$\text{damping\_amount} = 0,9999 + (\text{idamp} * 0,002)$

La valeur par défaut de *damping\_amount* est 0,9999 ce qui signifie que la valeur par défaut de *idamp* est 0. Le maximum de *damping\_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 0,05.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

*imaxshake* (facultatif, 0 par défaut) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

*ifreq* (facultatif) -- la fréquence de résonance principale. La valeur par défaut est 2800.

*ifreq1* (facultatif) -- la première fréquence de résonance. La valeur par défaut est 2240.

*ifreq2* (facultatif) -- La seconde fréquence de résonance. La valeur par défaut est 3360.

## Exécution

*kamp* -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

## Exemples

Voici un exemple de l'opcode bamboo. Il utilise le fichier *bamboo.csd* [examples/bamboo.csd].

### Exemple 81. Exemple de l'opcode bamboo.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o bamboo.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1
asig  bamboo p4, 0.01,0, 0, 0, 8000
      outs asig, asig

endin

</CsInstruments>
<CsScore>

i1 0 1 20000
i1 2 1 20000
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*dripwater, guiro, sleighbells, tambourine*

## Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)

Adapté par John ffitch

Université de Bath, Codemist Ltd.

Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

# barmodel

barmodel — Crée un timbre similaire à une barre de métal frappée.

## Description

La sortie audio est un timbre semblable à celui d'une barre de métal frappée, mettant en œuvre un modèle physique développé à partir de la résolution de l'équation différentielle partielle. On contrôle les conditions aux limites ainsi que les caractéristiques de la barre.

## Syntaxe

```
ares barmodel kbcL, kbcR, iK, ib, kscan, iT30, ipos, ivel, iwid
```

## Initialisation

*iK* -- paramètre de raideur sans dimension. Si ce paramètre est négatif, l'initialisation est ignorée et l'état précédent de la barre est prolongé.

*ib* -- paramètre de perte des hautes fréquences (à garder petit).

*iT30* -- temps de décroissance à 30 db en secondes.

*ipos* -- position le long de la barre où a lieu la frappe.

*ivel* -- vitesse de frappe normalisée.

*iwid* -- largeur spatiale de la frappe.

## Exécution

Une note est jouée sur une barre métallique, avec les arguments suivants.

*kbcL* -- Condition aux limites à l'extrémité gauche de la barre (1 fixée, 2 pivotante, 3 libre).

*kbcR* -- Condition aux limites à l'extrémité droite de la barre (1 fixée, 2 pivotante, 3 libre).

*kscan* -- Taux de lecture de la position de sortie.

Noter que le changement des conditions aux limites pendant l'exécution peut provoquer des bruits parasites ; cette possibilité est offerte à titre expérimental. L'utilisation d'un *kscan* différent de zéro peut produire des réintroductions apparentes du son à cause de la modulation.

## Exemples

Voici un exemple de l'opcode barmodel. Il utilise le fichier *barmodel.csd* [examples/barmodel.csd].

### Exemple 82. Exemple de l'opcode barmodel.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.



```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o barmodel.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr      =      44100
kr      =      4410
ksmps   =      10
nchnls  =      1

; Instrument #1.
instr 1
  aq      barmodel  1, 1, p4, 0.001, 0.23, 5, p5, p6, p7
          out      aq
endin

</CsInstruments>
<CsScore>

i1 0.0 0.5 3 0.2 500 0.05
i1 0.5 0.5 -3 0.3 1000 0.05
i1 1.0 0.5 -3 0.4 1000 0.1
i1 1.5 4.0 -3 0.5 800 0.05
e
/* barmodel */

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Stefan Bilbao  
 Université d'Edimbourg, UK  
 Auteur : John ffitch  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 5.01 de Csound

# bbcutm

bbcutm — Extrait des segments dans le style breakbeat à partir d'un flux audio mono.

## Description

Le BreakBeat Cutter extrait automatiquement des segments à partir d'un flux audio dans le style des manipulations du "drum and bass/jungle breakbeat". Il y a deux versions, pour les sources mono (*bbcutm*) ou stéréo (*bbcuts*). Bien que basé à l'origine sur les coupures breakbeat, l'opcode peut être appliqué à n'importe quel type de source audio.

La séquence de coupure typique sur une mesure subdivisée en croches serait

3+ 3R + 2

dans laquelle nous prenons un bloc de trois unités au début de la source, le répétons, puis deux unités venant des 7èmes et 8èmes croches de la source.

Nous parlons de restituer des phrases (une séquence de coupures avant d'atteindre une nouvelle phrase au début d'une mesure) et des unités (comme subdivisions des notes).

L'opcode donne un rendu plus vivant lorsqu'on utilise simultanément plusieurs versions synchronisées.

## Syntaxe

```
al bbcutm asource, ibps, isubdiv, ibarlength, iphrasebars, inumrepeats \  
    [, istutterspeed] [, istutterchance] [, ienvchoice ]
```

## Initialisation

*ibps* -- Tempo pour les coupures, en pulsations par seconde.

*isubdiv* -- Unité de subdivision pour une mesure. Par exemple 8 désigne la croche (dans une mesure à 4/4).

*ibarlength* -- Nombre de pulsations par mesure. Il vaut 4 pour la mesure par défaut à 4/4.

*iphrasebars* -- Les coupures sont générées par phrases, chaque phrase durant *iphrasebars*.

*inumrepeats* -- Dans une utilisation normale, l'algorithme permet une répétition supplémentaire d'une coupure donnée à la fois. Ce paramètre permet de modifier ce comportement. La valeur 1 représente la norme d'une répétition supplémentaire. 0 supprime la répétition et l'on obtient la source originale excepté pour l'enveloppe et le stuttering.

*istutterspeed* -- (facultatif, par défaut=1) Le stutter peut être un multiple entier de la vitesse de subdivision. Par exemple, si *isubdiv* vaut 8 (croches) et *istutterspeed* vaut 2, le stutter est en doubles croches (= subdiv de 16). La valeur par défaut est 1.

*istutterchance* -- (facultatif, par défaut=0) La fin d'une phrase a cette probabilité de devenir l'unité de répétition du stutter (0,0 à 1,0). La valeur par défaut est 0.

*ienvchoice* -- (facultatif, par défaut=1) Choisir 1 pour l'activer (enveloppe exponentielle pour les grains de coupure) ou 0 pour le désactiver. S'il est désactivé, on entendra des clics, mais ça peut donner de bons résultats bruiteux, en particulier avec les sources percussives. La valeur par défaut est 1, actif.

## Exécution

*asource* -- Le signal sonore à couper. Cette version fonctionne en temps réel sans connaissance des événements audio futurs.

## Exemples

Voici un exemple de l'opcode *bbcutm*. Il utilise les fichiers *bbcutm.csd* [examples/bbcutm.csd] et *beats.wav* [examples/beats.wav].

### Exemple 83. Un exemple simple de l'opcode *bbcutm*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o bbcutm.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - Play an audio file normally.
instr 1
  asource soundin "beats.wav"
  out asource
endin

; Instrument #2 - Cut-up an audio file.
instr 2
  asource soundin "beats.wav"

  ibps = 4
  isubdiv = 8
  ibarlength = 4
  iphrasebars = 1
  inumrepeats = 2

  a1 bbcutm asource, ibps, isubdiv, ibarlength, iphrasebars, inumrepeats

  out a1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 3 2
e
```

```
</CsScore>
</CsoundSynthesizer>
```

Voici quelques exemples plus avancés ...

### Exemple 84. Premiers pas - versions mono et stéréo

```
<CsoundSynthesizer>
<CsInstruments>
sr      =      44100
kr      =      4410
ksmps   =      10
nchnls  =      2

instr 1
  asource diskin "break7.wav",1,0,1 ; a source breakbeat sample, wraparound lest it stop!

  ; cuts in eighth notes per 4/4 bar, up to 4 bar phrases, up to 1
  ; repeat in total (standard use) rare stuttering at 16 note speed,
  ; no enveloping
  asig bbcutm asource, 2.6937, 8, 4, 4, 1, 2, 0.1, 0

  outs      asig,asig
endin

instr 2 ;stereo version
  asource1,asource2 diskin "break7stereo.wav", 1, 0, 1 ; a source breakbeat sample, wraparound lest it stop!

  ; cuts in eighth notes per 4/4 bar, up to 4 bar phrases, up to 1
  ; repeat in total (standard use) rare stuttering at 16 note speed,
  ; no enveloping
  asig1,asig2 bbcuts asource1, asource2, 2.6937, 8, 4, 4, 1, 2, 0.1, 0

  outs asig1,asig2
endin

</CsInstruments>
<CsScore>
i1 0 10
i2 11 10
e
</CsScore>
</CsoundSynthesizer>
```

### Exemple 85. Breaks multiples simultanés synchronisés

```
<CsoundSynthesizer>
<CsInstruments>
sr      =      44100
kr      =      4410
ksmps   =      10
nchnls  =      2

instr 1
  ibps = 2.6937
  iplaybackspeed = ibps/p5
  asource diskin p4, iplaybackspeed, 0, 1

  asig bbcutm asource, 2.6937, p6, 4, 4, p7, 2, 0.1, 1

  out asig
endin
```

```

</CsInstruments>
<CsScore>

; source      bps cut repeats
i1 0 10 "break1.wav" 2.3 8   2 //2.3 is the source original tempo
i1 0 10 "break2.wav" 2.4 8   3
i1 0 10 "break3.wav" 2.5 16  4
e
</CsScore>
</CsoundSynthesizer>

```

**Exemple 86. Coupure de n'importe quelle source audio ancienne - des bruits bien plus intéressants que ceux-ci sont possibles !**

```

<CsoundSynthesizer>
<CsInstruments>
sr      =      44100
kr      =      4410
ksmps   =      10
nchnls  =      2

instr 1
  asource oscil 20000, 70, 1
  ; ain, bps, subdiv, barlength, phrasebars, numrepeats,
  ;stutterspeed, stutterchance, envelopingon
  asig bbcutm asource, 2, 32, 1, 1, 2, 4, 0.6, 1
  outs asig
endin

</CsInstruments>
<CsScore>
f1 0 256 10 1
i1 0 10
e
</CsScore>
</CsoundSynthesizer>

```

**Exemple 87. Faux stuttering constant, impossible car on ne peut appliquer le stutter que dans la dernière demie-mesure, pourrait faire un paramètre optionnel supplémentaire de stuttering**

```

<CsoundSynthesizer>
<CsInstruments>
sr      =      44100
kr      =      4410
ksmps   =      10
nchnls  =      2

instr 1
  asource disk "break7.wav", 1, 0, 1

  ;16th note cuts- but cut size 2 over half a beat.
  ;each half beat will either survive intact or be turned into
  ;the first sixteenth played twice in succession

  asig bbcutm asource, 2.6937, 2, 0.5, 1, 2, 2, 1.0, 0
  outs asig
endin

</CsInstruments>
<CsScore>
i1 0 30
e

```

```
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*bbcuts*

## Crédits

Auteur : Nick Collins  
Londres  
Août 2001

Nouveau dans la version 4.13

# bbcuts

bbcuts — Extrait des segments dans le style breakbeat à partir d'un flux audio stéréo.

## Description

Le BreakBeat Cutter extrait automatiquement des segments à partir d'un flux audio dans le style des manipulations du "drum and bass/jungle breakbeat". Il y a deux versions, pour les sources mono (*bbcutm*) ou stéréo (*bbcuts*). Bien que basé à l'origine sur les coupures breakbeat, l'opcode peut être appliqué à n'importe quel type de source audio.

La séquence de coupure typique sur une mesure subdivisée en croches serait

3+ 3R + 2

dans laquelle nous prenons un bloc de trois unités au début de la source, le répétons, puis deux unités venant des 7èmes et 8èmes croches de la source.

Nous parlons de restituer des phrases (une séquence de coupures avant d'atteindre une nouvelle phrase au début d'une mesure) et des unités (comme subdivisions des notes).

L'opcode donne un rendu plus vivant lorsqu'on utilise simultanément plusieurs versions synchronisées.

## Syntaxe

```
a1,a2 bbcuts asource1, asource2, ibps, isubdiv, ibarlength, iphrasebars, \
inumrepeats [, istutterspeed] [, istutterchance] [, ienvchoice]
```

## Initialisation

*ibps* -- Tempo pour les coupures, en pulsations par seconde.

*isubdiv* -- Unité de subdivision pour une mesure. Par exemple 8 désigne la croche (dans une mesure à 4/4).

*ibarlength* -- Nombre de pulsations par mesure. Il vaut 4 pour la mesure par défaut à 4/4.

*iphrasebars* -- Les coupures sont générées par phrases, chaque phrase durant *iphrasebars*.

*inumrepeats* -- Dans une utilisation normale, l'algorithme permet une répétition supplémentaire d'une coupure donnée à la fois. Ce paramètre permet de modifier ce comportement. La valeur 1 représente la norme d'une répétition supplémentaire. 0 supprime la répétition et l'on obtient la source originale excepté pour l'enveloppe et le stuttering.

*istutterspeed* -- (facultatif, par défaut=1) Le stutter peut être un multiple entier de la vitesse de subdivision. Par exemple, si *isubdiv* vaut 8 (croches) et *istutterspeed* vaut 2, le stutter est en doubles croches (= subdiv de 16). La valeur par défaut est 1.

*istutterchance* -- (facultatif, par défaut=0) La fin d'une phrase a cette probabilité de devenir l'unité de répétition du stutter (0,0 à 1,0). La valeur par défaut est 0.

*ienvchoice* -- (facultatif, par défaut=1) Choisir 1 pour l'activer (enveloppe exponentielle pour les grains de coupure) ou 0 pour le désactiver. S'il est désactivé, on entendra des clics, mais ça peut donner de bons résultats bruiteux, en particulier avec les sources percussives. La valeur par défaut est 1, actif.

## Exécution

*asource* -- Le signal sonore à couper. Cette version fonctionne en temps réel sans connaissance des événements audio futurs.

## Exemples

Voici un exemple de l'opcode *bbcuts*. Il utilise le fichier *bbcuts.csd* [examples/bbcuts.csd].

### Exemple 88. Exemple de l'opcode *bbcuts*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime output leave only the line below:
; -o bbcuts.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1 ;Play an audio file
aleft, aright diskin2 "kickroll.wav", 1, 0
outs aleft, aright

endin

instr 2 ;Cut-up stereo audio file.

ibps = 16
isubdiv = 2
ibarlength = 2
iphrasebars = 1
inumrepeats = 8

aleft, aright diskin2 "kickroll.wav", 1, 0
aleft, aright bbcuts aleft, aright, ibps, isubdiv, ibarlength, iphrasebars, inumrepeats
outs aleft, aright

endin

</CsInstruments>
<CsScore>

i 1 0 2
i 2 3 2
e
```



```
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*bbcutm*

## Crédits

Auteur : Nick Collins  
Londres  
Août 2001

Nouveau dans la version 4.13

# beadsynt

beadsynt — Banc d'oscillateurs à bande améliorée.

## Description

Opcode du greffon beosc.

Banc d'oscillateurs à bande améliorée, portage de l'oscillateur de loris (basé sur BEAdsynt de Supercollider). Peut fonctionner avec des tableaux et des tables de fonction. La famille d'opcodes à bande améliorée (beosc, beadsynt) implémente une modélisation du son et une synthèse qui préservent l'élégance et la souplesse du modèle sinusoïdal tout en ajustant les sons avec des composantes bruiteuses (non sinusoïdales). L'analyse est faite hors-ligne par une méthode de McAulay-Quatieri (MQ) améliorée qui extrait l'information de largeur de bande en plus des paramètres sinusoïdaux pour chaque partiel. Pour produire les composantes bruiteuses, la synthèse est faite avec des oscillateurs sinusoïdaux modifiés pour permettre l'introduction d'une largeur de bande variable.

La synthèse peut être contrôlée via un ensemble d'indicateurs (voir *iflags*), permettant de choisir entre bruit uniforme et bruit gaussien pour les composantes bruiteuses, table d'onde interpolée (désactivée pour économiser le cpu), et interpolation de fréquence entre les cycles-k (désactivée pour économiser le cpu).



### Note

L'implémentation originale (loris) utilise du bruit gaussien (normal) pour les composantes non sinusoïdales, qui, lorsqu'il est implémenté naïvement comme dans loris, consomme beaucoup de ressources cpu. Le portage dans Supercollider utilise un simple bruit uniforme. Nous implémentons les deux, avec une implémentation très efficace du bruit gaussien (utilisant une table précalculée), ce qui le rend presque aussi efficace que le bruit uniforme.

## Syntaxe

```
aout beadsynt kFreqs[], kAmps[], kBws[] \
    [, inumosc, iflags, kfreq, kbw, ifn, iphs ]

aout beadsynt ifreqft, iampft, ibwft, inumosc \
    [, iflags, kfreq, kbw, ifn, iphs ]
```

## Initialisation

*ifreqft* -- Une table contenant les fréquences pour chaque partiel.

*iampft* -- Une table contenant les amplitudes pour chaque partiel.

*ibwft* -- Une table contenant les largeurs de bande pour chaque partiel.

*inumosc* -- Le nombre de partiels à resynthétiser. dans la version avec tableaux, on peut le laisser non initialisé.

*iflags* -- 0 : bruit uniforme noise, 1 : bruit gaussien, +2 : oscillateur à interpolation linéaire, +4 : interpolation des fréquences. Vaut 1 par défaut.

*ifn* -- Une table contenant une onde sinus (ou -1 pour utiliser la table interne). Vaut -1 par défaut.

*iphs* -- Phase initiale. -1 : aléatoire, 0..1 : phase, > 1 : numéro d'une table contenant les phases. Vaut -1 par défaut.

## Exécution

*aout* -- Le son généré.

*kFreqs[]* -- Un tableau contenant les fréquences de chaque partiel.

*kAmps[]* -- Un tableau contenant les amplitudes de chaque partiel.

*kBws[]* -- Un tableau contenant les largeurs de bande de chaque partiel.

*kfreq* -- Pondération des fréquences. Toutes les fréquences sont multipliées par ce facteur (1 par défaut).

*kbw* -- Pondération de la largeur de bande. Toutes les largeurs de bande sont multipliées par ce facteur (1 par défaut).



### Note

*kFreqs[]*, *kAmps[]* et *kBws[]* doivent avoir la même taille (ceci vaut également pour *ifreqft*, *iampft* et *ibwft*).

## Exemples

Voici un exemple de l'opcode *beadsynt*. Il utilise le fichier *beadsynt.csd* [examples/beadsynt.csd].

### Exemple 89. Exemple de l'opcode *beadsynt*.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

/*

This is the example file for beadsynt

beadsynt
=====

Band-enhanced additive synthesis.
A port of Loris' band-enhanced resynthesis algorithms
(basen on Supercollider's BEOsc)

The band-enhanced family of opcodes (beosc, beadsynt) implement
sound modeling and synthesis that preserves the elegance and
malleability of a sinusoidal model, while accommodating sounds
with noisy (non-sinusoidal) components. Analysis is done offline,
with an enhanced McAulay-Quatieri (MQ) style analysis that extracts
bandwidth information in addition to the sinusoidal parameters for
each partial. To produce noisy components, we synthesize with sine
wave oscillators that have been modified to allow the introduction
of variable bandwidth.

Syntax
=====

beadsynt exists in two forms, one using arrays, the other using f-tables
```

```
aout beadsynt kFreqs[], kAmps[], kBws[], inumosc=-1, iflags=1, kfreq=1, kbw=1, ifn=-1, iphs=-1
aout beadsynt ifreqft, iampft, ibwft, inumosc, iflags=1, kfreq=1, kbw=1, ifn=-1, iphs=-1
```

```
kFreqs[]: an array holding the frequencies of each partial
kAmps[]: an array holding the amplitudes of each partial
kBws[]: an array holding the bandwidths of each partial
ifreqft: a table holding the frequencies of each partial
iampft: a table holding the amplitudes of each partial
ibwft: a table holding the bandwidths of each partial
inumosc: the number of partials to resynthesize (-1 to synthesize all)
iflags: 0: uniform noise
        1: gaussian noise
        +2: use linear interpolation for the oscil (similar to oscili)
        +4: freq interpolation
kfreq: freq. scaling factor
kbw: bandwidth scaling factor
ifn: a table holding a sine wave (or -1 to use builtin sine)
iphs: initial phase of the oscillators.
      -1: randomize phase (default)
      0-1: initial phase
      >=1: table holding the phase for each oscillator (size>=inumosc)
```

```
NB: kFreqs, kAmps and kBws must all be the same size (this also holds true for
    ifreqft, iampfr and ibwft)
```

```
This example uses the analysis file fox.mtx.wav which was produced with
loristrck_pack, see https://github.com/gesellkammer/loristrck
```

```
*/

sr = 44100
ksmps = 128
nchnls = 2
0dbfs = 1.0

gispectrum ftgen 0, 0, 0, -1, "fox.mtx.wav", 0, 0, 0

instr 1
  ifn = gispectrum
  iskip tab_i 0, ifn
  idt tab_i 1, ifn
  inumcols tab_i 2, ifn
  inumrows tab_i 3, ifn
  itimestart tab_i 4, ifn
  inumpartials = inumcols / 3
  imaxrow = inumrows - 2
  it = ksmps / sr
  igain init 1
  ispeed init 1
  idur = imaxrow * idt / ispeed
  kGains[] init inumpartials
  kfilter init 0
  ifreqscale init 1

  kt timeinsts
  kplayhead = phasor:k(ispeed/idur)*idur
  krow = kplayhead / idt
  kF[] getrowlin krow, ifn, inumcols, iskip, 0, 0, 3
  kA[] getrowlin krow, ifn, inumcols, iskip, 1, 0, 3
  kB[] getrowlin krow, ifn, inumcols, iskip, 2, 0, 3

  if(kt > idur*0.5) then
    kfilter = 1
  endif

  if (kfilter == 1) then
    kGains bpf kF, 300, 0.001, 400, 1, 1000, 1, 1100, 0.001
```

```

    kA *= kGains
endif

iflags = 0      ; uniform noise, no interpolation
aout beadsynt kF, kA, kB, -1, iflags, ifreqscale

if(kt > idur) then
    event "e", 0, 0, 0
endif
aenv cosseg 0, 0.02, igain, idur-0.02-0.1, igain, 0.1, 0
aout *= aenv
outs aout, aout
endin

schedule 1, 0, -1

</CsInstruments>
<CsScore>
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*beosc, adsynt2,*

## Crédits

Auteur : Eduardo Moguillansky 2018

Nouveau greffon dans la version 6.12

# beosc

beosc — Oscillateur à bande améliorée.

## Description

Opcode du greffon beosc.

Oscillateur à bande améliorée, portage de l'oscillateur de loris (basé sur BEOsc de Supercollider). La famille d'opcodes à bande améliorée (beosc, beadsynt) implémente une modélisation du son et une synthèse qui préservent l'élégance et la souplesse du modèle sinusoïdal tout en ajustant les sons avec des composantes bruiteuses (non sinusoïdales). L'analyse est faite hors-ligne par une méthode de McAulay-Quatieri (MQ) améliorée qui extrait l'information de largeur de bande en plus des paramètres sinusoïdaux pour chaque partiel. Pour produire les composantes bruiteuses, la synthèse est faite avec des oscillateurs sinusoïdaux modifiés pour permettre l'introduction d'une largeur de bande variable.

## Syntaxe

```
aout beosc xfreq, kbw [ , ifn, iphs, inoisetype ]
```

## Initialisation

*ifn* -- Une table contenant une onde sinus (-1 par défaut).

*iphs* -- La phase du sinus (0 par défaut).

*inoisetype* -- Le type du bruit. 0 = uniforme, 1 = gaussien (1 par défaut).



### Note

L'implémentation originale dans loris utilise un bruit gaussien ; le portage dans Supercollider utilise un bruit uniforme. Nous implémentons les deux. Dans notre implémentation le bruit gaussien est presque aussi efficace que le bruit uniforme, et donc le bruit normal a été choisi par défaut comme dans loris.

## Exécution

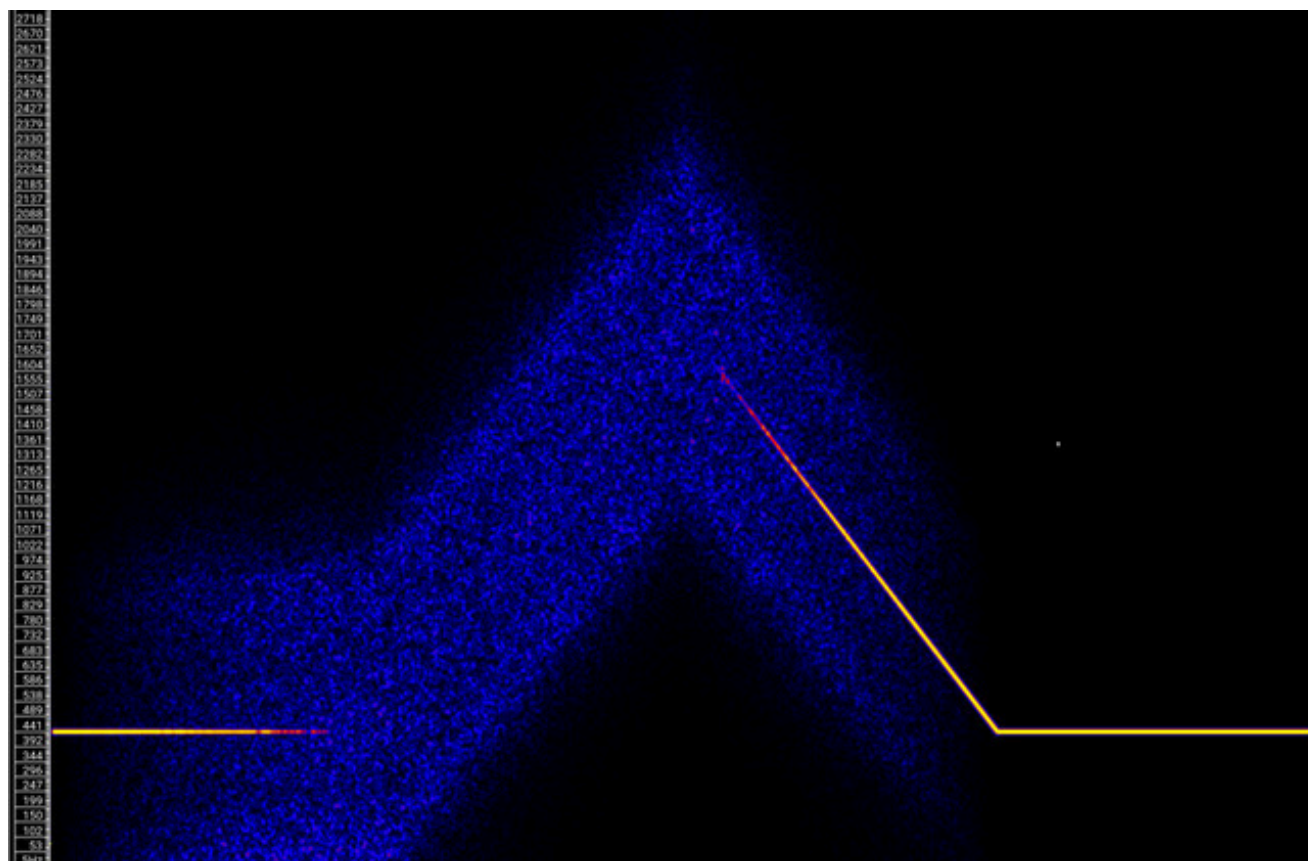
*aout* -- Le son généré.

*xfreq* -- la fréquence de l'oscillateur (taux-k ou -a).

Il n'y a pas de contrôle de l'amplitude. L'utilisateur doit pondérer la sortie *aout* par tout facteur nécessaire.

## Exemples

Voici un exemple de l'opcode beosc. Il utilise le fichier *beosc.csd* [examples/beosc.csd].

**Figure 1.****Exemple 90. Exemple de l'opcode beosc.**

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>
<CsOptions>
-odac      ;;;realtime audio out
</CsOptions>
<CsInstruments>

/*

This is the example file for beosc

beosc
=====

Band-Enhanced oscillator, a port of Loris's oscillator
(based on Supercollider's BEOsc)

The band-enhanced family of opcodes (beosc, beadsynt) implement
sound modeling and synthesis that preserves the elegance and
malleability of a sinusoidal model, while accommodating sounds
with noisy (non-sinusoidal) components. Analysis is done offline,
with an enhanced McAulay-Quatieri (MQ) style analysis that extracts
bandwidth information in addition to the sinusoidal parameters for
each partial. To produce noisy components, we synthesize with sine
```

*wave oscillators that have been modified to allow the introduction of variable bandwidth.*

```
aout beosc xfreq, kbw, ifn=-1, iphs=0, inoisetype=0
```

```
aout: the generated sound
afreq / kfreq: the frequency of the oscillator
kbw: the bandwidth of the oscillator. 0=pure sinusoidal
ifn: a table holding a sine wave (use -1 for the builtin wave used for oscili)
iphs: the phase of the sine (use unirand:i(6.28) to produce a random phase)
inoisetype: in the original implementation, gaussian (normal) noise is used,
            in supercollider's port, a simple uniform noise is used.
            We implement both. 0=uniform, 1=normal
```

*There is no control for amplitude. The user is supposed to just multiply the output aout by any factor needed.*

*NB: watch the output of this example with a freq. scope*  
*\*/*

```
sr = 44100
ksmps = 64
nchnls = 1
0dbfs = 1

instr 1
  idur1 = 8
  ifreq = 440
  kfreq linseg ifreq, idur1, ifreq, idur1, ifreq*4, idur1, ifreq
  kbw cosseg 0, idur1, 1, idur1, 1, idur1, 0
  ; freq bw fn phs noisetype(0=uniform)
  aout beosc kfreq, kbw, -1, unirand:i(6.28), 0
  aenv linsegr 0, 0.1, 1, 0.1, 1, 0.1, 0
  aout *= (aenv * 0.2)
  outch 1, aout
endin

</CsInstruments>
<CsScore>
i 1 0 32
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*beadsynt*

## Crédits

Auteur : Eduardo Moguillansky 2018

Nouveau greffon dans la version 6.12



# betarand

betarand — Générateur de nombres aléatoires de distribution beta (valeurs positives seulement).

## Description

Générateur de nombres aléatoires de distribution beta (valeurs positives seulement). C'est un générateur de bruit de classe x.

## Syntaxe

ares **betarand** krange, kalpha, kbeta

ires **betarand** krange, kalpha, kbeta

kres **betarand** krange, kalpha, kbeta

## Exécution

*krange* -- l'intervalle des nombres aléatoires (0 - *krange*).

*kalpha* -- valeur de alpha. Si *kalpha* est inférieur à un, ses petites valeurs favorisent les valeurs proches de 0.

*kbeta* -- valeur de beta. Si *kbeta* est inférieur à un, ses petites valeurs favorisent les valeurs proches de *krange*.

Si *kalpha* et *kbeta* sont tous deux égaux à un, nous obtenons une distribution uniforme. Si *kalpha* et *kbeta* sont tous deux supérieurs à un nous obtenons une sorte de distribution gaussienne. Ne produit que des nombres positifs.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode betarand. Il utilise le fichier *betarand.csd* [examples/betarand.csd].

### Exemple 91. Exemple de l'opcode betarand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o betarand.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
odbfs = 1

instr 1 ; every run time same values

kbeta betarand 100, 1, 1
printk .2, kbeta ; look
aout oscili 0.8, 440+kbeta, 1 ; & listen
outs aout, aout
endin

instr 2 ; every run time different values

seed 0
kbeta betarand 100, 1, 1
printk .2, kbeta ; look
aout oscili 0.8, 440+kbeta, 1 ; & listen
outs aout, aout
endin

</CsInstruments>
<CsScore>
; sine wave
f 1 0 16384 10 1

i 1 0 2
i 2 3 2
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra une ligne comme celle-ci :

```

i 1 time 0.00267: 85.74227
i 1 time 0.20267: 12.07606
i 1 time 0.40267: 25.03239
i 1 time 0.60267: 0.42037
i 1 time 0.80267: 76.69589
i 1 time 1.00000: 29.73339
i 1 time 1.20267: 48.29811
i 1 time 1.40267: 75.46507
i 1 time 1.60267: 74.80686
i 1 time 1.80000: 81.37473
i 1 time 2.00000: 55.48827
Seeding from current time 3472120656
i 2 time 3.00267: 57.21408
i 2 time 3.20267: 30.95705
i 2 time 3.40267: 19.71687
i 2 time 3.60000: 64.48965
i 2 time 3.80267: 72.35818
i 2 time 4.00000: 49.65395
i 2 time 4.20000: 55.25888
i 2 time 4.40267: 3.98308
i 2 time 4.60267: 52.98075
i 2 time 4.80267: 58.07925
i 2 time 5.00000: 56.38914

```

## Voir aussi

*seed, bexprnd, cauchy, exprand, gauss, linrand, pcauchy, poisson, trirand, unirand, weibull*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Existait dans la 3.30

# bexprnd

bexprnd — Générateur de nombres aléatoires de distribution exponentielle.

## Description

Générateur de nombres aléatoires de distribution exponentielle. C'est un générateur de bruit de classe x.

## Syntaxe

```
ares bexprnd krange  
ires bexprnd krange  
kres bexprnd krange
```

## Exécution

*krange* -- l'intervalle des nombres aléatoires (*-krange* à *+krange*)

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode bexprnd. Il utilise le fichier *bexprnd.csd* [examples/bexprnd.csd].

### Exemple 92. Exemple de l'opcode bexprnd.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac      ;;RT audio out  
;-iadc      ;;uncomment -iadc if RT audio input is needed too  
; For Non-realtime ouput leave only the line below:  
; -o bexprnd.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 32  
nchnls = 2  
0dbfs = 1  
  
instr 1    ; every run time same values  
  
kexp bexprnd 100  
  printk .2, kexp    ; look
```

```

aout oscili 0.8, 440+kexp, 1 ; & listen
outs aout, aout

endin

instr 2 ; every run time different values

seed 0
kexp bexprnd 100
printk .2, kexp ; look
aout oscili 0.8, 440+kexp, 1 ; & listen
outs aout, aout
endin

</CsInstruments>
<CsScore>
; sine wave
f 1 0 16384 10 1

i 1 0 2
i 2 3 2
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra une ligne comme celle-ci :

```

i 1 time 0.00267: 99.27598
i 1 time 0.20267: 74.97176
i 1 time 0.40267: -35.67213
i 1 time 0.60267: 1.10579
i 1 time 0.80267: -18.08816
i 1 time 1.00000: 28.93329
i 1 time 1.20267: 320.63733
i 1 time 1.40267: -332.05614
i 1 time 1.60267: -212.66361
i 1 time 1.80000: -92.57433
i 1 time 2.00000: 140.70939
Seeding from current time 4055201702
i 2 time 3.00267: 190.30495
i 2 time 3.20267: -58.30677
i 2 time 3.40267: 192.39784
i 2 time 3.60000: 12.72448
i 2 time 3.80267: 79.91503
i 2 time 4.00000: 34.44258
i 2 time 4.20000: 167.92680
i 2 time 4.40267: -117.10278
i 2 time 4.60267: -70.99155
i 2 time 4.80267: -23.24037
i 2 time 5.00000: -226.35500

```

## Voir aussi

*seed, betarand, cauchy, exprand, gauss, linrand, pcauchy, poisson, trirand, unirand, weibull*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

# bformenc1

bformenc1 — Encode un signal dans le format ambisonic B.

## Description

Encode un signal dans le format ambisonic B.

## Syntaxe

```
aw, ax, ay, az bformenc1 asig, kalpha, kbeta

aw, ax, ay, az, ar, as, at, au, av bformenc1 asig, kalpha, kbeta

aw, ax, ay, az, ar, as, at, au, av, ak, al, am, an, ao, ap, aq bformenc1 \
    asig, kalpha, kbeta

aarray[] bformenc1 asig, kalpha, kbeta
```

## Exécution

*aw, ax, ay, ...* -- cellules de sortie au format B.

*aarray* -- tableau de sortie pour recevoir les cellules au format B.

*asig* -- signal d'entrée.

*kalpha* -- angle d'azimut en degrés (dans le sens contraire des aiguilles d'une montre).

*kbeta* -- angle d'altitude en degrés.

## Exemples

Voici un exemple de l'opcode bformenc1. Il utilise le fichier *bformenc1.csd* [exemples/bformenc1.csd].

### Exemple 93. Exemple de l'opcode bformenc1.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
;-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o bformenc.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 32
nchnls = 8
0dbfs = 1

instr 1 ;without arrays
; generate pink noise
```

```
anoise pinkish 1

; two full turns
kalpha line 0, p3, 720
kbeta = 0

; generate B format
aw, ax, ay, az, ar, as, at, au, av bformenc1 anoise, kalpha, kbeta

; decode B format for 8 channel circle loudspeaker setup
a1, a2, a3, a4, a5, a6, a7, a8 bformdec1 4, aw, ax, ay, az, ar, as, at, au, av

; write audio out
outo a1, a2, a3, a4, a5, a6, a7, a8
endin

instr 2 ;with arrays (csound6)
;set file names for:
S_bform = "bform_out.wav" ;b-format (2nd order) output
S_sound = "sound_out.wav" ;sound output

; generate pink noise
anoise pinkish 1

; two full turns
kalpha line 0, p3, 720
kbeta = 0

;create array for B-format 2nd order (9 chnls)
aBform[] init 9

; generate B-format
aBform bformenc1 anoise, kalpha, kbeta

;write out b-format
fout "fout.wav", 18, aBform

;create array for audio output (8 channels)
aAudio[] init 8

;decode B format for 8 channel circle loudspeaker setup
aAudio bformdec1 4, aBform

; write audio out
fout S_sound, 18, aAudio
endin

</CsInstruments>
<CsScore>
i 1 0 8
i 2 8 8
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*bformdec1*

## Crédits

Auteurs : Richard Furse, Bruce Wiggins et Fons Adriaensen, d'après le code de Samuel Groner  
2008

Nouveau dans la version 5.09

Sortie sur tableau ajoutée dans la version 6.01



# bformdec1

bformdec1 — Décode un signal au format ambisonic B.

## Description

Décode un signal au format ambisonic B en signaux de haut-parleur spécifiques.

## Syntaxe

```
ao1, ao2 bformdec1 isetup, aw, ax, ay, az [, ar, as, at, au, av \
    [, abk, al, am, an, ao, ap, aq]]

ao1, ao2, ao3, ao4 bformdec1 isetup, aw, ax, ay, az [, ar, as, at, \
    au, av [, abk, al, am, an, ao, ap, aq]]

ao1, ao2, ao3, ao4, ao5 bformdec1 isetup, aw, ax, ay, az [, ar, as, \
    at, au, av [, abk, al, am, an, ao, ap, aq]]

ao1, ao2, ao3, ao4, ao5, ao6, ao7, ao8 bformdec1 isetup, aw, ax, ay, az \
    [, ar, as, at, au, av [, abk, al, am, an, ao, ap,
    aq]]]

aout[] bformdec1 isetup, abform[]
```

## Initialisation

Noter que les angles horizontaux sont mesurés dans le sens contraire des aiguilles d'une montre dans cette description.

*isetup* -- réglage de haut-parleur. Cinq réglages sont supportés :

- 1. Stéréo - L(90), R(-90) ; c'est un décodage stéréo dans le style M+S.
- 2. Quadraphonique - FL(45), BL(135), BR(-135), FR(-45). C'est un décodage "en phase" du premier ordre.
- 3. 5.0 - L(30), R(-30), C(0), BL(110), BR(-110). Noter que beaucoup de gens n'utilisent pas les angles ci-dessus pour leur grille de haut-parleurs et on peut réaliser un bon décodage pour DVD, etc en utilisant la configuration quadraphonique pour alimenter L, R, BL et BR (laissant C silencieux).
- 4. Octogone - FFL(22.5), FLL(67.5), BLL(112.5), BBL(157.5), BBR(-157.5), BRR(-112.5), FRR(-67.5), FFR(-22.5). C'est un décodage "en phase" de premier, deuxième ou troisième ordre en fonction du nombre de canaux en entrée.
- 5. Cube - FLD(45, -35.26), FLU(45, 35.26), BLD(135, -35.26), BLU(135, 35.26), BRD(-135, -35.26), BRU(-135, 35.26), FRD(-45, -35.26), FRU(-45, 35.26). C'est un décodage "en phase" du premier ordre.

## Exécution

*aw, ax, ay, ...* -- signal d'entrée au format B.

*ao1 .. ao8* -- signaux de haut-parleur spécifiques en sortie.

## Exemples

Voici un exemple de l'opcode `bformdec1`. Il utilise le fichier `bformenc1.csd` [examples/bformenc1.csd].

### Exemple 94. Exemple de l'opcode `bformdec1`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
;-odac        -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o bformenc.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 32
nchnls = 8
0dbfs = 1

instr 1 ;without arrays
; generate pink noise
anoise pinkish 1

; two full turns
kalpha line 0, p3, 720
kbeta = 0

; generate B format
aw, ax, ay, az, ar, as, at, au, av bformenc1 anoise, kalpha, kbeta

; decode B format for 8 channel circle loudspeaker setup
a1, a2, a3, a4, a5, a6, a7, a8 bformdec1 4, aw, ax, ay, az, ar, as, at, au, av

; write audio out
outo a1, a2, a3, a4, a5, a6, a7, a8
endin

instr 2 ;with arrays (csound6)
;set file names for:
S_bform = "bform_out.wav" ;b-format (2nd order) output
S_sound = "sound_out.wav" ;sound output

; generate pink noise
anoise pinkish 1

; two full turns
kalpha line 0, p3, 720
kbeta = 0

;create array for B-format 2nd order (9 chnls)
aBform[] init 9

; generate B-format
aBform bformenc1 anoise, kalpha, kbeta

;write out b-format
fout "fout.wav", 18, aBform

;create array for audio output (8 channels)
aAudio[] init 8
```

```
;decode B format for 8 channel circle loudspeaker setup
aAudio bformdecl 4, aBform

; write audio out
fout S_sound, 18, aAudio
endin

</CsInstruments>
<CsScore>
i 1 0 8
i 2 8 8
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*bformenc1*

## Crédits

Auteurs : Richard Furse, Bruce Wiggins et Fons Adriaensen, d'après le code de Samuel Groner 2008

Nouveau dans la version 5.09

# binit

binit — Conversion de bandes PVS en amplitude+fréquence.

## Description

L'opcode *binit* reçoit une entrée contenant un flot de signal TRACKS de vocodeur de phase (généré par exemple par *partials* et le convertit en trames de bins à largeur de bande régulière contenant des paires amplitude/fréquence (PVS\_AMP\_FREQ), adaptées à la resynthèse par recouvrement-addition (telle que celle réalisée par *pvsynth*) ou bien à des transformations de flot de signal de vocodeur de phase PVS. Pour chaque bin de fréquence, il cherche une bande de signal adaptée pour la remplir ; s'il n'en trouve pas, le bin sera vide (amplitude 0). Si plus d'une bande correspond à un bin, celle qui a la plus grande amplitude sera choisie. Cela signifie que l'intégralité du signal n'est pas traitée, c'est une opération avec pertes. Cependant, dans bien des situations, la perte n'est pas perceptible.

## Syntaxe

```
fsig binit fin, isize
```

## Exécution

*fsig* -- flot pv en sortie au format PVS\_AMP\_FREQ.

*fin* -- flot pv en entrée au format TRACKS.

*isize* -- taille de la TFR de la sortie (N).

## Exemples

Voici un exemple de l'opcode binit. Il utilise le fichier *binit.csd* [exemples/binit.csd].

### Exemple 95. Exemple de l'opcode binit.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o binit.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1
;ain inch 1      ; for live input
ain diskin "beats.wav", 1 ; input signal
fsl,fsi2 pvsifd ain, 2048, 512, 1 ; ifd analysis
```

```
fst partials fs1, fsi2, .003, 1, 3, 500 ; partial tracking
fbins binit fst, 2048 ; convert it back to bins
aout pvsynth fbins ; overlap-add resynthesis
outs aout, aout

endin

</CsInstruments>
<CsScore>

i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

L'exemple ci-dessus montre la recherche de partiels d'un signal d'analyse de distribution de fréquence instantanée, leur conversion en trames de bins et la resynthèse par recouvrement-addition.

## Crédits

Auteur : Victor Lazzarini  
Février 2006

Nouveau dans Csound5.01

# biquad

biquad — Un filtre numérique biquadratique glissant à usage général.

## Description

Un filtre numérique biquadratique glissant à usage général.

## Syntaxe

```
ares biquad asig, kb0, kb1, kb2, ka0, ka1, ka2 [, iskip]
```

## Initialisation

*iskip* (facultatif, 0 par défaut) -- s'il est non nul, l'initialisation est ignorée. Vaut 0 par défaut. (Nouveau dans la version 3.50 de Csound.)

## Exécution

*asig* -- signal d'entrée

*biquad* est un filtre numérique biquadratique à usage général de la forme :

$$a0*y(n) + a1*y[n-1] + a2*y[n-2] = b0*x[n] + b1*x[n-1] + b2*x[n-2]$$

Ce filtre a pour réponse en fréquence :

$$H(Z) = \frac{B(Z)}{A(Z)} = \frac{b0 + b1*Z^{-1} + b2*Z^{-2}}{a0 + a1*Z^{-1} + a2*Z^{-2}}$$

On rencontre souvent ce type de filtre dans la littérature sur le traitement numérique du signal. Il accepte six coefficients de taux-k définis par l'utilisateur.

## Exemples

Voici un exemple de l'opcode biquad. Il utilise le fichier *biquad.csd* [examples/biquad.csd].

### Exemple 96. Exemple de l'opcode biquad.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o biquad.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Instrument #1.
instr 1
; Get the values from the score.
idur = p3
iamp = p4
icps = cpspch(p5)
kfco = p6
krez = p7

; Calculate the biquadratic filter's coefficients
kfcon = 2*3.14159265*kfco/sr
kalpha = 1-2*krez*cos(kfcon)*cos(kfcon)+krez*krez*cos(2*kfcon)
kbeta = krez*krez*sin(2*kfcon)-2*krez*cos(kfcon)*sin(kfcon)
kgama = 1+cos(kfcon)
km1 = kalpha*kgama+kbeta*sin(kfcon)
km2 = kalpha*kgama-kbeta*sin(kfcon)
kden = sqrt(km1*km1+km2*km2)
kb0 = 1.5*(kalpha*kalpha+kbeta*kbeta)/kden
kb1 = kb0
kb2 = 0
ka0 = 1
ka1 = -2*krez*cos(kfcon)
ka2 = krez*krez

; Generate an input signal.
axn vco 1, icps, 1

; Filter the input signal.
ayn biquad axn, kb0, kb1, kb2, ka0, ka1, ka2
outs ayn*iamp/2, ayn*iamp/2
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Sta Dur Amp Pitch Fco Rez
i 1 0.0 1.0 20000 6.00 1000 .8
i 1 1.0 1.0 20000 6.03 2000 .95
e

</CsScore>
</CsoundSynthesizer>

```

Voici un autre exemple de l'opcode biquad utilisé pour de la synthèse modale. Il utilise le fichier *biquad-2.csd* [exemples/biquad-2.csd]. Voir l'annexe *Rapports de Fréquence Modale* pour d'autres rapports de fréquence.

### Exemple 97. Exemple de l'opcode biquad pour de la synthèse modale.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out Audio in
-odac          -iadc      ;;RT audio I/O

```

```

; For Non-realtime ouput leave only the line below:
; -o biquad-2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

/* modal synthesis using biquad filters as oscillators
   Example by Scott Lindroth 2007 */

instr 1

    ipi = 3.1415926
    idenom = sr*0.5

    ipulseSpd = p4
    icps      = p5
    ipan      = p6
    iamp      = p7
    iModes    = p8

    apulse    mpulse iamp, 0

    icps      = cpspch( icps )

; filter gain

    iamp1 = 600
    iamp2 = 1000
    iamp3 = 1000
    iamp4 = 1000
    iamp5 = 1000
    iamp6 = 1000

; resonance

    irpole1 = 0.99999
    irpole2 = irpole1
    irpole3 = irpole1
    irpole4 = irpole1
    irpole5 = irpole1
    irpole6 = irpole1

; modal frequencies

    if (iModes == 1) goto modes1
    if (iModes == 2) goto modes2

modes1:
    if1    = icps * 1           ;pot lid
    if2    = icps * 6.27
    if3    = icps * 3.2
    if4    = icps * 9.92
    if5    = icps * 14.15
    if6    = icps * 6.23
    goto nextPart

modes2:
    if1    = icps * 1           ;uniform wood bar
    if2    = icps * 2.572
    if3    = icps * 4.644
    if4    = icps * 6.984
    if5    = icps * 9.723
    if6    = icps * 12.0

```



```

goto nextPart

nextPart:

; convert frequency to radian frequency

itheta1 = (if1/idenom) * ipi
itheta2 = (if2/idenom) * ipi
itheta3 = (if3/idenom) * ipi
itheta4 = (if4/idenom) * ipi
itheta5 = (if5/idenom) * ipi
itheta6 = (if6/idenom) * ipi

; calculate coefficients

ib11 = -2 * irpole1 * cos(itheta1)
ib21 = irpole1 * irpole1
ib12 = -2 * irpole2 * cos(itheta2)
ib22 = irpole2 * irpole2
ib13 = -2 * irpole3 * cos(itheta3)
ib23 = irpole3 * irpole3
ib14 = -2 * irpole4 * cos(itheta4)
ib24 = irpole4 * irpole4
ib15 = -2 * irpole5 * cos(itheta5)
ib25 = irpole5 * irpole5
ib16 = -2 * irpole6 * cos(itheta6)
ib26 = irpole6 * irpole6

;printk 1, ib 11
;printk 1, ib 21

; also try setting the -1 coeff. to 0, but be sure to scale down the amplitude!

asin1    biquad  apulse * iamp1, 1, 0, -1, 1, ib11, ib21
asin2    biquad  apulse * iamp2, 1, 0, -1, 1, ib12, ib22
asin3    biquad  apulse * iamp3, 1, 0, -1, 1, ib13, ib23
asin4    biquad  apulse * iamp4, 1, 0, -1, 1, ib14, ib24
asin5    biquad  apulse * iamp5, 1, 0, -1, 1, ib15, ib25
asin6    biquad  apulse * iamp6, 1, 0, -1, 1, ib16, ib26

afin      =      (asin1 + asin2 + asin3 + asin4 + asin5 + asin6)

outs      afin * sqrt(p6), afin*sqrt(1-p6)

endin
</CsInstruments>
<CsScore>
;ins      st      dur  pulseSpd  pch      pan      amp      Modes
i1        0      12    0          7.089    0        0.7      2
i1        .      .    .          7.09     1        .        .
i1        .      .    .          7.091    0.5      .        .

i1        0      12    0          8.039    0        0.7      2
i1        0      12    0          8.04     1        0.7      2
i1        0      12    0          8.041    0.5      0.7      2

i1        9      .    .          7.089    0        .        2
i1        .      .    .          7.09     1        .        .
i1        .      .    .          7.091    0.5      .        .

i1        9      12    0          8.019    0        0.7      2
i1        9      12    0          8.02     1        0.7      2
i1        9      12    0          8.021    0.5      0.7      2
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*biquada, moogvcf, rezzzy*

## Crédits

Auteur : Hans Mikelson  
Octobre 1998

Nouveau dans la version 3.49 de Csound.

# biquada

biquada — Un filtre numérique biquadratique glissant à usage général avec des paramètres de taux-a.

## Description

Un filtre numérique biquadratique glissant à usage général.

## Syntaxe

ares **biquada** asig, ab0, ab1, ab2, aa0, aa1, aa2 [, iskip]

## Initialisation

*iskip* (facultatif, 0 par défaut) -- s'il est non nul, l'initialisation est ignorée. Vaut 0 par défaut. (Nouveau dans la version 3.50 de Csound.)

## Exécution

*asig* -- signal d'entrée

*biquada* est un filtre numérique biquadratique à usage général de la forme :

$$a0*y(n) + a1*y[n-1] + a2*y[n-2] = b0*x[n] + b1*x[n-1] + b2*x[n-2]$$

Ce filtre a pour réponse en fréquence :

$$H(Z) = \frac{B(Z)}{A(Z)} = \frac{b0 + b1*Z^{-1} + b2*Z^{-2}}{a0 + a1*Z^{-1} + a2*Z^{-2}}$$

On rencontre souvent ce type de filtre dans la littérature sur le traitement numérique du signal. Il accepte six coefficients de taux-a définis par l'utilisateur.

## Exemples

Voici un exemple de l'opcode biquada. Il utilise le fichier *biquad.csd* [examples/biquada.csd].

### Exemple 98. Exemple de l'opcode biquada.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o biquad.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Instrument #1.
instr 1
; Get the values from the score.
idur = p3
iamp = p4
icps = cpspch(p5)
afco expon 100, p3, 2000
arez line 0.8, p3, 0.99

; Calculate the biquadratic filter's coefficients
afcon = 2*3.14159265*afco/sr
aalpha = 1-2*arez*cos(afcon)*cos(afcon)+arez*arez*cos(2*afcon)
abeta = arez*arez*sin(2*afcon)-2*arez*cos(afcon)*sin(afcon)
agama = 1+cos(afcon)
am1 = aalpha*agama+abeta*sin(afcon)
am2 = aalpha*agama-abeta*sin(afcon)
aden = sqrt(am1*am1+am2*am2)
ab0 = 1.5*(aalpha*aalpha+abeta*abeta)/aden
ab1 = ab0
ab2 = 0
aa0 = 1
aa1 = -2*arez*cos(afcon)
aa2 = arez*arez

; Generate an input signal.
axn vco 1, icps, 1

; Filter the input signal.
ayn biquada axn, ab0, ab1, ab2, aa0, aa1, aa2
outs ayn*iamp/2, ayn*iamp/2
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Sta Dur Amp Pitch
i 1 0.0 5.0 20000 6.00
e

</CsScore>
</CsSoundSynthesizer>

```

## Voir aussi

*biquad*

## Crédits

Auteur : Hans Mikelson

Auteur : John ffitch d'après Hans Mikelson

Août 2001

Nouveau dans la version 4.13 de Csound.

# birnd

birnd — Retourne un nombre aléatoire dans un intervalle bipolaire.

## Description

Retourne un nombre aléatoire dans un intervalle bipolaire.

## Syntaxe

**birnd**(x) (taux-i ou -k seulement)

Où l'argument entre parenthèses peut être une expression. Ces convertisseurs de valeur échantillonnent une séquence aléatoire globale, mais sans référencer une *racine*. Le résultat peut devenir un terme d'une expression ultérieure.

## Exécution

Retourne un nombre aléatoire dans l'intervalle bipolaire allant de  $-x$  à  $x$ . *rnd* et *birnd* obtiennent leurs valeurs d'un générateur de nombres pseudo-aléatoires global, puis les mettent à l'échelle de l'intervalle demandé. Le générateur global unique distribuera ainsi sa séquence à ces unités durant toute l'exécution, quelque soit l'ordre d'arrivée de ces demandes.

## Exemples

Voici un exemple de l'opcode birnd. Il utilise le fichier *birnd.csd* [exemples/birnd.csd].

### Exemple 99. Exemple de l'opcode birnd.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

instr 1 ; Generate a random number from -1 to 1.

kbin = birnd(1)
printk .2, kbin

endin

</CsInstruments>
<CsScore>
```

```
i 1 0 1
i 1 + .
i 1 + .
i 1 + .
i 1 + .
i 1 + .
i 1 + .
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
instr 1: i1 = 0.94700
instr 1: i1 = -0.72119
instr 1: i1 = 0.53898
instr 1: i1 = 0.05001
instr 1: i1 = 0.24533
instr 1: i1 = 0.93902
instr 1: i1 = 0.43364
```

## Voir aussi

*rnd*

## Crédits

Auteur: Barry L. Vercoe  
MIT  
Cambridge, Massachussetts  
1997

Etendu dans la version 3.47 au taux-x par John ffitch.

# bpf

bpf — Fonction à points charnière avec interpolation linéaire.

## Description

Opcodes du greffon emugens.

Fonction à points charnière avec interpolation linéaire.

## Syntaxe

```
ky bpf kx, kx1, ky1, kx2, ..., kxn, kyn
iy bpf ix, ix1, iy1, ix2, ..., ixn, iyn
kys[] bpf kxs[], kx1, ky1, kx2, ..., kxn, kyn
iys[] bpf ixs[], ix1, iy1, ix2, ..., ixn, iyn
ky bpf kx, kxs[], kys[]
iy bpf ix, ixs[], iys[]
ay bpf ax, kx1, ky1, kx2, ..., kxn, kyn
ay bpf ax, kxs[], kys[]
ky, kw bpf kx, kxs[], kys[], kws[]
```

## Exécution

**kx** -- Valeur d'entrée.

**kxn, kyn** -- Définit un point charnière. Peut changer au taux-k, mais tous les *kxs* doivent être ordonnés.

Les points (kx1, ky1), (kx2, ky2), etc, définissent une fonction interpolée linéairement. Cette fonction est évaluée au point *kx*. Cette fonction s'étend à la fois vers -inf et +inf, si bien que si *kx* < kx1 alors *ky* = ky1 et pareillement à l'autre extrémité.

Les lignes suivantes sont équivalentes :

```
ky bpf kx, 0, 0, 0.5, 10, 1.02, 200

itab ftgenonce 0, 0, -27, 0, 0, 50, 10, 102, 200
ky = tablei(limit(kx, 0, 1.02)*100, itab)
```



### Note

Les valeurs *x* doivent être ordonnées. Les valeurs *x* et *y* peuvent changer mais les valeurs *x* doivent rester ordonnées.

## Exemples

Voici un exemple de l'opcode *bpf*. Il utilise le fichier *bpf.csd* [examples/bpf.csd].



## Exemple 100. Exemple de l'opcode bpf.

```

<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

; Example for opcode bpf

/*

    bpf stands for Break Point Function

    Given an x value and a series of pairs (x, y), it returns
    the corresponding y value in the linear curve defined by the
    pairs

    It works both at i- and k- time

    ky    bpf kx,    kx0, ky0, kx1, ky1, kx2, ky2, ...
    kys[] bpf kxs[], kx0, ky0, kx1, ky1, kx2, ky2, ...

    NB: x values must be ordered (kx0 < kx1 < kx2 etc)

    See also: bpfcos, linlin, lincos

*/

sr = 44100
ksmps = 64
nchnls = 2
0dbfs = 1

instr 1
    kx line -1, p3, 2.5
    ky bpf kx, \
        0, 0, \
        1.01, 10, \
        2, 0.5, \
        2.5, -1
    printks "kx: %f    ky: %f \n", 0.1, kx, ky
endin

instr 2
; test i-time
ix = 1.2
iy bpf ix, 0,0, 0.5,5, 1,10, 1.5,15, 2,20, 2.5,25, 3,30
print iy
turnoff
endin

instr 3
; bpf also works with arrays
kx[] fillarray 0, 0.15, 0.25, 0.35, 0.45, 0.55, 0.6
ky[] bpf kx, 0,0, 0.1,10, 0.2,20, 0.3,30, 0.4,40, 0.5,50
printarray ky, 1, "", "ky="
turnoff
endin

instr 4
; bpf as an envelope generator, like linsegb but driven by external phase
; bpf + rms can also be used as compressor

atime linseg 0, p3*0.62, p3, p3*0.38, 0

```

```

aenv = bpf(atime, 0,0, 0.1,1, 0.5, 0.2) ^ 2
kbw = bpf(timeinsts(), 0, 0, p3*0.62, 1) ^ 3
asig = (beosc(1000, kbw, -1, rnd(6.28)) + beosc(1012, kbw, -1, rnd(6.28))) * 0.3
kratio bpf dbamp(rms:k(asig)), -12, 1, -6, 0.4, -3, 1/100
asig *= aenv * interp(sc_lagud(kratio, 0.01, 0.1))
outs asig, asig
endin

</CsInstruments>
<CsScore>
; i 1 0 3
; i 2 0 -1
; i 3 0 -1
i 4 0 3

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*bpfcos, linlin xyscale, scale, ntrpol,*

## Crédits

Auteur : Eduardo Moguillansky 2017

# bpfcos

bpfcos — Fonction à points charnière avec interpolation par cosinus (transitions douces).

## Description

Opcode du greffon emugens.

Fonction à points charnière avec interpolation par cosinus. Etant données une valeur  $x$  et une série de paires  $(x, y)$ , elle retourne la valeur de  $y$  correspondante dans la courbe du demi-cosinus définie par les paires. Fonctionne au taux- $i$  ainsi qu'au taux- $k$ .

## Syntaxe

```
ky bpfcos kx, kx1, ky1, kx2, ..., kxn, kyn
kys[] bpfcos kxs[], kx1, ky1, kx2, ..., kxn, kyn
ky bpfcos kx, kxs[], kys[]
ky bpfcos kx, ix[], iys[]
ky, kz bpfcos kx, kxs[], kys[], kzs[]
ky, kz bpfcos kx, ix[], iys[], izs[]
```

## Exécution

**kx** -- Valeur en entrée. Peut également être un tableau.

**kxn, kyn** -- Définit un point charnière. Peut changer au taux- $k$ , mais tous les  $kxs$  doivent être ordonnés.

Les points  $(kx1, ky1)$ ,  $(kx2, ky2)$ , etc, définissent une fonction interpolée par cosinus. Cette fonction est évaluée au point  $kx$ . Cette fonction s'étend à la fois vers  $-\infty$  et  $+\infty$ , si bien que si  $kx < kx1$  alors  $ky = ky1$  et pareillement à l'autre extrémité.

## Variantes avec tableaux

```
kys[] bpfcos kxs[], kx1, ky1, kx2, ..., kxn, kyn
```

Pour chaque  $x$  dans  $kxs$ , la valeur  $y$  définie par les points  $(kx1, ky1)$ , ...  $(kxn, kyn)$  est calculée.

```
ky bpfcos kx, ix[], iys[]
```

Dans cette variante les points sont définis par des tableaux.  $ixs$  contient toutes les valeurs de  $x$ ,  $iys$  contient toutes les valeurs de  $y$ .

```
ky, kz bpfcos kx, kxs[], kys[], kzs[]
```

Semblable à la variante ci-dessus, mais pour des points multidimensionnels. Etant données deux courbes partageant les mêmes points  $x$ , calcule la valeur pour  $x$  dans les deux dimensions simultanément. (Voir exemple)



### Note

Les valeurs  $x$  doivent être triées. Les valeurs  $x$  et les valeurs  $y$  peuvent changer mais les valeurs  $x$  doivent rester triées.

## Exemples

Voici un exemple de l'opcode `bpfcos`. Il utilise le fichier `bpfcos.csd` [examples/bpfcos.csd].

### Exemple 101. Exemple de l'opcode `bpfcos`.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

; Example for opcode bpfcos

/*

    bpf stands for Break Point Function

    Given an x value and a series of pairs (x, y), it returns
    the corresponding y value in the half cosine curve defined by the
    pairs

    It works both at i- and k- time

    ky      bpfcos kx,      kx0, ky0, kx1, ky1, kx2, ky2, ...
    kys[] bpfcos kxs[], kx0, ky0, kx1, ky1, kx2, ky2, ...
    ky      bpfcos kx, kxs[], kys[]
    ky, kz bpfcos kx, kxs[], kys[], kzs[]

    NB: x values must be ordered (kx0 < kx1 < kx2 etc)

    See also: bpf, linlin, lincos

*/
sr = 44100
ksmps = 64
nchnls = 1
0dbfs = 1

instr 1
  kx line -1, p3, 2.5
  ky bpfcos kx, \
           0, 0, \
           1.01, 10, \
           2, 0.5, \
           2.5, -1
  printks "kx: %f   ky: %f \n", 0.1, kx, ky
endin

instr 2
; test i-time
ix = 1.2
iy bpfcos ix, 0,0, 0.5,5, 1,10, 1.5,15, 2,20, 2.5,25, 3,30
print iy
turnoff
endin

instr 3
; bpfcos also works with arrays. For each kx value in kxs,
; calculate the corresponding ky
kxs[] fillarray 0, 0.15, 0.25, 0.35, 0.45, 0.55, 0.6
kys[] bpfcos kxs, 0,0, 0.1,10, 0.2,20, 0.3,30, 0.4,40, 0.5,50
printarray kys, 1, " ", "kys="
turnoff
```

```

endin

instr 4
; bpfcos is useful to implement envelopes with ease-in/out shape
kpitch bpfcos timeinsts(), 0, 60, 2, 61, 3, 65, 3.5, 60
a0 oscili 0.5, mtof(kpitch)
kenv0 linseg 0, 0.5, 1, p3-1, 1, 0.5, 0
kenv bpfcos kenv0, 0, 0, 0.5, 0.25, 1, 0.7
a0 *= interp(kenv)
outch 1, a0
endin

instr 5
; arrays can also be used to define the points of a break-point-function
; multiple arrays can be used simultaneously
; In this case, we define a line and for each point in the line a
; corresponding pitch (midinote) and amplitude
; NB: kTimes uses absolute times
kTimes[] fillarray 0, 1, 1.5, 2, 3, 5
kPitches[] fillarray 60, 65, 64, 69, 60, 61
kAmps[] fillarray 0, 1, 0.1, 1, 0.1, 1
; play the envelopes at half speed
kpitch, kamp bpfcos timeinsts()*0.5, kTimes, kPitches, kAmps
aout oscili a(kamp), a(mtof:k(kpitch))
; declick
aout *= linsegr(0, 0.1, 1, 0.1, 0)
outch 1, aout
endin

</CsInstruments>
<CsScore>
; i 1 1 3
; i 2 0 -1
; i 3 0 -1
; i 4 0 5
i 5 0 10
e 12
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*bpf, lincos, cosseg linlin, scale, ntrpol, xyscale*

## Crédits

Auteur : Eduardo Moguillansky 2018

Nouveau greffon dans la version 6.12

Variantes avec tableaux dans la version 6.14

# bqrez

bqrez — Un filtre multi-modes du second ordre.

## Description

Un filtre multi-modes du second ordre.

## Syntaxe

```
ares bqrez asig, xfco, xres [, imode] [, iskip]
```

## Initialisation

*imode* (facultatif, 0 par défaut) -- Le mode du filtre. Un des choix suivants :

- 0 = passe-bas (par défaut)
- 1 = passe-haut
- 2 = passe-bande
- 3 = réjecteur de bande
- 4 = passe-tout

*iskip* (facultatif, 0 par défaut) -- s'il est non nul, l'initialisation du filtre est ignorée. (Nouveau dans les versions 4.23f13 et 5.0 de Csound).

## Exécution

*ares* -- signal audio en sortie.

*asig* -- signal audio n entrée.

*xfco* -- fréquence de coupure du filtre en Hz. Peut-être de taux-i, de taux-k ou de taux-a.

*xres* -- importance de la résonance. Des valeurs entre 1 et 100 sont typiques. La résonance doit être supérieure ou égale à 1. Une valeur de 100 donne un gain de 20 dB à la fréquence de coupure. Peut-être de taux-i, de taux-k ou de taux-a.

Tous les modes du filtre peuvent être modulés sur la fréquence ainsi que sur la résonance.

*bqrez* est un filtre passe-bas résonant créé au moyen des équations du domaine  $s$  de Laplace pour les filtres passe-bas, passe-haut et passe-bande normalisées à une fréquence. On a utilisé la transformation bilinéaire contenant une constante de transformation de fréquence du domaine  $s$  dans le domaine  $z$  pour faire concorder exactement les fréquences ensemble. De nombreuses identités trigonométriques ont été utilisées pour simplifier les calculs. Il est très stable sur tout l'intervalle de travail des fréquences jusqu'à la fréquence de Nyquist.

## Exemples

Voici un exemple de l'opcode *bqrez*. Il utilise le fichier *bqrez.csd* [examples/bqrez.csd].

## Exemple 102. Exemple de l'opcode **bqrez** emprunté de l'opcode « rezzy » dans le manuel de Kevin Conder.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o bqrez.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1    ;sawtooth waveform.

kfco line 200, p3, 2000;filter-cutoff frequency from .2 to 5 KHz.
kres = p4 ;resonance
imode = p5 ;mode
asig vco 0.2, 220, 1
afilt bqrez asig, kfco, kres, imode
asig balance afilt, asig
outs asig, asig

endin

</CsInstruments>
<CsScore>
;sine wave
f 1 0 16384 10 1

i 1 0 3 1 0 ; low pass
i 1 + 3 30 0 ; low pass
i 1 + 3 1 1 ; high pass
i 1 + 3 30 1 ; high pass

e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*biquad, moogvcf, rezzy*

## Crédits

Auteur : Matt Gerassimoff  
Nouveau dans la version 4.32  
Ecrit en novembre 2002.

# butbp

butbp — Identique à butterbp.

## Description

Identique à *butterbp*.

## Syntaxe

```
ares butbp asig, kfreq, kband [, iskip]
```



# butbr

butbr — Identique à butterbr.

## Description

Identique à *butterbr*.

## Syntaxe

```
ares butbr asig, kfreq, kband [, iskip]
```

# buthp

buthp — Identique à butterhp.

## Description

Identique à *butterhp*.

## Syntaxe

```
ares buthp asig, kfreq [, iskip]
```

```
ares buthp asig, afreq [, iskip]
```

# butlp

butlp — Identique à butterlp.

## Description

Identique *butterlp*.

## Syntaxe

```
ares butlp asig, kfreq [, iskip]
```

```
ares butlp asig, afreq [, iskip]
```

# butterbp

butterbp — Un filtre de Butterworth passe-bande.

## Description

Implémentation d'un filtre de Butterworth passe-bande du second ordre. Cet opcode peut aussi être écrit comme *butbp*.

## Syntaxe

```
ares butterbp asig, xfreq, xband [, iskip]
```

## Initialisation

*iskip* (facultatif, 0 par défaut) -- L'initialisation est ignorée s'il est présent et non nul.

## Exécution

Ces filtres sont des filtres RII de Butterworth du second ordre. Ils sont légèrement plus lents que les filtres originaux de Csound, mais ils offrent une bande passante presque plate et une précision et une atténuation de la bande bloquée très bonnes.

*asig* -- Signal d'entrée à filtrer.

*xfreq* -- Fréquence de coupure ou centrale pour chacun des filtres.

*xband* -- Largeur de la bande passante ou de la bande de réjection des filtres.

## Exemples

Voici un exemple de l'opcode butterbp. Il utilise le fichier *butterbp.csd* [examples/butterbp.csd].

### Exemple 103. Exemple de l'opcode butterbp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o butterbp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
```

```
instr 1 ; White noise signal

asig rand 0.6
outs asig, asig

endin

instr 2 ;filtered noise

asig rand 1
abp butterbp asig, 2000, 100 ;passing only 1950 to 2050 Hz
outs abp, abp

endin

</CsInstruments>
<CsScore>

i 1 0 2
i 2 2.5 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*butterbr, butterhp, butterlp*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Existait dans la version 3.30

Paramètres de taux audio introduits dans la version 6.02

Octobre 2013.

# butterbr

butterbr — Un filtre de Butterworth réjecteur de bande.

## Description

Implémentation d'un filtre de Butterworth réjecteur de bande du second ordre. Cet opcode peut aussi être écrit comme *butbr*.

## Syntaxe

```
ares butterbr asig, xfreq, xband [, iskip]
```

## Initialisation

*iskip* (facultatif, 0 par défaut) -- L'initialisation est ignorée s'il est présent et non nul.

## Exécution

Ces filtres sont des filtres RII de Butterworth du second ordre. Ils sont légèrement plus lents que les filtres originaux de Csound, mais ils offrent une bande passante presque plate et une précision et une atténuation de la bande bloquée très bonnes.

*asig* -- Signal d'entrée à filtrer.

*xfreq* -- Fréquence de coupure ou centrale pour chacun des filtres.

*xband* -- Largeur de la bande passante ou de la bande de réjection des filtres.

## Exemples

Voici un exemple de l'opcode butterbr. Il utilise le fichier *butterbr.csd* [examples/butterbr.csd].

### Exemple 104. Exemple de l'opcode butterbr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o butterbr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
```

```
instr 1 ; White noise

asig rand 0.5
outs asig, asig

endin

instr 2 ; filtered noise

asig rand 0.7
abr butterbr asig, 3000, 2000 ;center frequency = 3000, bandwidth = +/- (2000)/2, so 2000-4000
outs abr, abr

endin

</CsInstruments>
<CsScore>

i 1 0 2
i 2 2.5 2

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*butterbp, butterhp, butterlp*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Existait dans la version 3.30

Paramètres de taux audio introduits dans la version 6.02

Octobre 2013.

# butterhp

butterhp — Un filtre de Butterworth passe-haut.

## Description

Implémentation d'un filtre de Butterworth passe-haut du second ordre. Cet opcode peut aussi être écrit comme *buthp*.

## Syntaxe

```
ares butterhp asig, kfreq [, iskip]
ares butterhp asig, afreq [, iskip]
```

## Initialisation

*iskip* (facultatif, 0 par défaut) -- L'initialisation est ignorée s'il est présent et non nul.

## Exécution

Ces filtres sont des filtres RII de Butterworth du second ordre. Ils sont légèrement plus lents que les filtres originaux de Csound, mais ils offrent une bande passante presque plate et une précision et une atténuation de la bande bloquée très bonnes.

*asig* -- Signal d'entrée à filtrer.

*kfreq* -- Fréquence de coupure ou centrale pour chacun des filtres.

## Exemples

Voici un exemple de l'opcode butterhp. Il utilise le fichier *butterhp.csd* [examples/butterhp.csd].

### Exemple 105. Exemple de l'opcode butterhp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o butterhp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1    ; White noise
```



```
asig rand 0.5
outs asig, asig

endin

instr 2 ; filtered noise

asig rand 0.6
ahp butterhp asig, 500 ;pass frequencies above 500 Hz
outs ahp, ahp

endin

</CsInstruments>
<CsScore>

i 1 0 2
i 2 2.5 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*butterbp, butterbr, butterlp*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Existait dans la version 3.30

Paramètres de taux audio introduits dans la version 6.02

Octobre 2013.

# butterlp

butterlp — Un filtre de Butterworth passe-bas.

## Description

Implémentation d'un filtre de Butterworth passe-bas du second ordre. Cet opcode peut aussi être écrit comme *butlp*.

## Syntaxe

```
ares butterlp asig, kfreq [, iskip]
ares butterlp asig, afreq [, iskip]
```

## Initialisation

*iskip* (facultatif, 0 par défaut) -- L'initialisation est ignorée s'il est présent et non nul.

## Exécution

Ces filtres sont des filtres RII de Butterworth du second ordre. Ils sont légèrement plus lents que les filtres originaux de Csound, mais ils offrent une bande passante presque plate et une précision et une atténuation de la bande bloquée très bonnes.

*asig* -- Signal d'entrée à filtrer.

*kfreq/afreq* -- Fréquence de coupure ou centrale pour chacun des filtres.

## Exemples

Voici un exemple de l'opcode butterlp. Il utilise le fichier *butterlp.csd* [examples/butterlp.csd].

### Exemple 106. Exemple de l'opcode butterlp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o butterlp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; White noise signal
```

```
asig rand 0.5
    outs asig, asig

endin

instr 2 ; filtered noise

asig rand 0.7
alp butterlp asig, 1000 ;cutting frequencies above 1 KHz
    outs alp, alp

endin

</CsInstruments>
<CsScore>

i 1 0 2
i 2 2.5 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*butterbp, butterbr, butterhp*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Existait dans la version 3.30

Paramètres de taux audio introduits dans la version 6.02

Octobre 2013.

# button

button — Contrôles sur l'écran.

## Description

Contrôles sur l'écran. Nécessite Winsound ou TCL/TK.

## Syntaxe

```
kres button knum
```

## Exécution

Noter que cet opcode n'est pas disponible sous Windows à cause de l'implémentation des tuyaux sur ce système.

*kres* -- valeur du contrôle bouton. Si le bouton a été enfoncé depuis la dernière k-période, retourne 1, sinon 0.

*knun* -- le numéro du bouton. S'il n'existe pas, il apparaît sur l'écran à l'initialisation.

## Exemples

Voici un exemple de l'opcode checkbox. Il utilise le fichier *checkbox.csd* [examples/checkbox.csd].

### Exemple 107. Exemple simple de l'opcode checkbox.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o checkbox.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 10
nchnls = 2

instr 1
  kq init 0
  ; Get the value from the checkbox.
  k1 checkbox 1

  ; If the checkbox is selected then k2=440, otherwise k2=880.
  k2 = (k1 == 0 ? 440 : 880)

  a1 oscil 10000, k2, 1
  outs a1, a1
```

```
    kg button 1
    schedkwhen kg, 0, 1, 2, 0, 0
endin

instr 2
    exitnow
endin

</CsInstruments>
<CsScore>

; sine wave.
f 1 0 32768 10 1

i 1 0 1000
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*checkbox*

## Crédits

Auteur : John ffitch  
Université de Bath, Codemist. Ltd.  
Bath, UK  
Septembre 2000

Nouveau dans la version 4.08 de Csound

# **buzz**

**buzz** — La sortie est un ensemble de partiels sinus en relation harmonique.

## **Description**

La sortie est un ensemble de partiels sinus en relation harmonique.

## **Syntaxe**

```
ares buzz xamp, xcps, knh, ifn [, iphs]
```

## **Initialisation**

*ifn* -- numéro de la table d'une fonction stockée contenant une onde sinus. Une grande table d'au moins 8192 points est recommandée.

*iphs* (facultatif, par défaut 0) -- phase initiale de la fréquence fondamentale, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative l'initialisation de la phase sera ignorée. La valeur par défaut est zéro.

## **Exécution**

*xamp* -- amplitude

*xcps* -- fréquence en cycles par seconde

Les unités *buzz* génèrent un ensemble additif de partiels cosinus en relation harmonique de fréquence fondamentale *xcps*, et dont les amplitudes sont pondérées de telle façon que la crête de leur somme égale *xamp*. Le choix et l'importance des partiels sont déterminés par les paramètres de contrôle suivants :

*knh* -- nombre total d'harmoniques demandés. Nouveau dans la version 3.57 de Csound, *knh* vaut un par défaut. Si *knh* est négatif, sa valeur absolue est utilisée.

*buzz* et *gbuzz* sont utiles comme sources de son complexe dans la synthèse soustractive. *buzz* est un cas particulier du plus général *gbuzz* dans lequel *klh* = *kmul* = 1 ; il produit ainsi un ensemble de *knh* harmoniques de même importance, commençant avec le fondamental. (C'est un train d'impulsions à bande de fréquence limitée ; si les partiels vont jusqu'à la fréquence de Nyquist, c'est-à-dire  $knh = \text{int}(sr / 2 / \text{fréq. fondamentale})$ , le résultat est un train d'impulsions réelles d'amplitude *xamp*.)

Bien que l'on puisse faire varier *knh* durant l'exécution, sa valeur interne est nécessairement un entier ce qui peut provoquer des « pops » dûs à des discontinuités dans la sortie. *buzz* peut être modulé en amplitude et/ou en fréquence soit par des signaux de contrôle soit par des signaux audio.

Nota Bene : cette unité a son pendant avec *GENII*, dans lequel le même ensemble de cosinus peut être stocké dans une table de fonction qui sera lue par un oscillateur. Bien que plus efficace en termes de calcul, le train d'impulsions stocké a un contenu spectral fixe, non variable dans le temps comme celui décrit ci-dessus.

## **Exemples**

Voici un exemple de l'opcode *buzz*. Il utilise le fichier *buzz.csd* [examples/buzz.csd].

## Exemple 108. Exemple de l'opcode buzz.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o buzz.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kcps = 110
ifn = 1

knh line p4, p3, p5
asig buzz 1, kcps, knh, ifn
  outs asig, asig
endin

</CsInstruments>
<CsScore>

;sine wave.
f 1 0 16384 10 1

i 1 0 3 20 20
i 1 + 3 3 3
i 1 + 3 10 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*gbuzz*

## Crédits

Septembre 2003. Merci à Kanata Motohashi pour avoir corrigé les mentions du paramètre *kmul*.

# c2r

c2r — Conversion du format complexe au format réel.

## Description

Convertit un tableau de valeurs complexes en un tableau de valeurs réelles, en supprimant la partie imaginaire. La taille du tableau de sortie est la moitié de celle du tableau d'entrée. C'est une opération utilitaire pour faciliter les opérations sur des valeurs complexes dont le résultat est purement réel.

## Syntaxe

```
kout[] c2r kin[]
```

## Exécution

*kout[]* -- tableau de sortie contenant les valeurs réelles. Créé s'il n'existe pas.

*kin[]* -- tableau d'entrée contenant les valeurs complexes à parties réelle et imaginaire.

## Exemples

Voici un exemple de l'opcode c2r. Il utilise le fichier *c2r.csd* [examples/c2r.csd].

### Exemple 109. Exemple de l'opcode c2r.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-d -o dac
</CsOptions>
<CsInstruments>
ksmps = 64

instr 1
  ifftsize = 1024
  kcnt init 0
  kIn[] init ifftsize
  kOut[] init ifftsize

  a1 oscili 0dbfs/2, 440

  if kcnt >= ifftsize then
    kCmplx[] r2c kIn
    kSpec[] fft kCmplx
    kCmplx fftinv kSpec
    kOut c2r kCmplx
    kcnt = 0
  endif

  kIn[] shiftin a1
  a2 shiftout kOut
  kcnt += ksmps
```



```
    out a2  
endin  
</CsInstruments>  
<CsScore>  
i1 0 10  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux.

## Crédits

Auteur : Victor Lazzarini  
NUI Maynooth  
2014

Nouveau dans la version 6.04

# cabasa

cabasa — Modèle semi-physique d'un son de cabasa.

## Description

*cabasa* est un modèle semi-physique d'un son de cabasa. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntaxe

```
ares cabasa iamp, idettack [, inum] [, idamp] [, imaxshake]
```

## Initialisation

*iamp* -- Amplitude de la sortie. Note : comme ces instruments sont de type stochastique, ce n'est qu'une approximation.

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (optional) -- (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 512.

*idamp* (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

$$\text{damping\_amount} = 0.998 + (\text{idamp} * 0.002)$$

La valeur par défaut de *damping\_amount* est 0,997 ce qui signifie que la valeur par défaut de *idamp* est -0,5. Le maximum de *damping\_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 1,0.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

*imaxshake* (facultatif) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

## Exemples

Voici un exemple de l'opcode cabasa. Il utilise le fichier *cabasa.csd* [examples/cabasa.csd].

### Exemple 110. Exemple de l'opcode cabasa.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
```

```

; -o cabasa.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

inum = p4
idamp = p5
asig cabasa 0.9, 0.01, inum, idamp
outs asig, asig

endin

</CsInstruments>
<CsScore>

i1 1 1 48 .95
i1 + 1 1000 .5

e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*crunch, sandpaper, sekere, stix*

## Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)

Adapté par John ffitich

Université de Bath, Codemist Ltd.

Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

# cauchy

cauchy — Générateur de nombres aléatoires de distribution de Cauchy.

## Description

Générateur de nombres aléatoires de distribution de Cauchy. C'est un générateur de bruit de classe x.

## Syntaxe

```
ares cauchy kalpha
```

```
ires cauchy kalpha
```

```
kres cauchy kalpha
```

## Exécution

*kalpha* -- contrôle la dispersion centrée sur zéro (un grand *kalpha* = une grande dispersion). Donne en sortie des nombres positifs et négatifs.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetst: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode cauchy. Il utilise le fichier *cauchy.csd* [examples/cauchy.csd].

### Exemple 111. Exemple de l'opcode cauchy.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o cauchy.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1    ; every run time same values

kalpha cauchy 100
```

```

    printk .2, kalpha ; look
    aout oscili 0.8, 440+kalpha, 1 ; & listen
    outs aout, aout
    endin

    instr 2 ; every run time different values

    seed 0
    kalpha cauchy 100
    printk .2, kalpha ; look
    aout oscili 0.8, 440+kalpha, 1 ; & listen
    outs aout, aout
    endin
</CsInstruments>
<CsScore>
; sine wave
f 1 0 16384 10 1

i 1 0 2
i 2 3 2
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

i 1 time 0.00267: -0.20676
i 1 time 0.20267: -0.28814
i 1 time 0.40267: 0.61651
i 1 time 0.60267: -18.18552
i 1 time 0.80267: 1.18140
i 1 time 1.00000: -0.75432
i 1 time 1.20267: -0.02002
i 1 time 1.40267: 0.01785
i 1 time 1.60267: -0.48834
i 1 time 1.80000: 9.69401
i 1 time 2.00000: -0.41257
Seeding from current time 3112109827
i 2 time 3.00267: -0.46887
i 2 time 3.20267: 0.06189
i 2 time 3.40267: -0.40303
i 2 time 3.60000: 0.89312
i 2 time 3.80267: -0.40374
i 2 time 4.00000: 0.86557
i 2 time 4.20000: 0.09192
i 2 time 4.40267: -0.16748
i 2 time 4.60267: 0.30133
i 2 time 4.80267: 0.31657
i 2 time 5.00000: 0.44681

```

## Voir aussi

*seed, betarand, bexprnd, exprand, gauss, linrand, pcauchy, poisson, trirand, unirand, weibull*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Existait dans la 3.30

# cauchy

cauchy — Générateur de nombres aléatoires à distribution de Cauchy avec interpolation.

## Description

Générateur de nombres aléatoires à distribution de Cauchy avec interpolation contrôlée entre les valeurs. C'est un générateur de bruit de classe x.

## Syntaxe

```
ares cauchy kalpha, xamp, xcps
```

```
ires cauchy kalpha, xamp, xcps
```

```
kres cauchy kalpha, xamp, xcps
```

## Exécution

*kalpha* -- contrôle la dispersion centrée sur zéro (un grand *kalpha* = une grande dispersion). Donne en sortie des nombres positifs et négatifs.

Pour des explications plus détaillées, voir :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

*xamp* -- intervalle de distribution des nombres aléatoires.

*xcps* -- fréquence à laquelle de nouveaux nombres sont générés.

## Exemples

Voici un exemple de l'opcode cauchy. Il utilise le fichier *cauchy.csd* [examples/cauchy.csd].

### Exemple 112. Exemple de l'opcode cauchy.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime output leave only the line below:
; -o exprand.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
```

```

nchnls = 2
0dbfs = 1

instr 1
klambda cauchy 100, 1, 3
  printk2 klambda ; look
aout oscili 0.8, 440+klambda, 1 ; & listen
  outs aout, aout
endin

</CsInstruments>
<CsScore>
; sine wave
f 1 0 16384 10 1

i 1 0 4
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*cauchy*

## Crédits

Auteur : John ffitch  
 Bath  
 Mai 2011  
 Nouveau dans la version 5.14

# ceil

ceil — Retourne le plus petit entier supérieur ou égal à  $x$ .

## Description

Retourne le plus petit entier supérieur ou égal à  $x$ .

## Syntaxe

**ceil**( $x$ ) (argument au taux d'initialisation, de contrôle ou audio)

**ceil**( $k/i[]$ ) ( $k$ - ou  $i$ -tableau)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode ceil. Il utilise le fichier *ceil.csd* [examples/ceil.csd].

### Exemple 113. Exemple de l'opcode ceil.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too

</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1

inum = p4
iceil = ceil(inum)
print iceil

endin

</CsInstruments>
<CsScore>

i 1 0 0 1
i . . . 0.999999
i . . . 0.000001
i . . . 0
i . . . -0.0000001
```



```
i . . . -0.9999999
i . . . -1
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
instr 1:  iceil = -1.000
instr 1:  iceil = 1.000
instr 1:  iceil = 1.000
instr 1:  iceil = 1.000
instr 1:  iceil = 0.000
instr 1:  iceil = 0.000
instr 1:  iceil = 0.000
```

## Voir aussi

*abs, exp, int, log, log10, i, sqrt*

## Crédits

Auteur : Istvan Varga  
Nouveau dans Csound 5  
2005

# cell

cell — Automate cellulaire

## Description

Opcodes du greffon cellular.

Automate cellulaire à une dimension. Cet opcode est une version modifiée de *vcella* par Gabriel Maldonado.

## Syntaxe

```
cell ktrig, kreinit, ioutFunc, initStateFunc, iRuleFunc, ielements
```

## Initialisation

*ioutFunc* -- numéro de la table dans laquelle l'état de chaque cellule est stocké.

*initStateFunc* -- numéro de la table contenant l'état initial des cellules.

*iRuleFunc* -- numéro de la table contenant la règle sur 8 bit.

*ielements* -- nombre total de cellules dans une ligne.

## Exécution

*ktrig* -- signal de déclenchement. Chaque fois qu'il est différent de zéro, une nouvelle génération de cellules est évaluée.

*kreinit* -- signal de réinitialisation. Chaque fois qu'il est différent de zéro, toutes les cellules sont mises dans l'état correspondant de *initStateFunc*.

*cell* modélise un automate cellulaire classique à une dimension et stocke l'état de chaque cellule dans une table identifiée par *ioutFunc*.

*initStateFunc* est un vecteur d'entrée contenant l'état initial de la ligne de cellules, tandis que *iRuleFunc* est un vecteur d'entrée contenant la règle choisie dans sa forme binaire (bit de moindre poids en premier).

Une nouvelle génération de cellules est évaluée chaque fois que *ktrig* contient une valeur non nulle. On peut aussi forcer l'état de toutes les cellules à l'état correspondant dans *initStateFunc* chaque fois que *kreinit* contient une valeur non nulle.

Noter que l'on suppose que chaque cellule peut être dans un état parmi deux (1="vivante", 0="morte"), bien que des valeurs fractionnaires peuvent également fonctionner, car elles sont tronquées.

## Exemples

Voici un exemple simple de l'opcode cell. Il utilise le fichier *cell.csd* [examples/cell.csd].

## Exemple 114. Un exemple simple de l'opcode cell.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
;-odac      -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o cell.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
; cell.csd by Gleb Rogozinsky

sr      = 44100
kr      = 4410
ksmps   = 10
nchnls  = 1
0dbfs   = 1

; Cellular automaton-driven synthesis in spectral domain
instr 1

    iatt      = p4                      ; envelope attack time
    isus      = p5                      ; envelope sustain time
    irel      = p6                      ; envelope release time
    ivol      = p7                      ; overall volume

; create some white noise
asig      rand      0.8

; spectral analysis of asig
fsig      pvsanal   asig, 2048, 1024, 2048, 0 ; get a vector of magnitudes

; calculate cellular automaton state
kfreq     line      50, 5, 1            ; variable CA triggering frequency
ktrig     metro     kfreq               ; trigger the CA to update cells
          cell      ktrig, 0, 3, 1, 2, 2048 ; cells are written into ftable 3

; use current row of cells in spectral domain
fmas      pvstencil fsig, 0, 1, 3       ; apply spectral mask
aout      pvsynth   fmas               ; reconstruct time signal

; apply envelope and out signal
kenv      expseg    .001, iatt, 1, isus, 1, irel, .001
          out      aout*kenv*ivol

endin

</CsInstruments>
<CsScore>

; This example uses one-dimensional cellular automaton
; to produce structures in spectral domain

; We have to prepare initial row of cells.
; One alive cell is enough to produce a simple fractal,
; so two alive cells will make structure more sophisticated
f1 0 2048 7 0 150 0 0 1 1 1 0 0 45 0 0 1 1 1 0 0

; The CA rule is used as follows:
; the states (values) of each cell are summed with their neighboring cells.
; Each sum is used as an index to read a next state of cell
; from the rule table.
; Let us try rule # 129 (LSB binary 1 0 0 0 0 0 0 1).
```

```
; This rule will produce a fractal structure for single active cell
; For more rules see http://mathworld.wolfram.com/ElementaryCellularAutomaton.html
f2 0 8 -2 1 0 0 0 0 0 1
; Try some different rules i.E. 254 (0 1 1 1 1 1 1 1) or 169 (1 0 0 1 0 1 0 1)

; Prepare the output table of ielements size
f3 0 2048 10 0

; Time to make it sound!
i1 0 13 0.3 7 3 1
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Gleb Rogozinsky  
Octobre 2011

Nouveau dans la version 5.16.6 de Csound.

# cent

cent — Calcule un facteur pour élever/abaisser une fréquence d'un certain nombre de cents.

## Description

Calcule un facteur pour élever/abaisser une fréquence d'un certain nombre de cents.

## Syntaxe

`cent (x)`

Cette fonction travaille aux taux-i, -k et -a.

## Initialisation

*x* -- une valeur exprimée en cents.

## Exécution

La valeur retournée par la fonction *cent* est un facteur. On peut multiplier une fréquence par ce facteur pour l'élever/l'abaisser du nombre de cents spécifié.

## Exemples

Voici un exemple de l'opcode *cent*. Il utilise le fichier *cent.csd* [examples/cent.csd].

### Exemple 115. Exemple de l'opcode cent.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o cent.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; original pitch

iroot = 440 ; root note = A (440 Hz)

print iroot ;print out

asig oscili 0.6, iroot, 1
```

```

    outs asig, asig

endin

instr 2

iroot  = 440 ; root note = A (440 Hz)
icents = p4 ; change root note by 300 and 1200 cents

ifactor = cent(icents) ; calculate new note
inew    = iroot * ifactor

print iroot ; Print all
print ifactor
print inew

asig oscili 0.6, inew, 1
    outs asig, asig

endin

</CsInstruments>
<CsScore>
; sine wave
f1 0 32768 10 1

i 1 0 2 ;no change
i 2 2.5 2 300 ;note = C above A
i 2 5 2 1200 ;1 octave higher

e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra ces lignes :

```

instr 1:  iroot = 440.000

instr 2:  iroot = 440.000
instr 2:  ifactor = 1.189
instr 2:  inew = 523.251

instr 2:  iroot = 440.000
instr 2:  ifactor = 2.000
instr 2:  inew = 880.000

```

## Voir aussi

*db, octave, semitone*

Nouveau dans la version 4.16

# centroid

centroid — Calcule le barycentre spectral d'un signal.

## Description

Calcul du barycentre spectral d'un signal audio, déclenché par un signal d'excitation.

## Syntaxe

```
kcent centroid asig, ktrig, ifftsize
```

## Initialisation

*ifftsize* -- taille de la TFR. Les valeurs qui ne sont pas des puissances de deux sont converties à la première puissance de deux qui n'est pas inférieure à *ifftsize*.

## Exécution

*kcent* -- le barycentre spectral en Hz.

*asig* -- un signal audio en entrée.

*ktrig* -- 1 pour calculer un nouveau barycentre, 0 pour passer outre et retourner la valeur précédente.

## Exemples

Voici un exemple de l'opcode centroid. Il utilise le fichier *centroid.csd* [examples/centroid.csd].

### Exemple 116. Exemple de l'opcode centroid.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o centroid.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 16384, 10, 1

instr 1

ktrig init 1      ;calculate centroid
a1 oscil3 0.5, p4, giSine ;of the sine wave
```

```
k1 centroid a1, ktrig, 16384
asig oscil3 0.5, k1, giSine
printk2 k1 ;print & compare:
outs a1, asig ;left = original, right = centroid signal

endin
</CsInstruments>
<CsScore>

i1 0 2 20
i1 + 2 200
i1 + 2 2000
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvscent*

## Crédits

Auteur : V Lazzarini;  
Août 2012



# ceps

**ceps** — Calcule le cepstre d'un tableau en entrée, avec coefficients de liftrage facultatifs.

## Description

## Syntaxe

```
keps[] ceps kmags[][[, icoefs]
```

## Initialisation

*icoefs* -- le nombre de coefficients retenus dans le cepstre en sortie. Par défaut, aucun coefficient n'est liftré.

## Exécution

*keps[]* -- le cepstre en sortie, un tableau de taille N+1, où N est une puissance de deux.

*kmags[]* -- un tableau en entrée contenant N+1 amplitudes.

## Exemples

Voici un exemple d'utilisation de l'opcode *ceps*. Il utilise le fichier *ceps.csd* [examples/ceps.csd].

### Exemple 117. Exemple de l'opcode *ceps*.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

instr 1

a1 diskin "fox.wav",1,0,1
k1 randh 80, 2.5
a2 vco2 1, 220+k1
kfsig[] init 1026
fsig pvsanal a1,1024,256,1024,1
fsig2 pvsanal a2,1024,256,1024,1
kf pvs2tab kfsig,fsig
keps[] ceps c2r(kfsig),30
kenv[] cepsinv keps
fenv tab2pvs r2c(kenv)
fvoc pvsfilter fsig2, fenv, 1
asig pvsynth fvoc

out asig
endin

</CsInstruments>
```

```
<CsScore>  
i1 0 60  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn, pvspitch*

## Crédits

Auteur : Victor Lazzarini  
2014

# cepsinv

cepsinv — Calcule l'inverse du cepstre dans un tableau.

## Description

## Syntaxe

```
kenv cepsinv keps[ ]
```

## Exécution

*keps[ ]* -- le cepstre en entrée, un tableau de taille N+1, où N est une puissance de deux, contenant les coefficients cepstraux.

*kenv* -- l'inverse du cepstre (enveloppe spectrale), un tableau de N+1 amplitudes.

## Exemples

Voici un exemple d'utilisation de l'opcode *cepsinv*. Il utilise le fichier *cepsinv.csd* [exemples/cepsinv.csd].

### Exemple 118. Exemple de l'opcode *cepsinv*.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

instr 1

a1 diskin "fox.wav", 1, 0, 1
k1 randh 80, 2.5
a2 vco2 8, 220+k1
fsig pvsanal a1, 1024, 256, 1024, 1
fsig2 pvsanal a2, 1024, 256, 1024, 1
keps[] pvsceps fsig, 30
kenv[] cepsinv keps
fenv tab2pvs r2c(kenv)
fvoc pvsfilter fsig2, fenv, 1
asig pvsynth fvoc

    out asig
endin

</CsInstruments>
<CsScore>
i1 0 60
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
2014

# cggoto

cggoto — Transfert conditionnel du contrôle à chaque passage.

## Description

Transfère le contrôle vers *label* à chaque passage. (Combinaison de *cigoto* et de *ckgoto*)

## Syntaxe

**cggoto** condition, label

où *label* se trouve dans le même bloc d'instrument et n'est pas une expression, et où *condition* utilise un des opérateurs relationnels (<, =, <=, ==, !=) (et = par commodité, voir aussi *Valeurs Conditionnelles*).

## Exemples

Voici un exemple de l'opcode cggoto. Il utilise le fichier *cggoto.csd* [examples/cggoto.csd].

### Exemple 119. Exemple de l'opcode cggoto.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; -o cggoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = p4

  ; If i1 is equal to one, play a high note.
  ; Otherwise play a low note.
  cggoto (i1 == 1), highnote

lownote:
  a1 oscil 10000, 220, 1
  goto playit

highnote:
  a1 oscil 10000, 440, 1
  goto playit

playit:
  out a1
endin
```

```
</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; Play lownote for one second.
i 1 0 1 1
; Play highnote for one second.
i 1 1 1 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*cigoto, ckgoto, cngoto, if, igoto, kgoto, tigoto, timeout*

## Crédits

Exemple écrit par Kevin Conder.

# chanctrl

chanctrl — Prend la valeur actuelle d'un contrôleur d'un canal MIDI.

## Description

Prend la valeur actuelle d'un contrôleur et le configure optionnellement dans un intervalle spécifié.

## Syntaxe

```
ival chanctrl ichnl, ictlno [, ilow] [, ihigh]
kval chanctrl ichnl, ictlno [, ilow] [, ihigh]
```

## Initialisation

*ichnl* -- le canal MIDI (1-16).

*ictlno* -- le numéro du contrôleur MIDI (0-127).

*ilow*, *ihigh* -- Limites inférieure et supérieure de la configuration

## Exemples

Voici un exemple de l'opcode chanctrl. Il utilise le fichier *chanctrl.csd* [examples/chanctrl.csd].

### Exemple 120. Exemple de l'opcode chanctrl.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  No messages  MIDI in
-odac          -d          -M0   ;;RT audio I/O with MIDI in
;-iadc         ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o chanctrl.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; press your midi keyboard and move your midi controller to see result

ichnl = 1 ;MIDI note inputs on channel 1
ictlno = 7 ;use midi volume controller
kch chanctrl ichnl, 7 ;to control amplitude of oscil
printk2 kch

asig oscil kch*(1/127), 220, 1
outs asig, asig
```

```
    endin

    </CsInstruments>
    <CsScore>
    ;Dummy f-table to give time for real-time MIDI events
    f 0 30
    ;sine wave.
    f 1 0 16384 10 1
    e

    </CsScore>
    </CsoundSynthesizer>
```

## Crédits

Auteur : Mike Berry  
Collège Mills  
Mai, 1997

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.



# changed

changed — Détecteur de changement d'un signal de taux-k.

## Description

Cet opcode renvoie un signal de déclenchement indiquant tout changement d'un de ses arguments de taux-k. Utile avec les widgets valuateurs ou les contrôleurs MIDI.

## Syntaxe

```
ktrig changed kvar1 [, kvar2,..., kvarN]
```

## Exécution

*ktrig* - Renvoie la valeur 1 lorsqu'un des signaux de taux-k a changé, sinon renvoie la valeur 0.

*kvar1* [, *kvar2*,..., *kvarN*] - variables de taux-k dont les changements sont surveillés.



### Note

Si chaque kvar est différente de zéro lors du premier cycle-k de l'appel de *changed*, un changement sera rapporté. Pour un comportement plus cohérent, utiliser *changed2*.

## Exemples

Voici un exemple de l'opcode *changed*. Il utilise le fichier *changed.csd* [examples/changed.csd].

### Exemple 121. Exemple de l'opcode *changed*.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o changed.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1

ksig oscil 2, 0.5, 1
kint = int(ksig)
ktrig changed kint
printk 0.2, kint
printk2 ktrig

endin

</CsInstruments>
```

```
<CsScore>
f 1 0 1024 10 1
i 1 0 20

e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci:

```
i 1 time 0.00267: 0.00000
il 0.00000
il 1.00000
il 0.00000
i 1 time 0.20267: 1.00000
i 1 time 0.40267: 1.00000
il 1.00000
il 0.00000
i 1 time 0.60267: 1.00000
i 1 time 0.80267: 1.00000
il 1.00000
il 0.00000
i 1 time 1.00000: 0.00000
il 1.00000
il 0.00000
i 1 time 1.20267: -1.00000
i 1 time 1.40267: -1.00000
il 1.00000
il 0.00000
i 1 time 1.60267: -1.00000
i 1 time 1.80000: -1.00000
il 1.00000
il 0.00000
i 1 time 2.00000: -0.00000
il 1.00000
il 0.00000
.....
```

## Crédits

Ecrit par Gabriel Maldonado.

Exemple écrit par Andrés Cabrera.

Nouveau dans Csound 5.7.

# changed2

changed2 — Détecteur de changement d'un signal de taux-k.

## Description

Cet opcode renvoie un signal de déclenchement indiquant tout changement d'un de ses arguments de taux-k ou d'une valeur dans un tableau. Utile avec les widgets valueurs ou les contrôleurs MIDI.

## Syntaxe

```
ktrig changed2 kvar1 [, kvar2,..., kvarN]
ktrig changed2 karr[]
ktrig changed2 aarr[]
```

## Exécution

*ktrig* - Renvoie la valeur 1 lorsqu'un des signaux de taux-k ou qu'une valeur du tableau a changé, sinon renvoie la valeur 0.

*kvar1* [, *kvar2*,..., *kvarN*] - variables de taux-k dont les changements sont surveillés.

*xarray* - n'importe quel type de tableau.

Au contraire de l'opcode *changed* cet opcode ne rapporte jamais le premier cycle comme un changement.

## Exemples

Voici deux exemples de l'opcode *changed2*. Il utilise les fichiers *changed2.csd* [examples/changed2.csd] et *changed2a.csd* [examples/changed2a.csd].

### Exemple 122. Exemple de l'opcode *changed2*.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o changed.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1

ksig oscil 2, 0.5, 1
kint = int(ksig)
ktrig changed2 kint
printk 0.2, kint
```

```

        printk2 ktrig

    endin

</CsInstruments>
<CsScore>
f 1 0 1024 10 1
i 1 0 20

e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci:

```

i 1 time 0.00267: 0.00000
il 0.00000
il 1.00000
il 0.00000
i 1 time 0.20267: 1.00000
i 1 time 0.40267: 1.00000
il 1.00000
il 0.00000
i 1 time 0.60267: 1.00000
i 1 time 0.80267: 1.00000
il 1.00000
il 0.00000
i 1 time 1.00000: 0.00000
il 1.00000
il 0.00000
i 1 time 1.20267: -1.00000
i 1 time 1.40267: -1.00000
il 1.00000
il 0.00000
i 1 time 1.60267: -1.00000
i 1 time 1.80000: -1.00000
il 1.00000
il 0.00000
i 1 time 2.00000: -0.00000
il 1.00000
il 0.00000
.....

```

### Exemple 123. Exemple de l'opcode changed2.

```

<CsoundSynthesizer>

<CsOptions>
-n
</CsOptions>
<CsInstruments>
gkArray[][] init 2,3
gkArray fillarray 1,2,3,7,6,5

instr 1
k1 changed2 gkArray
if k1==1 then
    printks "An element in the array changed", 0
endif
endin

instr 2; change value of channel 'step1'
    gkArray[1][0] = 3
endin

```

```
</CsInstruments>  
<CsScore>  
i1 0 100  
i2 4 .1  
</CsScore>  
  
</CsoundSynthesizer>
```

## Crédits

Ecrit par John ffitch.

Exemple écrit par Andrés Cabrera.

Nouveau dans Csound 6.7.

# chani

chani — Lit des données depuis le bus logiciel.

## Description

Lit des données depuis un canal du bus logiciel entrant.

## Syntaxe

```
kval chani kchan
```

```
aval chani kchan
```

## Exécution

*kchan* -- un entier positif indiquant quel canal du bus logiciel lire.

Noter que les bus logiciels entrant et sortant sont indépendants et qu'ils ne sont pas des bus mélangeurs. De plus, les bus au taux-k et au taux-a sont indépendants. La dernière valeur reste disponible jusqu'à ce qu'une nouvelle valeur soit écrite. Il n'y a pas de limite imposée au nombre de bus mais comme ils consomment de la mémoire, il est préférable d'en utiliser un petit nombre.

## Exemples

L'exemple montre l'utilisation du bus logiciel comme signal de contrôle asynchrone pour fixer la fréquence de coupure du filtre. On suppose qu'un programme externe utilisant l'API fournit les valeurs.

```
sr = 44100
kr = 100
ksmps = 1

instr 1
  kc  chani      1
  a1  oscil      p4, p5, 100
  a2  lowpass2   a1, kc, 200
  out  a2
endin
```

## Crédits

Auteur : John fitch  
2005

Nouveau dans Csound 5.00

# chano

chano — Envoie des données vers le bus logiciel sortant.

## Description

Envoie des données vers un canal du bus logiciel sortant.

## Syntaxe

```
chano kval, kchan
```

```
chano aval, kchan
```

## Exécution

*xval* --- valeur à transmettre.

*kchan* -- un entier positif indiquant sur quel canal du bus logiciel écrire.

Noter que les bus logiciels entrant et sortant sont indépendants et qu'ils ne sont pas des bus mélangeurs. De plus, les bus au taux-k et au taux-a sont indépendants. La dernière valeur reste disponible jusqu'à ce qu'une nouvelle valeur soit écrite. Il n'y a pas de limite imposée au nombre de bus mais comme ils consomment de la mémoire, il est préférable d'en utiliser un petit nombre.

## Exemples

L'exemple montre l'utilisation du bus logiciel comme canal de sortie d'un signal audio. On suppose qu'un programme externe utilisant l'API reçoit les valeurs.

```
sr = 44100
kr = 100
ksmps = 1

instr 1
  al oscil p4, p5, 100
    chano 1, al
endin
```

## Crédits

Auteur : John ffitich  
2005

Nouveau dans Csound 5.00

# cbrt

cbrt — Fonction racine cubique.

## Description

Retourne la valeur de la racine cubique de son argument.

## Syntaxe

```
iRes[] cbrt iArg
```

```
kRes[] cbrt kArg
```

## Initialisation

*iArg[]* -- l'argument.

## Exécution

*kArg[]* -- l'argument.

## Exemples

Voici un exemple de l'opcode cbrt. Il utilise le fichier *cbrt.csd* [examples/cbrt.csd].

### Exemple 124. Exemple de l'opcode cbrt.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

instr 1

iArg[] fillarray 1,2,3
iRes[] cbrt iArg
ik init 0

while ik < lenarray(iRes) do
  print iRes[ik]
  ik += 1
od

endin

</CsInstruments>
<CsScore>
i1 0 0
</CsScore>
</CsoundSynthesizer>
```



## Crédits

Auteur : Victor Lazzarini  
2017

# chebyshevpoly

chebyshevpoly — Evalue efficacement la somme de polynomes de Tchebychev d'ordre arbitraire.

## Description

L'opcode *chebyshevpoly* calcule la valeur d'une expression polynomiale d'une variable d'entrée unique de taux-*a*. Cette expression est constituée d'une combinaison linéaire des *N* premiers polynome de Tchebychev de première espèce. Chaque polynome de Tchebychev,  $T_n(x)$ , est pondéré par un coefficient de taux-*k*, *kn*, de façon à ce que l'opcode calcule la somme de n'importe quel nombre de termes de la forme  $kn * T_n(x)$ . Ainsi, l'opcode *chebyshevpoly* permet d'effectuer la distorsion non-linéaire d'un signal audio avec une fonction de transfert *dynamique* donnant un contrôle précis du contenu harmonique de la sortie.

## Syntaxe

```
aout chebyshevpoly ain, k0 [, k1 [, k2 [...]]]
```

## Exécution

*ain* -- le signal d'entrée utilisé comme variable indépendante des polynomes de Tchebychev ("x").

*aout* -- le signal de sortie ("y").

*k0*, *k1*, *k2*, ... -- multiplicateurs de taux-*k* pour chaque polynome de Tchebychev.

Cet opcode est très utile pour la distorsion non-linéaire dynamique d'un signal audio. Les techniques traditionnelles de distorsion non-linéaire utilisent une table à consulter pour la fonction de transfert -- habituellement une somme de polynomes de Tchebychev. Lorsqu'une onde sinusoïdale dont l'amplitude couvre toute l'échelle est utilisée comme index pour lire la table, le spectre harmonique précis défini par les poids des polynomes de Tchebychev est produit. On obtient un spectre dynamique en variant l'amplitude de l'onde sinusoïdale en entrée, mais cela produit un changement non-linéaire du spectre.

En calculant directement les polynomes de Tchebychev, l'opcode *chebyshevpoly* donne plus de contrôle sur le spectre, et le nombre d'harmoniques ajoutés à l'entrée peut varier dans le temps. La valeur de chaque coefficient *kn* contrôle directement l'amplitude du nième harmonique si l'entrée *ain* est une onde sinus d'amplitude = 1.0. Cela fait de *chebyshevpoly* un outil de synthèse additive efficace pour *N* partiels, ne nécessitant qu'un oscillateur au lieu de *N*. On peut aussi changer l'amplitude ou la forme d'onde du signal d'entrée pour obtenir différents effets de distorsion non-linéaire.

Si nous considérons que le paramètre d'entrée *ain* est "x" et que la sortie *aout* est "y", alors l'opcode *chebyshevpoly* calcule l'équation suivante :

$$y = k0*T0(x) + k1*T1(x) + k2*T2(x) + k3*T3(x) + \dots$$

où les  $T_n(x)$  sont définis par la relation de récurrence

$$\begin{aligned} T0(x) &= 1, \\ T1(x) &= x, \\ Tn(x) &= 2x*T[n-1](x) - T[n-2](x) \end{aligned}$$

On peut trouver plus d'information sur les polynômes de Tchebychev sur Wikipedia à [http://en.wikipedia.org/wiki/Chebyshev\\_polynomial](http://en.wikipedia.org/wiki/Chebyshev_polynomial) [http://en.wikipedia.org/wiki/Chebyshev\_polynomial]

## Voir aussi

*polynomial, mac maca*

## Exemples

Voici un exemple de l'opcode `chebyshevpoly`. Il utilise le fichier *chebyshevpoly.csd* [examples/chebyshevpoly.csd].

### Exemple 125. Exemple de l'opcode `chebyshevpoly`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime output leave only the line below:
; -o chebyshevpoly.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

; time-varying mixture of first six harmonics
instr 1
; According to the GEN13 manual entry,
; the pattern + - - + + - - for the signs of
; the chebyshev coefficients has nice properties.

; these six lines control the relative powers of the harmonics
k1      line      1.0, p3, 0.0
k2      line      -0.5, p3, 0.0
k3      line      -0.333, p3, -1.0
k4      line      0.0, p3, 0.5
k5      line      0.0, p3, 0.7
k6      line      0.0, p3, -1.0

; play the sine wave at a frequency of 256 Hz with amplitude = 1.0
ax      oscili      1, 256, 1

; waveshape it
ay      chebyshevpoly ax, 0, k1, k2, k3, k4, k5, k6

; avoid clicks, scale final amplitude, and output
adeclick linseg      0.0, 0.05, 1.0, p3 - 0.1, 1.0, 0.05, 0.0
outs      ay * adeclick * 10000, ay * adeclick * 10000
endin

</CsInstruments>
<CsScore>
f1 0 32768 10 1 ; a sine wave

i1 0 5
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : Anthony Kozar  
Janvier 2008

Nouveau dans la version 5.08 de Csound.

# checkbox

checkbox — Case à cocher sur l'écran.

## Description

Opcode du greffon control.

Case à cocher sur l'écran. Nécessite Winsound ou Tcl/Tk.

## Syntaxe

```
kres checkbox knum
```

## Exécution

Noter que cet opcode n'est pas disponible sous Windows à cause de l'implémentation des tuyaux sur ce système.

*kres* -- valeur de la case à cocher. Vaut 1 lorsqu'elle est cochée, sinon 0.

*knun* -- le numéro de la case à cocher. Si celle-ci n'existe pas, elle est créée sur l'écran à l'initialisation.

## Exemples

Voici un exemple de l'opcode checkbox. Il utilise le fichier *checkbox.csd* [examples/checkbox.csd].

### Exemple 126. Simple exemple de l'opcode checkbox.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o checkbox.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 10
nchnls = 2

instr 1
  kq init 0
  ; Get the value from the checkbox.
  k1 checkbox 1

  ; If the checkbox is selected then k2=440, otherwise k2=880.
  k2 = (k1 == 0 ? 440 : 880)

  a1 oscil 10000, k2, 1
```

```
outs a1, a1
kg button 1
schedkwhen kg, 0, 1, 2, 0, 0
endin

instr 2
  exitnow
endin

</CsInstruments>
<CsScore>

; sine wave.
f 1 0 32768 10 1

i 1 0 1000
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*button*

## Crédits

Auteur : John ffitch  
Université de Bath, Codemist. Ltd.  
Bath, UK  
Septembre, 2000

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.08 de Csound

# chn

chn — Déclare un canal du bus logiciel nommé.

## Description

Déclare un canal du bus logiciel nommé, en donnant des paramètres facultatifs dans le cas d'un canal de contrôle. Si le canal n'existe pas encore, il est créé avec une valeur initiale de zéro ou une chaîne de caractères vide. Sinon le type (contrôle, audio ou chaînes de caractères) du canal existant doit correspondre à la déclaration ou bien il y aura une erreur d'initialisation. Le mode entrée/sortie d'un canal existant est mis à jour de façon à représenter le OU binaire entre la valeur précédente et la nouvelle.

## Syntaxe

```
chn_k Sname, imode[, itype, idflt, imin, imax, ix, iy, iwidth, iheight, Sattributes]
```

```
chn_a Sname, imode
```

```
chn_s Sname, imode
```

## Initialisation

*imode* -- somme d'au moins un des nombres suivants : 1 pour entrée et 2 pour sortie.

*itype* (facultatif, 0 par défaut) -- sous-type du canal, seulement pour les canaux de contrôle. Les valeurs possibles sont :

- 0 : par défaut / indéfini (*idflt*, *imin* et *imax* sont ignorés)
- 1 : seulement des valeurs entières
- 2 : échelle linéaire
- 3 : échelle exponentielle

*idflt* (facultatif, 0 par défaut) -- valeur par défaut, seulement pour les canaux de contrôle avec *itype* différent de zéro. Doit être supérieur ou égal à *imin* et inférieur ou égal à *imax*.

*imin* (facultatif, 0 par défaut) -- valeur minimale, seulement pour les canaux de contrôle avec *itype* différent de zéro. Doit être différent de zéro pour l'échelle exponentielle (*itype* = 3).

*imax* (facultatif, 0 par défaut) -- valeur maximale, seulement pour les canaux de contrôle avec *itype* différent de zéro. Doit être supérieur à *imin*. Dans le cas d'une échelle exponentielle, il doit également avoir le même signe que *imin*.

*ix* -- position x suggérée pour le contrôleur.

*iy* -- position y suggérée pour le contrôleur.

*iwidth* -- position suggérée de la largeur pour le contrôleur.

*iheight* -- position suggérée de la hauteur pour le contrôleur.

*Sattributes* -- attributs pour le contrôleur.

## Notes

Les paramètres du canal (*imode*, *itype*, *idflt*, *imin* et *imax*) ne sont que des indications pour l'application hôte ou un logiciel externe accédant au bus par l'API, et ils ne restreignent en rien la lecture ou l'écriture sur le canal. De plus, la valeur initiale d'un nouveau canal de contrôle est zéro, quelque soit la valeur de *idflt*.

Il peut être préférable d'utiliser *chnexport* pour communiquer avec un logiciel externe, car il permet un accès direct aux variables de l'orchestre exportées comme des canaux du bus, ce qui évite l'utilisation de *chnset* et de *chnget* pour envoyer ou recevoir des données.

## Exécution

**chn\_k**, **chn\_a**, et **chn\_S** déclarent respectivement un canal de contrôle, un canal audio ou un canal de chaînes de caractères.

## Exemples

L'exemple montre l'utilisation du bus logiciel comme signal de contrôle asynchrone pour fixer la fréquence de coupure du filtre. On suppose qu'un programme externe utilisant l'API fournit les valeurs.

```
sr = 44100
kr = 100
ksmps = 1

chn_k "cutoff", 1, 3, 1000, 500, 2000

instr 1
  kc  chnget    "cutoff"
  a1  oscil     p4, p5, 100
  a2  lowpass2  a1, kc, 200
      out      a2
endin
```

## Crédits

Auteur : Istvan Varga  
2005



# chnclear

chnclear — Efface des canaux de sortie audio du bus logiciel nommé.

## Description

Met à zéro une liste de canaux de sortie audio du bus logiciel nommé. Cela implique une déclaration du canal avec *imode=2* (voir aussi *chn\_a*).

## Syntaxe

```
chnclear Sname1[, Sname2, ...]
```

## Initialisation

*Sname\** -- une chaîne de caractères indiquant quel canal du bus logiciel effacer.

## Exemples

Voici un exemple de l'opcode chnclear. Il utilise le fichier *chnclear.csd* [examples/chnclear.csd].

### Exemple 127. Exemple de l'opcode chnclear.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o chnclear.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
;Example by Joachim Heintz
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1; send i-values
    chnset 1, "sio"
    chnset -1, "non"
endin

instr 2; send k-values
kfreq    randomi 100, 300, 1
    chnset kfreq, "cntrfreq"
kbw      =      kfreq/10
    chnset kbw, "bandw"
endin

instr 3; send a-values
anois    rand .1
    chnset anois, "noise"

loop:
idur     random .3, 1.5
    timeout 0, idur, do
```

```

    reinit    loop
do:
  ifreq      random    400, 1200
  iamp       random    .1, .3
  asig       oscils    iamp, ifreq, 0
  aenv       transeg   1, idur, -10, 0
  asine      =         asig * aenv
             chnset    asine, "sine"
    endin

    instr 11; receive some chn values and send again
  ival1      chnget    "sio"
  ival2      chnget    "non"
             print     ival1, ival2
  kcntfreq   chnget    "cntrfreq"
  kbandw     chnget    "bandw"
  anoise     chnget    "noise"
  afilt      reson     anoise, kcntfreq, kbandw
  afilt      balance   afilt, anoise
             chnset    afilt, "filtered"
    endin

    instr 12; mix the two audio signals
  amix1      chnget    "sine"
  amix2      chnget    "filtered"
             chnmix     amix1, "mix"
             chnmix     amix2, "mix"
    endin

    instr 20; receive and reverb
  amix       chnget    "mix"
  aL, aR     freeverb  amix, amix, .8, .5
             outs      aL, aR
    endin

    instr 100; clear
             chnclear   "mix"
    endin

</CsInstruments>
<CsScore>
i 1 0 20
i 2 0 20
i 3 0 20
i 11 0 20
i 12 0 20
i 20 0 20
i 100 0 20
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Istvan Varga  
2006

# chnexport

chnexport — Exporte une variable globale en tant que canal du bus.

## Description

Exporte une variable globale en tant que canal du bus ; le canal ne doit pas déjà exister sinon il y aura une erreur d'initialisation. On appelle normalement cet opcode depuis l'en-tête de l'orchestre ; il permet à l'application hôte de lire et d'écrire directement dans des variables de l'orchestre, sans avoir à utiliser *chnget* ou *chnset* pour copier les données.

## Syntaxe

```
gival chnexport Sname, imode[, itype, idflt, imin, imax]
```

```
gkval chnexport Sname, imode[, itype, idflt, imin, imax]
```

```
gaval chnexport Sname, imode
```

```
gSval chnexport Sname, imode
```

## Initialisation

*imode* -- somme d'au moins un des nombres suivants : 1 pour entrée et 2 pour sortie.

*itype* (facultatif, 0 par défaut) -- sous-type du canal, seulement pour les canaux de contrôle. Les valeurs possibles sont :

- 0: par défaut / indéfini (*idflt*, *imin* et *imax* sont ignorés)
- 1: seulement des valeurs entières
- 2: échelle linéaire
- 3: échelle exponentielle

*idflt* (facultatif, 0 par défaut) -- valeur par défaut, seulement pour les canaux de contrôle avec *itype* différent de zéro. Doit être supérieur ou égal à *imin* et inférieur ou égal à *imax*.

*imin* (facultatif, 0 par défaut) -- valeur minimale, seulement pour les canaux de contrôle avec *itype* différent de zéro. Doit être différent de zéro pour l'échelle exponentielle (*itype* = 3).

*imax* (facultatif, 0 par défaut) -- valeur maximale, seulement pour les canaux de contrôle avec *itype* différent de zéro. Doit être supérieur à *imin*. Dans le cas d'une échelle exponentielle, il doit également avoir le même signe que *imin*.

## Notes

Les paramètres du canal (*imode*, *itype*, *idflt*, *imin* et *imax*) ne sont que des indications pour l'application hôte ou un logiciel externe accédant au bus par l'API, et ils ne restreignent en rien la lecture ou l'écriture sur le canal.

Bien que la variable globale soit utilisée comme argument de sortie, *chnexport* ne la change pas, et ne s'exécute seulement qu'au taux-i. Si la variable n'a pas été préalablement déclarée, elle est créée par Csound avec une valeur initiale de zéro ou une chaîne de caractères nulle.

## Exemples

L'exemple montre l'utilisation du bus logiciel comme signal de contrôle asynchrone pour fixer la fréquence de coupure du filtre. On suppose qu'un programme externe utilisant l'API fournit les valeurs.

```
sr = 44100
kr = 100
ksmps = 1

gkc init 1000 ; set default value
gkc chnexport "cutoff", 1, 3, i(gkc), 500, 2000

instr 1
  a1 oscil p4, p5, 100
  a2 lowpass2 a1, gkc, 200
  out a2
endin
```

## Crédits

Auteur : Istvan Varga  
2005

# chnget

chnget — Lit des données depuis le bus logiciel.

## Description

Lit des données depuis un canal du bus logiciel entrant nommé. Cela implique la déclaration du canal avec *imode=1* (voir aussi *chn\_k*, *chn\_a* et *chn\_S*).

## Syntaxe

```
ival chnget Sname  
kval chnget Sname  
aval chnget Sname  
Sval chnget Sname  
Sval chngetks Sname  
ival[] chngeti Sname[]  
kval[] chngetk Sname[]  
aval[] chngeta Sname[]  
Sval[] chngets Sname[]
```

## Initialisation

*Sname* -- une chaîne de caractères identifiant le canal du bus logiciel nommé à lire.

*Sname[]* -- un tableau de canaux du bus logiciel nommés à interroger.

*ival* -- la valeur de contrôle lue à l'initialisation.

*Sval* -- la chaîne de caractères lue à l'initialisation.

*ival[]* -- un tableau de valeurs de contrôle lues à l'initialisation.

*Sval[]* -- un tableau de chaînes de caractères lues à l'initialisation.

## Exécution

*kval* -- la valeur de contrôle lue pendant l'exécution.

*kval[]* -- un tableau de valeurs de contrôle lues pendant l'exécution.

*aval* -- le signal audio lu pendant l'exécution.

*aval[]* -- le tableau de vecteurs audio lus pendant l'exécution.

*Sval* -- la chaîne de caractères lue au taux-k. L'opcode *chnget* fonctionne au temps-i et durant l'exécution, tandis que *chngetks* ne fonctionne que pendant l'exécution. Les variables chaîne de caractères ne sont mises à jour que si le canal a changé.



## Note

Bien qu'il soit possible de boucler sur les noms de canaux d'un tableau avec *chnget* et *chnset*, l'utilisation de la variante basée sur des tableaux est plus efficace.

## Exemples

L'exemple montre l'utilisation du bus logiciel comme signal de contrôle asynchrone pour fixer la fréquence de coupure du filtre. On suppose qu'un programme externe utilisant l'API fournit les valeurs.

```
sr = 44100
kr = 100
ksmps = 1

instr 1
  kc  chnget  "cutoff"
  a1  oscil   p4, p5, 100
  a2  lowpass2 a1, kc, 200
      out    a2
endin
```

Voici un autre exemple de l'opcode *chnget*. Il utilise le fichier *chnget.csd* [examples/chnget.csd].

### Exemple 128. Exemple de l'opcode *chnget*.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o chnget.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
;Example by Joachim Heintz
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1; send i-values
  chnset 1, "sio"
  chnset -1, "non"
endin

instr 2; send k-values
kfreq  randomi 100, 300, 1
  chnset kfreq, "cntrfreq"
kbw    = kfreq/10
  chnset kbw, "bandw"
endin

instr 3; send a-values
anois  rand .1
  chnset anois, "noise"

loop:
idur   random .3, 1.5
  timeout 0, idur, do
  reinit  loop
do:
```

```

ifreq    random    400, 1200
iamp     random    .1, .3
asig     oscils    iamp, ifreq, 0
aenv     transeg   1, idur, -10, 0
asine    =         asig * aenv
          chnset    asine, "sine"

    endin

    instr 11; receive some chn values and send again
ival1    chnget    "sio"
ival2    chnget    "non"
    print    ival1, ival2
kcntfreq chnget    "cntrfreq"
kbandw   chnget    "bandw"
anoise    chnget    "noise"
afilt     reson    anoise, kcntfreq, kbandw
afilt     balance   afilt, anoise
          chnset    afilt, "filtered"

    endin

    instr 12; mix the two audio signals
amix1     chnget    "sine"
amix2     chnget    "filtered"
          chnmix    amix1, "mix"
          chnmix    amix2, "mix"

    endin

    instr 20; receive and reverb
amix       chnget    "mix"
aL, aR     freeverb  amix, amix, .8, .5
          outs      aL, aR

    endin

    instr 100; clear
          chnclear   "mix"

    endin

</CsInstruments>
<CsScore>
i 1 0 20
i 2 0 20
i 3 0 20
i 11 0 20
i 12 0 20
i 20 0 20
i 100 0 20
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Istvan Varga

2005

Opcodes de canaux basés sur des tableaux ajoutés dans la version 6.14 - Rory Walsh

# chnmix

chnmix — Écrit des données audio vers le bus logiciel nommé, en les mélangeant à la sortie précédente.

## Description

Ajoute un signal audio à un canal du bus logiciel nommé. Cela implique la déclaration du canal avec *imode=2* (voir aussi *chn\_a*).

## Syntaxe

```
chnmix aval, Sname
```

## Initialisation

*Sname* -- une chaîne de caractères indiquant le canal nommé du bus logiciel sur lequel écrire.

## Exécution

*aval* -- le signal audio à écrire pendant l'exécution.

## Exemples

Voici un exemple de l'opcode chnmix. Il utilise le fichier *chnmix.csd* [examples/chnmix.csd].

### Exemple 129. Exemple de l'opcode chnmix.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o chnmix.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
;Example by Joachim Heintz
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1; send i-values
    chnset 1, "sio"
    chnset -1, "non"
endin

instr 2; send k-values
kfreq    randomi 100, 300, 1
    chnset kfreq, "cntrfreq"
kbw      = kfreq/10
    chnset kbw, "bandw"
endin
```



```

    instr 3; send a-values
    anois      rand      .1
               chnset    anois, "noise"

    loop:
    idur       random    .3, 1.5
               timeout    0, idur, do
               reinit     loop

    do:
    ifreq      random    400, 1200
    iamp       random    .1, .3
    asig       oscils    iamp, ifreq, 0
    aenv       transeg   1, idur, -10, 0
    asine      =         asig * aenv
               chnset    asine, "sine"

    endin

    instr 11; receive some chn values and send again
    ival1      chnget    "sio"
    ival2      chnget    "non"
               print     ival1, ival2
    kcntfreq   chnget    "cntrfreq"
    kbandw     chnget    "bandw"
    anoise     chnget    "noise"
    afilt      reson     anoise, kcntfreq, kbandw
    afilt      balance   afilt, anoise
               chnset    afilt, "filtered"

    endin

    instr 12; mix the two audio signals
    amix1      chnget    "sine"
    amix2      chnget    "filtered"
               chnmix    amix1, "mix"
               chnmix    amix2, "mix"

    endin

    instr 20; receive and reverb
    amix       chnget    "mix"
    aL, aR     freeverb  amix, amix, .8, .5
               outs      aL, aR

    endin

    instr 100; clear
               chnclear  "mix"

    endin

</CsInstruments>
<CsScore>
i 1 0 20
i 2 0 20
i 3 0 20
i 11 0 20
i 12 0 20
i 20 0 20
i 100 0 20
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Istvan Varga  
2006

# chnparams

chnparams — Demande les paramètres d'un canal.

## Description

Demande les paramètres d'un canal (s'il n'existe pas, toutes les valeurs retournées sont nulles).

## Syntaxe

```
itype, imode, ictltype, idflt, imin, imax chnparams Sname
```

## Initialisation

*itype* -- type des données du canal (1 : contrôle, 2 : audio, 3 : chaînes de caractères)

*imode* -- somme d'au moins un des nombres suivants : 1 pour entrée et 2 pour sortie.

*ictltype* -- paramètre spécial seulement pour les canaux de contrôle ; s'il n'est pas disponible, il est mis à zéro.

*idflt* -- paramètre spécial seulement pour les canaux de contrôle ; s'il n'est pas disponible, il est mis à zéro.

*imin* -- paramètre spécial seulement pour les canaux de contrôle ; s'il n'est pas disponible, il est mis à zéro.

*imax* -- paramètre spécial seulement pour les canaux de contrôle ; s'il n'est pas disponible, il est mis à zéro.

*Sname* -- chaîne de caractères identifiant le canal.

## Crédits

Auteur : Istvan Varga  
2005

# chnset

chnset — Ecrit des données vers le bus logiciel nommé.

## Description

Ecrit sur un canal du bus logiciel nommé. Cela implique la déclaration du canal avec *imod=2* (voir aussi *chn\_k*, *chn\_a* et *chn\_S*).

## Syntaxe

```
chnset ival, Sname
chnset kval, Sname
chnset aval, Sname
chnset Sval, Sname
chnsetks Sval, Sname
chnseti ival[], []Sname
chnsetk kval[], []Sname
chnseta aval[], []Sname
chnsets Sval[], []Sname
```

## Initialisation

*Sname* -- une chaîne de caractères indiquant le canal nommé du bus logiciel sur lequel écrire.

*Sname[]* -- un tableau de chaînes de caractères indiquant sur quels canaux nommés du bus logiciel écrire.

*ival* -- la valeur de contrôle écrite à l'initialisation.

*ival[]* -- un tableau de valeurs de contrôle à écrire à l'initialisation.

*Sval* -- la chaîne de caractères écrite à l'initialisation.

*Sval[]* -- un tableau de chaînes de caractères à écrire à l'initialisation.

## Exécution

*kval* -- la valeur de contrôle écrite pendant l'exécution.

*aval* -- le signal audio écrit pendant l'exécution.

*Sval* -- la chaîne de caractères à écrire au taux-k. L'opcode *chnset* avec des chaînes de caractères fonctionne au temps-i et durant l'exécution, tandis que *chnsetks* ne fonctionne que pendant l'exécution. Le contenu du canal n'est mis à jour que si la variable chaîne de caractères est modifiée.

*kval[]* -- un tableau de valeurs de contrôle à écrire pendant l'exécution.

*aval[]* -- un tableau de vecteurs audio à écrire pendant l'exécution.



## Note

Bien qu'il soit possible de boucler sur les noms de canaux d'un tableau avec *chnget* et *chnset*, l'utilisation de la variante basée sur des tableaux est plus efficace.

## Exemples

L'exemple montre l'utilisation du bus logiciel pour écrire une information de hauteur vers un programme de contrôle.

```
sr = 44100
kr = 100
ksmps = 1

instr 1
  al    in
  kp,ka pitchamdf al
        chnset    kp, "pitch"
endin
```

Voici un autre exemple de l'opcode *chnset*. Il utilise le fichier *chnset.csd* [examples/chnset.csd].

### Exemple 130. Exemple de l'opcode *chnset*.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o chnset.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
;Example by Joachim Heintz
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1; send i-values
  chnset    1, "sio"
  chnset   -1, "non"
endin

instr 2; send k-values
kfreq      randomi    100, 300, 1
  chnset    kfreq, "cntrfreq"
kbw        =          kfreq/10
  chnset    kbw, "bandw"
endin

instr 3; send a-values
anois      rand       .1
  chnset    anois, "noise"

loop:
idur       random     .3, 1.5
  timeout  0, idur, do
  reinit    loop

do:
ifreq      random     400, 1200
```

```

iamp      random      .1, .3
asig      oscils      iamp, ifreq, 0
aenv      transeg     1, idur, -10, 0
asine     =           asig * aenv
           chnset     asine, "sine"

    endin

    instr 11; receive some chn values and send again
ival1     chnget      "sio"
ival2     chnget      "non"
           print      ival1, ival2
kcntfreq  chnget      "cntrfreq"
kbandw    chnget      "bandw"
anoise    chnget      "noise"
afilt     reson       anoise, kcntfreq, kbandw
afilt     balance     afilt, anoise
           chnset     afilt, "filtered"

    endin

    instr 12; mix the two audio signals
amix1     chnget      "sine"
amix2     chnget      "filtered"
           chnmix     amix1, "mix"
           chnmix     amix2, "mix"

    endin

    instr 20; receive and reverb
amix      chnget      "mix"
aL, aR    freeverb    amix, amix, .8, .5
           outs       aL, aR

    endin

    instr 100; clear
           chnclear   "mix"

    endin

</CsInstruments>
<CsScore>
i 1 0 20
i 2 0 20
i 3 0 20
i 11 0 20
i 12 0 20
i 20 0 20
i 100 0 20
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Istvan Varga

2005

Opcodes de canaux basés sur des tableaux ajoutés dans la version 6.14 - Rory Walsh

# chuap

chuap — Simule un oscillateur de Chua, un oscillateur RLC avec une résistance active, qui peut avoir bifurcation et attracteurs chaotiques, avec un contrôle de taux-k des éléments du circuit.

## Description

Opcode du greffon chua.

Simule un oscillateur de Chua, un oscillateur RLC avec une résistance active, qui peut avoir bifurcation et attracteurs chaotiques, avec un contrôle de taux-k des éléments du circuit.

## Syntaxe

```
aI3, aV2, aV1 chuap kL, kR0, kC1, kG, kGa, kGb, kE, kC2, iI3, iV2, iV1, ktime_step
```

## Initialisation

*iI3* -- Courant initial dans G

*iV2* -- Tension initiale aux bornes de C2

*iV1* -- Tension initiale aux bornes de C1

## Exécution

*kL* -- Inductance L

*kR0* -- Résistance R0

*kC1* -- Capacité C1

*kG* -- Résistance G

*kGa* -- Résistance V (terme non linéaire)

*kGb* -- Résistance V (terme non linéaire)

*kGb* -- Résistance V (terme non linéaire)

*ktime\_step* -- Pas temporel de l'équation aux différences, permet de contrôler plus ou moins la hauteur.

*L'oscillateur de Chua* est un simple oscillateur RLC avec une résistance active. L'oscillateur peut être amené à une bifurcation de période, et ainsi vers le chaos, à cause de la réponse non linéaire de la résistance active.

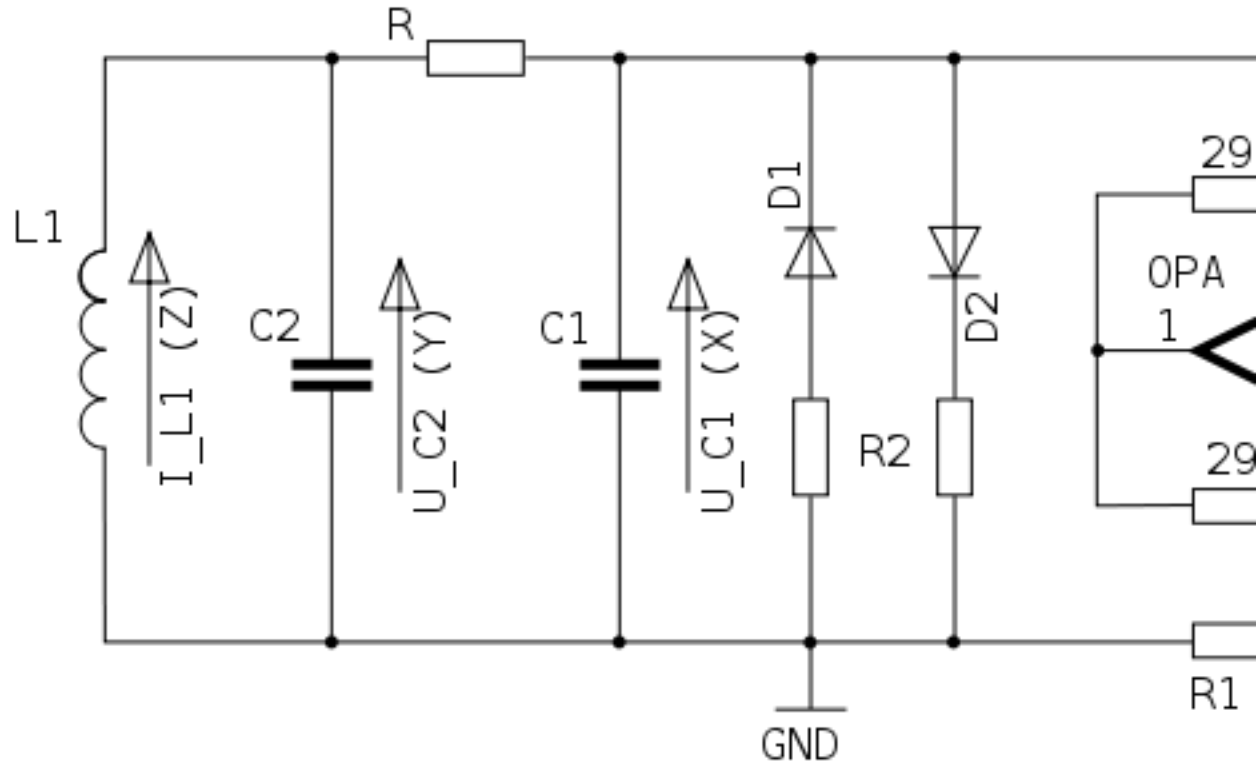


Diagramme du Circuit de l'Oscillateur de Chua

Le circuit est décrit par un ensemble de trois équations différentielles ordinaires appelées équations de Chua :

$$\frac{dI_3}{dt} = -\frac{R_0}{L} I_3 - \frac{1}{L} V_2$$

$$\frac{dV_2}{dt} = -\frac{1}{C_2} I_3 - \frac{G}{C_2} (V_2 - V_1)$$

$$\frac{dV_1}{dt} = \frac{G}{C_1} (V_2 - V_1) - \frac{1}{C_1} f(V_1)$$

où  $f()$  est une fonction dscontinue par morceaux simulant la résistance active :

$$f(V_1) = G_b V_1 + - (G_a - G_b)(|V_1 + E| - |V_1 - E|)$$

Une solution  $(I_3, V_2, V_1)(t)$  de ces équations partant d'un état initial  $(I_3, V_2, V_1)(0)$  est appelée une trajectoire de l'oscillateur de Chua. L'implémentation dans Csound est une simulation de l'oscillateur de Chua par une équation aux différences avec intégration de Runge-Kutta.



## Note

Cet algorithme utilise des boucles de rétroaction internes non linéaires ce qui fait dépendre le résultat audio du taux d'échantillonnage de l'orchestre. Par exemple, si l'on développe un projet avec  $sr=48000\text{Hz}$  et si l'on veut produire un CD audio de ce projet, il faut enregistrer un fichier avec  $sr=48000\text{Hz}$ , puis sous-échantillonner ce fichier à  $44100\text{Hz}$  avec l'utilitaire *src\_conv*.



## Avertissement

Attention ! Certains jeux de paramètres produiront des pics d'amplitude ou une rétroaction positive pouvant endommager vos haut-parleurs.

## Exemples

Voici un exemple de l'opcode chuap. Il utilise le fichier *chuap.csd* [examples/chuap.csd].

### Exemple 131. Exemple de l'opcode chuap.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o chuas_oscillator.wav.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 32
nchnls  = 2
0dbfs   = 1

gibuzztable ftgen 1, 0, 16384, 10, 1

instr 1
; sys_variables = system_vars(5:12); % L,R0,C2,G,Ga,Gb,E,C1 or p8:p15
; integ_variables = [system_vars(14:16),system_vars(1:2)]; % x0,y0,z0,dataset_size,step_size or p17:p19
istep_size = p5
iL          = p8
iR0         = p9
iC2         = p10
iG          = p11
iGa         = p12
iGb         = p13
iE          = p14
iC1         = p15
iI3         = p17
iV2         = p18
iV1         = p19
ignore      = p21
ignore      = p22
ignore      = p23
ignore      = p24
ignore      = p25
iattack     = 0.02
isustain    = p3
irelease    = 0.02
p3          = iattack + isustain + irelease
iscale      = 1.0
```



```

adamping      linseg  0.0, iattack, iscale, isustain, iscale, irelease, 0.0
aguide        buzz   0.5, 440, sr/440, gibuzztable
aI3, aV2, aV1 chuap  iL, iR0, iC2, iG, iGa, iGb, iE, iC1, iI3, iV2, iV1, istep_size
asignal       balance aV2, aguide
              outs   adamping * asignal, adamping * asignal

endin
</CsInstruments>
<CsScore>
;          Adapted from ABC++ MATLAB example data.
i 1 0 20 1500 .1 -1 -1 -0.00707925 0.00001647 100 1 -.99955324 -1.00028375 1 -.00222159 204.8 -2.362
i 1 + 20 1500 .425 0 -1 1.3506168 0 -4.50746268737 -1 2.4924 .93 1 1 0 -22.28662665 .0
i 1 + 20 1024 .05 -1 -1 0.00667 0.000651 10 -1 .856 1.1 1 .06 51.2 -20.200590133667 .172539323
i 1 + 20 1024 0.05 -1 -1 0.00667 0.000651 10 -1 0.856 1.1 1 0.1 153.6 21.12496758 0.03001749 0.5158286
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Inventeur de l'oscillateur de Chua : *Leon O. Chua* [<http://www.eecs.berkeley.edu/~chua>]

Auteur de la simulation dans MATLAB : James Patrick McEvoy *MATLAB Adventures in Bifurcations and Chaos (ABC++)* [<http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=3541>]

Auteur du portage dans Csound : Michael Gogins

Nouveau dans la version 5.09 de Csound

Note ajoutée par François Pinot, août 2009

# cigoto

cigoto — Transfert conditionnel du contrôle pendant la phase d'initialisation.

## Description

Tranfert conditionnel du contrôle vers l'instruction étiquetée par *label*, lors de la phase d'initialisation seulement.

## Syntaxe

```
cigoto condition, label
```

où *label* se trouve dans le même bloc d'instrument et n'est pas une expression, et où *condition* utilise un des opérateurs relationnels (<, =, <=, ==, !=) (et = par commodité, voir aussi *Valeurs Conditionnelles*).

## Exemples

Voici un exemple de l'opcode cigoto. Il utilise le fichier *cigoto.csd* [examples/cigoto.csd].

### Exemple 132. Exemple de l'opcode cigoto.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Get the value of the 4th p-field from the score.
iparam = p4

; If iparam is 1 then play the high note.
; If not then play the low note.
cigoto (iparam ==1), highnote
      igoto lownote

highnote:
  ifreq = 880
  goto playit

lownote:
  ifreq = 440
```

```

    goto playit

playit:
    ; Print the values of iparam and ifreq.
    print iparam
    print ifreq

    a1 oscil 10000, ifreq, 1
    out a1
endin

</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; p4: 1 = high note, anything else = low note
; Play Instrument #1 for one second, a low note.
i 1 0 1 0
; Play a Instrument #1 for one second, a high note.
i 1 1 1 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

instr 1:  iparam = 0.000
instr 1:  ifreq = 440.000
instr 1:  iparam = 1.000
instr 1:  ifreq = 880.000

```

## Voir aussi

*cggoto, ckgoto, cngoto, goto, if, kgoto, rigoto, tigo, timeout*

## Crédits

Exemple écrit par Kevin Conder.

# ckgoto

ckgoto — Transfert conditionnel du contrôle lors des phases d'exécution.

## Description

Tranfert conditionnel du contrôle vers l'instruction étiquetée par *label*, lors des phases d'exécution seulement.

## Syntaxe

```
ckgoto condition, label
```

où *label* se trouve dans le même bloc d'instrument et n'est pas une expression, et où *condition* utilise un des opérateurs relationnels (<, =, <=, ==, !=) (et = par commodité, voir aussi *Valeurs Conditionnelles*).

## Exemples

Voici un exemple de l'opcode ckgoto. Il utilise le fichier *ckgoto.csd* [examples/ckgoto.csd].

### Exemple 133. Exemple de l'opcode ckgoto.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ckgoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Change kval linearly from 0 to 2 over
; the period set by the third p-field.
kval line 0, p3, 2

; If kval is greater than or equal to 1 then play the high note.
; If not then play the low note.
ckgoto (kval >= 1), highnote
      kgoto lownote

highnote:
      kfreq = 880
      goto playit

lownote:
```

```
kfreq = 440
goto playit

playit:
; Print the values of kval and kfreq.
printks "kval = %f, kfreq = %f\\n", 1, kval, kfreq

a1 oscil 10000, kfreq, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table: a simple sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
kval = 0.000000, kfreq = 440.000000
kval = 0.999732, kfreq = 440.000000
kval = 1.999639, kfreq = 880.000000
```

## Voir aussi

*cggoto, cigoto, cngoto, goto, if, igoto, tigoto, timeout*

## Crédits

Exemple écrit par Kevin Conder.

# clear

clear — Met à zéro une liste de signaux audio.

## Description

*clear* met à zéro une liste de signaux audio.

## Syntaxe

```
clear avar1 [, avar2] [, avar3] [...]
```

## Exécution

*avar1*, *avar2*, *avar3*, ... -- signaux à mettre à zéro.



### Note

Avant la version 6.13 cet opcode n'était pas compatible avec les opérations multi-coeurs s'il était utilisé avec une variable globale.

*clear* met à zéro chaque échantillon de chacun des signaux audio donnés lors de son exécution. C'est comme si l'on écrivait  $avarN = 0$  dans l'orchestre pour chaque variable spécifiée. Typiquement, *clear* est utilisé avec des variables globales qui combinent plusieurs signaux venant de sources différentes et qui sont changées à chaque passe au taux-k (boucle d'exécution) par toutes les instances d'instrument actives. Après la dernière utilisation d'une de ces variables et avant la passe de taux-k suivante, il faut l'effacer afin qu'elle n'ajoute pas les signaux du cycle suivant au précédent résultat. *clear* est particulièrement utile en combinaison avec *vincr* (incrément de variable) lors de leur utilisation conjointe avec des opcodes de sortie dans un fichier comme *fout*.

## Exemples

Voici un exemple de l'opcode *clear*. Il utilise le fichier *clear.csd* [examples/clear.csd].

### Exemple 134. Exemple de l'opcode *clear*.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o clear.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gaReverb init 0

instr 1
```

```

idur = p3
kpitch = p4
a1 diskin2 "fox.wav", kpitch
a1 = a1*.5 ;reduce volume
vincr gaReverb, a1
endin

instr 99 ; global reverb
a1, ar reverbsc gaReverb, gaReverb, .8, 10000
outs gaReverb+a1, gaReverb+ar

clear gaReverb

endin

</CsInstruments>
<CsScore>

i1 0 3 1
i99 0 5
e

</CsScore>
</CsoundSynthesizer>

```

Voir l'opcode *fout* pour autre exemple.

## Voir aussi

*vincr*

## Crédits

Auteur : Gabriel Maldonado  
 Italie  
 1999

Nouveau dans la version 3.56 de Csound

# clfilt

clfilt — Implémente des filtres passe-bas et passe-haut de différents styles.

## Description

Implémente les types de filtres analogiques standard classiques : passe-bas et passe-haut. Ils sont implémentés avec les quatre sortes de filtre classiques : Butterworth, Tchebychev de type 1, Tchebychev de type 2 et elliptique. Le nombre de pôles peut être n'importe quel nombre pair compris entre 2 et 80.

## Syntaxe

```
ares clfilt asig, kfreq, itype, inpol [, ikind] [, ipbr] [, isba] [, iskip]
```

## Initialisation

*itype* -- 0 pour passe-bas, 1 pour passe-haut.

*inpol* -- Le nombre de pôles du filtre. Ce doit être un nombre pair compris entre 2 et 80.

*ikind* (facultatif) -- 0 pour Butterworth, 1 pour Tchebychev de type 1, 2 pour Tchebychev de type 2, 3 pour elliptique. 0 par défaut (Butterworth).

*ipbr* (facultatif) -- Ondulation de la bande passante en dB. Doit être supérieure à 0. Elle est ignorée dans les filtres de Butterworth et de Tchebychev de type 2. La valeur par défaut est 1 dB.

*isba* (facultatif) -- Atténuation de la bande bloquée en dB. Doit être inférieure à 0. Elle est ignorée dans les filtres de Butterworth et de Tchebychev de type 1. La valeur par défaut est -60 dB.

*iskip* (facultatif) -- 0 initialise tous les états internes du filtre à 0. 1 ignore l'initialisation. La valeur par défaut est 0.

## Exécution

*asig* -- Le signal audio en entrée.

*kfreq* -- La fréquence de coupure du filtre passe-bas ou passe-haut.

## Exemples

Voici un exemple de l'opcode clfilt comme filtre passe-bas. Il utilise le fichier *clfilt\_lowpass.csd* [examples/clfilt\_lowpass.csd].

### Exemple 135. Exemple de l'opcode clfilt comme filtre passe-bas.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform
```



```

-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o clfilt_lowpass.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; white noise

asig rand 0.5
outs asig, asig

endin

instr 2 ; filtered noise

asig rand 0.9
; Lowpass filter signal asig with a
; 10-pole Butterworth at 500 Hz.
a1 clfilt asig, 500, 0, 10
outs a1, a1

endin

</CsInstruments>
<CsScore>

i 1 0 2
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>

```

Voici un exemple de l'opcode `clfilt` comme filtre passe-haut. Il utilise le fichier `clfilt_highpass.csd` [examples/clfilt\_highpass.csd].

### Exemple 136. Exemple de l'opcode `clfilt` comme filtre passe-haut.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o clfilt_highpass.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; white noise

asig rand 0.6

```

```

        outs asig, asig

    endin

    instr 2 ;filtered noise

    asig rand 0.7
    ; Highpass filter signal asig with a 6-pole Chebyshev
    ; Type I at 20 Hz with 3 dB of passband ripple.
    a1 clfilt asig, 20, 1, 6, 1, 3
        outs a1, a1

    endin

</CsInstruments>
<CsScore>

i 1 0 2
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Erik Spjut

Nouveau dans la version 4.20

# clip

clip — Rogne un signal à une limite prédéfinie.

## Description

Coupe un signal de taux-a à une limite prédéfinie, de manière « douce », en utilisant une méthode choisie parmi les trois possibles.

## Syntaxe

```
ares clip asig, imeth, ilimit [, iarg]
```

## Initialisation

*imeth* -- choisit la méthode de coupure. La valeur par défaut est 0. Les méthodes sont :

- 0 = méthode de Bram de Jong (par défaut)
- 1 = coupure par sinus
- 2 = coupure par tanh

*ilimit* -- valeur limite

*iarg* (facultatif, 0.5 par défaut) -- lorsque *imeth* = 0, indique le point, compris entre 0 et 1, où la coupure commence. N'est pas utilisé si *imeth* = 1 ou 2. Sa valeur par défaut est 0.5.

## Exécution

*asig* -- signal de taux-a en entrée

La méthode de Bram de Jong (*imeth* = 0) applique l'algorithme (en notant *ilimit* comme *limit* et *iarg* comme *a* :

```
|x| >= 0 and |x| <= (limit*a): f(x) = f(x)
|x| > (limit*a) and |x| <= limit: f(x) = sign(x) * (limit*a+(x-limit*a)/(1+((x-limit*a)/(limit*(1-a))))
|x| > limit: f(x) = sign(x) * (limit*(1+a))/2
```

La seconde méthode (*imeth* = 1) est la coupure par sinus :

```
|x| < limit: f(x) = limit * sin(#*x/(2*limit)), |x| >= limit: f(x) = limit * sign(x)
```

La troisième méthode (*imeth* = 2) est la coupure par tanh :

```
|x| < limit: f(x) = limit * tanh(x/limit)/tanh(1), |x| >= limit: f(x) = limit * sign(x)
```



### Note

Il semble que la méthode 1 n'était pas fonctionnelle dans la version 4.07 de Csound.

## Exemples

Voici un exemple de l'opcode clip. Il utilise le fichier *clip.csd* [examples/clip.csd].

### Exemple 137. Exemple de l'opcode clip.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime output leave only the line below:
; -o clip.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; white noise

arnd rand 1 ; full amlitude
; Clip the noisy waveform's amplitude to 0.5
a1 clip arnd, p4, 0.5
outs a1, a1

endin

instr 2 ; white noise

arnd rand 1 ; full amlitude
; Clip the noisy waveform's amplitude to 0.1
a1 clip arnd, p4, 0.1
outs a1, a1

endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1 2
; Play Instrument #2 for one second.
i 2 1 1 2
s 3
; Play Instrument #1 for one second.
i 1 0 1 0
; Play Instrument #2 for one second.
i 2 1 1 0
s 3
; Play Instrument #1 for one second.
i 1 0 1 1
; Play Instrument #2 for one second.
i 2 1 1 1
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch  
Université de Bath, Codemist Ltd.

Bath, UK  
Août, 2000

Nouveau dans la version 4.07 de Csound

Septembre 2009 : grâce à une note de Paolo Dell'Oso, les formules ont été corrigées.

# clockoff

clockoff — Arrête l'une des horloges internes.

## Description

Arrête l'une des horloges internes.

## Syntaxe

```
clockoff inum
```

## Initialisation

*inum* -- le numéro d'une horloge. Il y a 32 horloges numérotées de 0 à 31. Toutes les autres valeurs correspondent à l'horloge numéro 32.

## Exécution

Entre deux opcodes *clockon* et *clockoff*, le temps CPU utilisé est accumulé dans l'horloge. La précision dépend de la machine et elle est de l'ordre de la milliseconde sur les systèmes UNIX et Windows. L'opcode *readclock* lit la valeur courante d'une horloge pendant une phase d'initialisation.

## Exemples

Voici un exemple de l'opcode clockoff. Il utilise le fichier *clockoff.csd* [examples/clockoff.csd].

### Exemple 138. Exemple de l'opcode clockoff.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o clockoff.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
instr 1
; Start clock #1.
clockon 1
; Do something that keeps Csound busy.
a1 oscili 10000, 440, 1
out a1
; Stop clock #1.
clockoff 1
```

```

; Print the time accumulated in clock #1.
i1 readclock 1
print i1
endin

</CsInstruments>
<CsScore>

; Initialize the function tables.
; Table 1: an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for one second starting at 0:00.
i 1 0 1
; Play Instrument #1 for one second starting at 0:01.
i 1 1 1
; Play Instrument #1 for one second starting at 0:02.
i 1 2 1
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*clockon, readclock*

## Crédits

Auteur : John ffitch  
 Université de Bath/Codemist Ltd.  
 Bath, UK  
 Juillet 1999

Nouveau dans la version 3.56 de Csound

# clockon

clockon — Démarre l'une des horloges internes.

## Description

Démarre l'une des horloges internes.

## Syntaxe

```
clockon inum
```

## Initialisation

*inum* -- le numéro d'une horloge. Il y a 32 horloges numérotées de 0 à 31. Toutes les autres valeurs correspondent à l'horloge numéro 32.

## Exécution

Entre deux opcodes *clockon* et *clockoff*, le temps CPU utilisé est accumulé dans l'horloge. La précision dépend de la machine et elle est de l'ordre de la milliseconde sur les systèmes UNIX et Windows. L'opcode *readclock* lit la valeur courante d'une horloge pendant une phase d'initialisation.

## Exemples

Voici un exemple de l'opcode *clockon*. Il utilise le fichier *clockon.csd* [examples/clockon.csd].

### Exemple 139. Exemple de l'opcode clockon.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o clockon.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
instr 1
; Start clock #1.
clockon 1
; Do something that keeps Csound busy.
a1 oscili 10000, 440, 1
out a1
; Stop clock #1.
clockoff 1
```



```

; Print the time accumulated in clock #1.
i1 readclock 1
print i1
endin

</CsInstruments>
<CsScore>

; Initialize the function tables.
; Table 1: an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for one second starting at 0:00.
i 1 0 1
; Play Instrument #1 for one second starting at 0:01.
i 1 1 1
; Play Instrument #1 for one second starting at 0:02.
i 1 2 1
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*clockoff, readclock*

## Crédits

Auteur : John ffitch  
 Université de Bath/Codemist Ltd.  
 Bath, UK  
 Juillet 1999

Nouveau dans la version 3.56 de Csound

# cmp

cmp — Compare des signaux audio ou des tableaux.

## Description

Opcodes du greffon emugens.

Compare des signaux audio ou des tableaux, échantillon par échantillon ou valeur par valeur. Toutes les comparaisons sont possibles : `<`, `<=`, `>`, `>=`, `==`, `!=`.

Un signal audio peut être comparé à un autre signal audio, ou à un scalaire (valeur de taux-i ou -k).

```
aout cmp aL, ">", aR
aOut cmp aIn, ">=", 0.5
aOut cmp aIn, "`<=", kthreshold
```

Un tableau peut être comparé à un autre tableau, à un scalaire, ou testé s'il appartient à un intervalle entre deux scalaires. Toutes ces opérations sont valables pour les tableaux-i et -k.

```
kOut[] cmp kIn[], ">=", kx
kOut[] cmp kA[], "==", kB[]
kOut[] cmp 0.5, "`<", kIn[], "`<=", 1
```

## Syntaxe

```
aout cmp a1, S_operator, a2
aout cmp a1, S_operator, kx
kOut[] cmp kA, S_operator, kB
kOut[] cmp k1, S_operator1, kIn[], S_operator2, k2
```

## Initialisation

*S\_operator* -- un opérateur mathématique parmi `>`, `>=`, `<`, `<=`, `==`

## Exécution

*a1* -- Signal à gauche de l'opérateur.

*a2* -- Signal à droite de l'opérateur.

```
aout cmp aL, ">", aR      ; aout = aL > aR pour chaque échantillon
aout cmp aL, ">=", aR
aout cmp aL, "`<", aR
aout cmp aL, "`<=", aR
aout cmp aL, "==", aR
```

## Exemples

Voici un exemple de l'opcode `cmp`. Il utilise le fichier `cmp.csd` [examples/cmp.csd].

### Exemple 140. Exemple de l'opcode `cmp`.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 5
0dbfs = 1.0

; this is the example file for 'cmp'

/*

cmp

compare audio or arrays, value by value

Audio:
* compare audio signals, sample by sample, against another signal or scalar
* compara audio singal, sample by sample, within a range:
  aout cmp klo, "<", ain, "<=", khi

Arrays:
* compare arrays value by value or against a scalar
* compare array value by value within a range:
  kout[] = klo < kin[] <= khi  ->  kout[] cmp klo, "<", kin[], "<=", khi

aout cmp a1, Sop, a2          : aout cmp ain, "<", acmp
aout cmp a1, Sop, kval        : aout cmp ain, ">=", 0.1
kout[] cmp k1[], Sop, k2[]    : kout[] cmp kxs, "<", kys
iout[] cmp i1[], Sop, i2[]    : iout[] cmp ixs, "<", iys
kout[] cmp k1[], Sop, k       : kout[] cmp kxs, "<", 0.5
iout[] cmp i1[], Sop, i       : iout[] cmp ixs, "<", 0.5
kout[] cmp klo, Sop, kx[], Sop, khi : kout[] cmp 0, "<", kxs, "<=", 1
iout[] cmp ilo, Sop, ix[], Sop, ihi : iout[] cmp 0, "<", ixs, "<=", 1

TODO: implement array operations for multidim. arrays
      (at the time, array operations work only for 1D-arrays)

*/

; for audio operations, render this to a soundfile and open in an editor
; to check the results

instr 1
  a0 linseg 0, p3, 1
  a1 linseg 1, p3, 0
  aout1 cmp a0, "<", a1
  aout2 cmp a0, "<=", 0.5
  aout3 cmp a0, ">", 0.5
  outch 1, a0
  outch 2, a1
  outch 3, aout1
  outch 4, aout2
  outch 5, aout3
endin
```

```

instr 4
; cmp with arrays
ixs[] fillarray 0, 1, 2, 3, 4, 5
iys[] cmp ixs, ">=", 3
printarray iys, "", "instr 4, iys"

kxs[] fillarray 0, 1, 2, 3, 4, 5
kys[] cmp kxs, ">=", 3
printarray kys, 1, "", "instr 4, kys"
turnoff
endin

instr 5
; range
ixs[] fillarray 0, 1, 2, 3, 4, 5
iys[] cmp 1, "<", ixs, "<=", 4
printarray iys, "", "instr 5, iys"

kxs[] fillarray 0, 1, 2, 3, 4, 5
kys[] cmp 1, "<", kxs, "<=", 4
printarray kys, 1, "", "instr 5, kys"
turnoff
endin

</CsInstruments>
<CsScore>
i 1 0 2
i 4 0 1
i 5 0 1
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*max, min,*

## Crédits

Par : Eduardo Moguillansky 2017

# cmplxprod

cmplxprod — Multiplication complexe de deux tableaux.

## Description

Calcule le produit complexe de deux tableaux de la même taille et en format réel-imaginaire entrelacé.

## Syntaxe

```
kout[] cmplxprod kin1[], kin2[]
```

## Exécution

*kout[]* -- tableau de sortie contenant le produit. Créé s'il n'existe pas.

*kin1[],kin2[]* -- tableaux contenant les entrées complexes.

## Exemples

Voici un exemple de l'opcode cmplxprod. Il utilise le fichier *cmplxprod.csd* [examples/cmplxprod.csd].

### Exemple 141. Exemple de l'opcode cmplxprod.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>

<CsOptions>
-d -o dac
</CsOptions>

<CsInstruments>
;ksmps needs to be an integer div of hopsize
ksmps = 64
Odbfs = 1

instr 1

  ihopsize = 256 ; hopsize
  ifftsize = 1024 ; FFT size
  iolaps = ifftsize/ihopsize ; overlaps
  ibw = sr/ifftsize ; bin bandwidth
  kent init 0 ; counting vars
  krow init 0

  kOla[] init ifftsize ; overlap-add buffer
  kIn[] init ifftsize ; input buffer
  kFil[] init ifftsize ; filter buffer
  kOut[][] init iolaps, ifftsize ; output buffers

  kfirst init 1
  if kfirst == 1 then
    copyf2array kFil,1
```

```

    kfirst = 0
endif

a1 diskin2 "fox.wav",1,0,1 ; audio input

/* every hopsize samples */
if kcnt == ihopsiz then
    /* window and take FFT */
    kWin[] window kIn,krow*ihopsiz
    kSpec[] rfft kWin

    kProd[] cmplxprod kSpec, kFil

    /* IFFT + window */
    kRow[] rfft kProd
    kWin window kRow, krow*ihopsiz
    /* place it on out buffer */
    kOut setrow kWin, krow

    /* zero the ola buffer */
    kOla = 0
    /* overlap-add */
    ki = 0
    until ki == iolaps do
        kRow getrow kOut, ki
        kOla = kOla + kRow
        ki += 1
    od

    /* update counters */
    krow = (krow+1)%iolaps
    kcnt = 0
endif

/* shift audio in/out of buffers */
kIn shiftin a1
a2 shiftout kOla
    out a2/iolaps

/* increment counter */
kcnt += ksmps

endin

</CsInstruments>

<CsScore>
f1 0 1024 7  0 64 0.1 64 0.2 128 0.5 256 1 512 1
i1 0 10
</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux.

## Crédits

Auteur : Victor Lazzarini  
 NUI Maynooth  
 2014

Nouveau dans la version 6.04

# cngoto

cngoto — Transfère le contrôle à chaque passage si la condition n'est pas vraie.

## Description

Transfère le contrôle à chaque passage si la condition n'est *pas* vraie.

## Syntaxe

**cngoto** condition, label

où *label* se trouve dans le même bloc d'instrument et n'est pas une expression, et où *condition* utilise un des opérateurs relationnels (<, =, <=, ==, !=) (et = par commodité, voir aussi *Valeurs Conditionnelles*).

## Exemples

Voici un exemple de l'opcode cngoto. Il utilise le fichier *cngoto.csd* [examples/cngoto.csd].

### Exemple 142. Exemple de l'opcode cngoto.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; -o cngoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Change kval linearly from 0 to 2 over
; the period set by the third p-field.
kval line 0, p3, 2

; If kval *is not* greater than or equal to 1 then play
; the high note. Otherwise, play the low note.
cngoto (kval >= 1), highnote
      kgoto lownote

highnote:
      kfreq = 880
      goto playit

lownote:
      kfreq = 440
      goto playit

playit:
```

```

; Print the values of kval and kfreq.
printks "kval = %f, kfreq = %f\\n", 1, kval, kfreq

a1 oscil 10000, kfreq, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table: a simple sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

kval = 0.000000, kfreq = 880.000000
kval = 0.999732, kfreq = 880.000000
kval = 1.999639, kfreq = 440.000000

```

## Voir aussi

*cggoto, cigoto, ckgoto, goto, if, igoto, tigoto, timeout*

## Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.21



# comb

comb — Réverbère un signal d'entrée avec une réponse en fréquence « colorée ».

## Description

Réverbère un signal d'entrée avec une réponse en fréquence « colorée ».

## Syntaxe

```
ares comb asig, krvt, ilpt [, iskip] [, insmps]
```

## Initialisation

*ilpt* -- durée de boucle en secondes, déterminant la « densité d'échos » de la réverbération. Celle-ci caractérise à son tour la « couleur » du filtre *en peigne* dont la courbe de réponse en fréquence contiendra *ilpt* \* *sr/2* pics régulièrement espacés entre 0 et *sr/2* (la fréquence de Nyquist). La durée de boucle peut être aussi grande que le permet la mémoire disponible. L'espace requis pour une boucle de *n* secondes est de *n\*sr* nombres de type *float* ou *double* (habituellement 4 ou 8 octets). L'espace pour le retard est alloué et retourné comme dans *delay*.

*iskip* (facultatif, 0 par défaut) -- état initial de l'espace de données de la boucle de retard (cf. *reson*). La valeur par défaut est 0.

*insmps* (facultatif, 0 par défaut) -- valeur du retard, en nombre d'échantillons.

## Exécution

*krvt* -- la durée de réverbération (définie comme le temps en secondes pris par un signal pour décroître à 1/1000 ou 60 dB de son amplitude originale).

Ce filtre répète l'entrée avec une densité d'écho déterminée par la durée de boucle *ilpt*. Le taux d'atténuation est indépendant et il est déterminé par *krvt*, la durée de réverbération (définie comme le temps en secondes pris par un signal pour décroître à 1/1000 ou 60 dB de son amplitude originale). La sortie d'un filtre en peigne n'apparaît qu'après *ilpt* secondes.

## Exemples

Voici un exemple de l'opdoce comb. Il utilise le fichier *comb.csd* [examples/comb.csd].

### Exemple 143. Exemple de l'opdoce comb.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
```

```

; -o comb.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gamix init 0

instr 1

kcps    expon p5, p3, p4
asig vco2 0.3, kcps
outs asig, asig

gamix = gamix + asig

endin

instr 99

krvt = 3.5
ilpt = 0.1
aleft comb gamix, krvt, ilpt
aright comb gamix, krvt, ilpt*.2
outs aleft, aright

clear gamix ; clear mixer

endin

</CsInstruments>
<CsScore>

i 1 0 3 20 2000
i 1 5 .01 440 440

i 99 0 8
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*alpass, reverb, valpass, vcomb*

## Crédits

Barry L. Vercoe

# combinv

combinv — Réverbère un signal d'entrée avec une réponse en fréquence « colorée ».

## Description

Réverbère un signal d'entrée avec une réponse en fréquence « colorée » avec un filtre à RIF.

## Syntaxe

```
ares combinv asig, krvt, ilpt [, iskip] [, insmps]
```

## Initialisation

*ilpt* -- durée de boucle en secondes, déterminant la « densité d'échos » de la réverbération. Celle-ci caractérise à son tour la « couleur » du filtre *combinv* dont la courbe de réponse en fréquence contiendra *ilpt* \* *sr*/2 pics régulièrement espacés entre 0 et *sr*/2 (la fréquence de Nyquist). La durée de boucle peut être aussi grande que le permet la mémoire disponible. L'espace requis pour une boucle de *n* secondes est de *n*\**sr* nombres de type *float* ou *double* (habituellement 4 ou 8 octets). L'espace pour le retard est alloué et retourné comme dans *delay*.

*iskip* (facultatif, 0 par défaut) -- état initial de l'espace de données de la boucle de retard (cf. *reson*). La valeur par défaut est 0.

*insmps* (facultatif, 0 par défaut) -- valeur du retard, en nombre d'échantillons.

## Exécution

*krvt* -- la durée de réverbération (définie comme le temps en secondes pris par un signal pour décroître à 1/1000 ou 60 dB de son amplitude originale).

Ce filtre répète l'entrée avec une densité d'écho déterminée par la durée de boucle *ilpt*. Le taux d'atténuation est indépendant et il est déterminé par *krvt*, la durée de réverbération (définie comme le temps en secondes pris par un signal pour décroître à 1/1000 ou 60 dB de son amplitude originale). La sortie d'un filtre *combinv* n'apparaît qu'après *ilpt* secondes.

## Exemples

Voici un exemple de l'opdoce *combinv*. Il utilise le fichier *combinv.csd* [exemples/combinv.csd].

### Exemple 144. Exemple de l'opdoce *combinv*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
```

```

; -o comb.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kcps    expon p5, p3, p4
asig    oscil3 0.3, kcps
krvt =  3.5
ilpt =  0.1
aleft    combinv asig, krvt, ilpt
outs    aleft, asig

endin

</CsInstruments>
<CsScore>
i 1 0 3 20 2000
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*alpass, comb reverb, valpass, vcomb*

## Crédits

Auteur : John ffitich  
 Octobre 2012

# compress

`compress` — Compresse, limite, dilate, atténue ou impose un seuil à un signal audio.

## Description

Cette unité fonctionne comme un compresseur audio, un limiteur, un expander ou un noise gate, avec un coude arrondi ou non et des caractéristiques d'exécution variant dynamiquement. Il prend deux signaux audio en entrée, *aasig* et *acsig*, le premier étant modifié par l'analyse courante du second. Les deux signaux peuvent être le même signal, ou le premier peut être modifié par un signal de contrôle différent.

**compress** examine d'abord le signal de contrôle *acsig* en faisant une détection d'enveloppe. Celle-ci est déterminée par deux valeurs de contrôle *katt* et *krel*, définissant les constantes d'attaque et de relachement (en secondes) du détecteur. Le détecteur suit les crêtes (pas la valeur efficace) du signal de contrôle. Les valeurs typiques sont 0.01 et 0.1, la dernière étant habituellement du même ordre que *ilook*.

L'enveloppe courante est alors convertie en décibels puis passe par une fonction de sélection pour déterminer quelle action du compresseur (s'il y en a une) doit être appliquée. La fonction de sélection est définie par quatre valeurs de contrôle en décibels. Elles sont données sous forme de valeurs positives, où 0 dB correspond à une amplitude de 0dbfs/32768, et 90 dB correspond à une amplitude de 0dbfs.

## Syntaxe

```
ar compress aasig, acsig, kthresh, kloknee, khiknee, kratio, katt, krel, ilook
```

## Initialisation

*ilook* -- temps de prospection en secondes, pendant lequel un déclenchement d'enveloppe interne peut détecter ce qui se passe. Cela induit un délai entre l'entrée et la sortie, mais une petite durée de prospection améliore les performances du détecteur d'enveloppe. 0.05 secondes est une valeur typique, suffisante pour détecter les crêtes de la fréquence la plus basse dans *acsig*.

## Exécution

*kthresh* -- fixe le niveau le plus bas en décibels qui sera autorisé à traverser le module. Normalement 0 ou moins, mais si le seuil est plus élevé, les signaux de basse énergie tel que le bruit de fond commenceront à être enlevés.

*kloknee*, *khiknee* -- coude de la courbe en décibels indiquant où commencent la compression ou l'expansion. Cela fixe les limites d'un coude arrondi joignant la ligne 1:1 des basses amplitudes et la ligne du rapport de compression des fortes amplitudes. 48 et 60 dB sont des valeurs typiques. Si les deux points sont égaux, le coude est anguleux.

*kratio* -- rapport de compression lorsque le signal est au-delà du coude. La valeur 2 renforce la sortie d'un décibel pour chaque doublement du gain en entrée ; 3 renforce de un pour trois ; 20 de un pour vingt, etc. Les rapports inverses provoquent une expansion du signal : 0.5 donne deux pour un, 0.25 quatre pour un, etc. La valeur 1 ne provoque aucun changement.

Les actions de *compress* dépendent du réglage des paramètres. Un compresseur-limiteur à coude anguleux, par exemple, est obtenu avec une attaque proche de zéro, des limites de coude égales, et un rapport très

élevé (disons 100). Un noise-gate plus expander est obtenu avec un seuil positif et un rapport fractionnaire au-delà du coude. Un compresseur de musique déclenché par la voix (ducker) est obtenu en affectant la musique à *aasig* et la voix à *acsig*. Un de-esser de voix est obtenu en affectant la voix aux deux, avec un filtre passe-bande précédant l'entrée *acsig* pour renforcer les sifflantes. Il est nécessaire d'expérimenter chaque application pour trouver les meilleurs réglages des paramètres ; ceux-ci sont de taux-k pour faciliter cette expérimentation.

## Exemples

Voici un exemple de l'opcode *compress*. Il utilise le fichier *compress.csd* [examples/compress.csd].

### Exemple 145. Exemple de l'opcode *compress*.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -iadc      ;;RT audio out and in
; For Non-realtime ouput leave only the line below:
; -o compress.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1 ; uncompressed signal

asig disk2 "beats.wav", 1, 0, 1
    outs asig, asig
endin

instr 2 ; compressed signal.
; Use the "beats.wav" audio file and a mic
avoice in
asig disk2 "beats.wav", 1, 0, 1

; duck the audio signal "beats.wav" with your voice.
kthresh = 0
kloknee = 40
khiknee = 60
kratio = 3
katt = 0.1
krel = .5
ilook = .02
asig compress asig, avoice, kthresh, kloknee, khiknee, kratio, katt, krel, ilook ; voice-activated com
    outs asig, asig

endin

</CsInstruments>
<CsScore>

i 1 0 5

i 2 6 21

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*dam*

*compress2*

## Crédits

Ecrit par Barry L. Vercoe pour Extended Csound et introduit dans Csound 5.02.

# compileorc

compileorc — Compile un nouvel orchestre à partir d'un fichier ASCII.

## Description

: *compileorc* compile un ou plusieurs instruments pendant l'initialisation, qui seront ajoutés au moteur courant. Si des numéros ou des noms d'instrument existent déjà, ceux-ci seront remplacés, mais toute instance de l'ancienne définition des instruments encore active continuera son exécution jusqu'à son terme.

## Syntaxe

```
ires compileorc Sfilename
```

## Initialisation

« *Sfilename* » -- une chaîne de caractères indiquant le nom du fichier contenant l'orchestre.

« *ires* » -- 0 si la compilation se fait sans erreur, -1 sinon.

## Exemples

Voici un exemple de l'opcode *compileorc*. Il utilise le fichier *compileorc.csd* [exemples/compileorc.csd].

### Exemple 146. Exemple de l'opcode *compileorc*.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-o dac -d
</CsOptions>
<CsInstruments>
sr = 44100
nchnls = 1
ksmps = 32
0dbfs = 1

instr 1
ires compileorc "does_not_exist.orc"
print ires ; -1 as could not compile
ires compileorc "my.orc"
print ires ; 0 as compiled successfully
event_i "i", 2, 0, 3, .2, 465 ;send event
endin

</CsInstruments>
<CsScore>
i1 0 1
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*event*, *event\_i*, *schedule*, *schedwhen*, *schedkwhen*, *schedkwhennamed*



## Crédits

Auteur : Victor Lazzarini, 2013

# compilestr

compilestr — Compile un nouvel orchestre passé dans une chaîne ASCII.

## Description

*compilestr* compile un ou plusieurs instruments pendant l'initialisation, qui seront ajoutés au moteur courant. Si des numéros ou des noms d'instrument existent déjà, ceux-ci seront remplacés, mais toute instance de l'ancienne définition des instruments encore active continuera son exécution jusqu'à son terme. Seules les nouvelles instances utiliseront la nouvelle définition. Les chaînes de caractères sur plusieurs lignes sont acceptées ; elles sont délimitées par `{{ }}`.

## Syntaxe

```
ires compilestr Sorch
```

## Initialisation

« *Sorch* » -- une chaîne de caractères entre guillemets ou délimitée par `{{ }}` et contenant un ou plusieurs instruments.

« *ires* » -- 0 si la compilation se fait sans erreur, -1 sinon.

## Exemples

Voici un exemple de l'opcode *compilestr*. Il utilise le fichier *compilestr.csd* [examples/compilestr.csd].

### Exemple 147. Exemple de l'opcode *compilestr*.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-o dac -d
</CsOptions>
<CsInstruments>
sr = 44100
nchnls = 1
ksmps = 32
0dbfs = 1

instr 1

    ;will fail because of wrong code
    ires compilestr {{
    instr 2
    al oscilb p4, p5, 0
    out al
    endin
    }}
    print ires ; returns -1 because not successfull

    ;will compile ...
    ires compilestr {{
```

```
instr 2
al oscils p4, p5, 0
out al
endin
}}
print ires ; ... and returns 0

;call the new instrument
;(note that the overall performance is extended)
scoreline_i "i 2 0 3 .2 415"

endin

</CsInstruments>
<CsScore>
i1 0 1
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*event, event\_i, schedule, schedwhen, schedkwhen, schedkwhennamed*

## Crédits

Auteur : Victor Lazzarini, 2013

# compress2

`compress2` — Compresse, limite, dilate, atténue ou impose un seuil à un signal audio.

## Description

Cette unité fonctionne comme un compresseur audio, un limiteur, un expander ou un noise gate, avec un coude arrondi ou non et des caractéristiques d'exécution variant dynamiquement. Il prend deux signaux audio en entrée, *aasig* et *acsig*, le premier étant modifié par l'analyse courante du second. Les deux signaux peuvent être le même signal, ou le premier peut être modifié par un signal de contrôle différent.

**compress2** examine d'abord le signal de contrôle *acsig* en faisant une détection d'enveloppe. Celle-ci est déterminée par deux valeurs de contrôle *katt* et *krel*, définissant les constantes d'attaque et de relachement (en secondes) du détecteur. Le détecteur suit les crêtes (pas la valeur efficace) du signal de contrôle. Les valeurs typiques sont 0.01 et 0.1, la dernière étant habituellement du même ordre que *ilook*.

L'enveloppe courante est alors convertie en décibels puis passe par une fonction de sélection pour déterminer quelle action du compresseur (s'il y en a une) doit être appliquée. La fonction de sélection est définie par quatre valeurs de contrôle en décibels. Elles sont données sous forme de valeurs positives, où 0 dB correspond à une amplitude de 0dbfs.

## Syntaxe

```
ar compress2 aasig, acsig, kthresh, kloknee, khiknee, kratio, katt, krel, ilook
```

## Initialisation

*ilook* -- temps de prospection en secondes, pendant lequel un déclenchement d'enveloppe interne peut détecter ce qui se passe. Cela induit un délai entre l'entrée et la sortie, mais une petite durée de prospection améliore les performances du détecteur d'enveloppe. 0.05 secondes est une valeur typique, suffisante pour détecter les crêtes de la fréquence la plus basse dans *acsig*.

## Exécution

*kthresh* -- fixe le niveau le plus bas en décibels qui sera autorisé à traverser le module. Normalement -90 ou moins, mais si le seuil est plus élevé, les signaux de basse énergie tel que le bruit de fond commenceront à être enlevés.

*kloknee*, *khiknee* -- coude de la courbe en décibels indiquant où commencent la compression ou l'expansion. Cela fixe les limites d'un coude arrondi joignant la ligne 1:1 des basses amplitudes et la ligne du rapport de compression des fortes amplitudes. -52 et -30 dB sont des valeurs typiques. Si les deux points sont égaux, le coude est anguleux.

*kratio* -- rapport de compression lorsque le signal est au-delà du coude. La valeur 2 renforce la sortie d'un décibel pour chaque doublement du gain en entrée ; 3 renforce de un pour trois ; 20 de un pour vingt, etc. Les rapports inverses provoquent une expansion du signal : 0.5 donne deux pour un, 0.25 quatre pour un, etc. La valeur 1 ne provoque aucun changement.

Les actions de *compress2* dépendent du réglage des paramètres. Un compresseur-limiteur à coude anguleux, par exemple, est obtenu avec une attaque proche de zéro, des limites de coude égales, et un rapport

très élevé (disons 100). Un noise-gate plus expander est obtenu avec un seuil positif et un rapport fractionnaire au-delà du coude. Un compresseur de musique déclenché par la voix (ducker) est obtenu en affectant la musique à *asig* et la voix à *avoice*. Un de-esser de voix est obtenu en affectant la voix aux deux, avec un filtre passe-bande précédant l'entrée *avoice* pour renforcer les sifflantes. Il est nécessaire d'expérimenter chaque application pour trouver les meilleurs réglages des paramètres ; ceux-ci sont de taux-k pour faciliter cette expérimentation.

## Exemples

Voici un exemple de l'opcode `compress2`. Il utilise le fichier `compress2.csd` [examples/compress2.csd].

### Exemple 148. Exemple de l'opcode `compress2`.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -iadc      ;;RT audio out and in
; For Non-realtime ouput leave only the line below:
; -o compress2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1 ; uncompressed signal

asig diskin2 "beats.wav", 1, 0, 1
outs asig, asig
endin

instr 2 ; compressed signal.
; Use the "beats.wav" audio file and a mic
avoice in
asig diskin2 "beats.wav", 1, 0, 1

; duck the audio signal "beats.wav" with your voice.
kthresh = -90
kloknee = -50
khiknee = -30
kratio = 3
katt = 0.1
krel = .5
ilook = .02
asig compress2 asig, avoice, kthresh, kloknee, khiknee, kratio, katt, krel, ilook ; voice-activated co
outs asig, asig

endin

</CsInstruments>
<CsScore>

i 1 0 5

i 2 6 21

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*dam*

*compress*

## Crédits

Ecrit par John ffitch d'après Barry L. Vercoe pour Extended Csound mais avec des valeurs en dB plus communes ; nouveau dans la version 6.07.

# compilecsd

compilecsd — Compile un nouvel orchestre à partir d'un fichier ASCII.

## Description

*compilecsd* lit un fichier CSD et compile un ou plusieurs instruments durant l'initialisation, qui sont ajoutés à la machine courante. Si des numéros ou des noms d'instrument existent déjà ils sont remplacés, mais chaque instance de l'ancienne définition d'instrument toujours active s'exécute jusqu'à son terme. De plus, la partition contenue dans le fichier CSD (si elle existe) est lue et ajoutée à la liste des événements exécutés par Csound. L'opcode ignore toute section du fichier CSD qui n'est ni un orchestre ni une partition.

## Syntaxe

```
ires compilecsd Sfilename
```

## Initialisation

« *Sfilename* » -- une chaîne de caractères indiquant le nom du fichier contenant l'orchestre.

« *ires* » -- retourne 0 si la compilation a réussi, sinon -1.

## Exemples

Voici un exemple de l'opcode *compilecsd*. Il utilise le fichier *compilecsd.csd* [exemples/compilecsd.csd].

### Exemple 149. Exemple de l'opcode *compilecsd*.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-o dac -d
</CsOptions>
<CsInstruments>
sr = 44100
nchnls = 1
ksmps = 32
0dbfs = 1

instr 1
ires compilecsd "does_not_exist.csd"
print ires ; -1 as could not compile
ires compilecsd "my.csd"
print ires ; 0 as compiled successfully
event_i "i", 2, 0, 3, .2, 465 ;send event
endin

</CsInstruments>
<CsScore>
i1 0 1
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*event, event\_i, schedule, schedwhen, schedkwhen, schedkwhennamed*

## Crédits

Auteur: Victor Lazzarini, 2013



# connect

connect — Connecte un connecteur sortant d'une source à un connecteur entrant d'une destination.

## Description

Opcodes du greffon signalflowgraph.

L'opcode *connect*, valide seulement dans l'en-tête de l'orchestre, envoie les signaux du connecteur sortant indiqué dans toutes les instances de l'instrument source désigné vers le connecteur entrant indiqué dans toutes les instances de l'instrument de destination désigné. Chaque instance du connecteur entrant reçoit la somme des signaux provenant de toutes les instances du connecteur sortant. Ainsi plusieurs instances d'un connecteur sortant peuvent être distribuées sur plusieurs instances d'un connecteur entrant.

Lorsque Csound crée une nouvelle instance d'un modèle d'instrument, de nouvelles instances de ses connexions sont créées.

## Syntaxe

```
connect Tsource1, Soutlet1, Tsink1, Sinlet1
```

## Initialisation

*Tsource1* -- Nom sous forme de chaîne de caractères de la définition de l'instrument source.

*Soutlet1* -- Nom sous forme de chaîne de caractères du connecteur sortant utilisé dans l'instrument source.

*Tsink1* -- Nom sous forme de chaîne de caractères de la définition de l'instrument de destination.

*Sinlet1* -- Nom sous forme de chaîne de caractères du connecteur entrant utilisé dans l'instrument de destination.

## Exemples

Voici un exemple de l'opcode connect. Il utilise le fichier *connect.csd* [examples/connect.csd].

### Exemple 150. Exemple de l'opcode connect.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime output leave only the line below:
; -o connect.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Michael Gogins */
; Initialize the global variables.
sr = 44100
ksmps = 32
nchnls = 2

; Connect up the instruments to create a signal flow graph.
```

```

connect "SimpleSine", "leftout", "Reverberator", "leftin"
connect "SimpleSine", "rightout", "Reverberator", "rightin"

connect "Moogy", "leftout", "Reverberator", "leftin"
connect "Moogy", "rightout", "Reverberator", "rightin"

connect "Reverberator", "leftout", "Compressor", "leftin"
connect "Reverberator", "rightout", "Compressor", "rightin"

connect "Compressor", "leftout", "Soundfile", "leftin"
connect "Compressor", "rightout", "Soundfile", "rightin"

; Turn on the "effect" units in the signal flow graph.

alwayson "Reverberator", 0.91, 12000
alwayson "Compressor"
alwayson "Soundfile"

instr SimpleSine
  ihz = cpsmidinn(p4)
  iamplitude = ampdb(p5)
  print ihz, iamplitude
  ; Use ftgenonce instead of ftgen, ftgentmp, or f statement.
  isine ftgenonce 0, 0, 4096, 10, 1
  a1 oscili iamplitude, ihz, isine
  aenv madsr 0.05, 0.1, 0.5, 0.2
  asignal = a1 * aenv
  ; Stereo audio outlet to be routed in the orchestra header.
  outleta "leftout", asignal * 0.25
  outleta "rightout", asignal * 0.75
endin

instr Moogy
  ihz = cpsmidinn(p4)
  iamplitude = ampdb(p5)
  ; Use ftgenonce instead of ftgen, ftgentmp, or f statement.
  isine ftgenonce 0, 0, 4096, 10, 1
  asignal vco iamplitude, ihz, 1, 0.5, isine
  kfco line 200, p3, 2000
  krez init 0.9
  asignal moogvcf asignal, kfco, krez, 100000
  ; Stereo audio outlet to be routed in the orchestra header.
  outleta "leftout", asignal * 0.75
  outleta "rightout", asignal * 0.25
endin

instr Reverberator
  ; Stereo input.
  aleftin inleta "leftin"
  arightin inleta "rightin"
  idelay = p4
  icutoff = p5
  aleftout, arightout reverbbsc aleftin, arightin, idelay, icutoff
  ; Stereo output.
  outleta "leftout", aleftout
  outleta "rightout", arightout
endin

instr Compressor
  ; Stereo input.
  aleftin inleta "leftin"
  arightin inleta "rightin"
  kthreshold = 25000
  icomp1 = 0.5
  icomp2 = 0.763
  irtime = 0.1
  iftime = 0.1

```

```

aleftout dam aleftin, kthreshold, icompl, icompl2, irtime, iftime
arightout dam arightin, kthreshold, icompl, icompl2, irtime, iftime
; Stereo output.
outleta "leftout", aleftout
outleta "rightout", arightout
endin

instr Soundfile
; Stereo input.
aleftin inleta "leftin"
arightin inleta "rightin"
outs aleftin, arightin
endin

</CsInstruments>
<CsScore>
; Not necessary to activate "effects" or create f-tables in the score!
; Overlapping notes to create new instances of instruments.
i "SimpleSine" 1 5 60 85
i "SimpleSine" 2 5 64 80
i "Moogy" 3 5 67 75
i "Moogy" 4 5 71 70
;6 extra seconds after the performance
e 12
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*outleta outletk outletf inleta inletk inletf alwayson ftgenonce*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/issue13/signalFlowGraphOp-codes.html>, écrit par Michael Gogins

## Crédits

Par Michael Gogins, 2009

# control

control — Contrôleurs réglettes configurables pour une utilisation en temps réel.

## Description

Opcode du greffon control.

Contrôleurs réglettes configurables pour une utilisation en temps réel. Nécessite Winsound ou TCL/TK.  
*control* lit la valeur d'une réglette.

## Syntaxe

```
kres control knum
```

## Exécution

Noter que cet opcode n'est pas disponible sous Windows à cause de l'implémentation des tuyaux sur ce système.

*knum* -- numéro de la réglette à lire.

L'appel de *control* va créer une nouvelle réglette à l'écran. Il n'y a pas de limite théorique au nombre de réglettes. Winsound et TCL/TK n'utilisent que des entiers pour les valeurs de réglette, si bien qu'il peut être nécessaire de re-échelonner les valeurs. Parce que les interfaces graphiques passent habituellement leurs valeurs à une fréquence assez lente, il peut être sage de traiter la sortie du contrôleur avec *port*.

## Exemples

Voir l'opcode *setctrl* pour un exemple.

## Voir aussi

*setctrl*

## Crédits

Auteur : John ffitch  
Université de Bath, Codemist. Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.06 de Csound.

# convle

convle — Identique à l'opcode *convolve*.

## Description

Identique à l'opcode *convolve*.

# convolve

convolve — Convolution d'un signal par une réponse impulsionnelle.

## Description

La sortie est le produit de convolution du signal *ain* par la réponse impulsionnelle contenue dans *ifilcod*. S'il y a plus d'un signal de sortie, chacun sera obtenu par convolution avec la même réponse impulsionnelle. Noter qu'il est considérablement plus efficace d'utiliser une instance de l'opérateur lorsque l'on traite une entrée mono pour créer des sorties stéréo ou quadraphoniques.

Note : cet opcode peut aussi s'écrire *convle*.

## Syntaxe

```
ar1 [, ar2] [, ar3] [, ar4] convolve ain, ifilcod [, ichannel]
```

## Initialisation

*ifilcod* -- entier ou chaîne de caractères définissant un fichier de données contenant une réponse impulsionnelle. Un entier définit le suffixe d'un fichier *convolve.m* ; une chaîne de caractères (entre guillemets) donne un nom de fichier, éventuellement un nom de chemin complet. Si ce n'est pas un nom de chemin complet, le fichier est d'abord cherché dans le répertoire courant, puis dans celui qui est donné par la variable d'environnement SADIR (si elle est définie). Le fichier de données contient la transformée de Fourier d'une réponse impulsionnelle. L'occupation mémoire dépend de la taille du fichier de données qui est lu en entier et gardé en mémoire durant le calcul, mais qui est partagé par des appels multiples.

*ichannel* (facultatif) -- quel canal du fichier de données de la réponse impulsionnelle utiliser.

## Exécution

*ain* -- signal audio en entrée.

*convolve* implémente la convolution rapide. Le sortie de cet opérateur est retardée en fonction de l'entrée. On peut calculer le délai avec les formules suivantes :

```
Pour (1/kr) <= IRdur:
    Delay = ceil(IRdur * kr) / kr
Pour (1/kr) > IRdur:
    Delay = IRdur * ceil(1/(kr*IRdur))
Où :
    kr = taux de contrôle de Csound
    IRdur = durée, en secondes, de la réponse impulsionnelle
    ceil(n) = le plus petit entier qui n'est pas inférieur à n
```

Il faut également faire attention à prendre en compte le délai initial, s'il y en a un, de la réponse impulsionnelle. Par exemple, si une réponse impulsionnelle est créée à partir d'un enregistrement, le fichier son peut ne pas avoir de délai initial. Il faut ainsi soit s'assurer que le fichier son a la quantité correcte de zéro de remplissage au début, soit, de préférence compenser ce retard dans l'orchestre (cette dernière méthode étant plus efficace). Pour compenser le délai dans l'orchestre, il faut soustraire le délai initial du résultat calculé au moyen des formules ci-dessus, lorsque l'on calcule le délai requis à introduire dans la passe audio non "réverbérée".

Pour des applications typiques telles que la réverbération, le délai sera de l'ordre de 0.5 à 1.5 secondes, ou même plus long. Cela rend cette implémentation impropre aux applications en temps réel. Il est cependant concevable de l'utiliser pour du filtrage en temps réel, si le nombre de points de lecture est suffisamment petit.

L'auteur a l'intention de créer un opérateur de plus haut niveau qui mélangera le signal original et le signal réverbéré, en utilisant automatiquement la bonne quantité de délai.

## Exemples

Créer le fichier de réponse impulsionnelle dans le domaine fréquentiel au moyen de l'utilitaire *cvanal utility* :

```
csound -Ucvanal ll_44.wav ll_44.cv
```

Déterminer la durée de la réponse impulsionnelle. Pour une grande précision, déterminer le nombre de trames d'échantillon dans le fichier de la réponse impulsionnelle, puis calculer la durée avec :

$\text{durée} = (\text{trames d'échantillons}) / (\text{taux d'échantillonnage du fichier son})$

Cela est du au fait que l'utilitaire *sndinfo* ne fournit la durée arrondie qu'au 10 ms les plus proches. Si l'on dispose d'un utilitaire qui fournit la durée avec la précision requise, alors il suffit d'utiliser directement la valeur retournée.

```
sndinfo ll_44.wav
```

length = 60822 samples, sample rate = 44100

Duration = 60822/44100 = 1.379s.

Déterminer le délai initial, s'il existe, de la réponse impulsionnelle. Si le délai initial de la réponse impulsionnelle n'a pas été enlevé, alors on peut ignorer cette étape. S'il a été enlevé, la seule manière de connaître le délai initial est de se procurer l'information séparément. Pour cet exemple, on suppose que le délai initial est de 60 ms (0.06 s).

Déterminer le délai qu'il faut nécessairement appliqué au signal original pour l'aligner sur le signal convolué :

Si  $kr = 441$ :

$1/kr = 0.0023$ , qui est  $\leq IR_{dur}$  (1.379s), ainsi :

$$\begin{aligned} \text{Delay1} &= \text{ceil}(IR_{dur} * kr) / kr \\ &= \text{ceil}(608.14) / 441 \\ &= 609/441 \\ &= 1.38s \end{aligned}$$

En prenant comme délai initial :

$$\begin{aligned} \text{Delay2} &= 0.06s \\ \text{Total delay} &= \text{delay1} - \text{delay2} \\ &= 1.38 - 0.06 \\ &= 1.32s \end{aligned}$$

Voici un exemple similaire de l'opcode `convolve`. Il utilise le fichier `convolve.csd` [exemples/convolve.csd].

### Exemple 151. Exemple de l'opcode `convolve`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
;-odac      ;;RT audio out
-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime output leave only the line below:
;-o convolve.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
; NB: 'Small' reverbs often require a much higher
; percentage of wet signal to sound interesting. 'Large'
; reverbs seem require less. Experiment! The wet/dry mix is
; very important - a small change can make a large difference.

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
;The analysis file is not system independent!
; create "rv_mono.wav" and "rv_stereo.wav" with cvanal first!

instr 1

imix = 0.25 ;wet/dry mix. Vary as desired.
ivol = 1 ;Overall volume level of reverb. May need to adjust
;when wet/dry mix is changed, to avoid clipping.

idel  filelen p4 ;calculate length and number of channels of soundfile
print idel
ichnls filenchnls p4
print ichnls

if (ichnls == 1) then

adry  soundin "fox.wav" ; input (dry) audio
awet  convolve adry,"rv_mono.cva" ; mono convolved (wet) audio
awet  diff awet ; brighten
adrydel delay (1-imix)*adry, idel ; Delay dry signal to align it with convolved signal
; Apply level adjustment here too.
outs  ivol*(adrydel+imix*awet),ivol*(adrydel+imix*awet) ; Mix wet & dry

else

adry  soundin "fox.wav" ; input (dry) audio
awet1, awet2 convolve adry,"rv_stereo.cva" ; stereo convolved (wet) audio
awet1  diff awet1 ; brighten left
awet2  diff awet2 ; and brighten right
adrydel delay (1-imix)*adry, idel ; Delay dry signal to align it with convolved signal
; Apply level adjustment here too.
outs  ivol*(adrydel+imix*awet1),ivol*(adrydel+imix*awet2) ; Mix wet & dry signals

endif

endin

</CsInstruments>
<CsScore>
```



```
i 1 0 4 "rv_mono.wav"  
i 1 5 4 "rv_stereo.wav"  
  
e  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*pconvolve*, *dconv*, *cvanal*.

## Crédits

Auteur : Greg Sullivan

1996

Nouveau dans la version 3.28

# copya2ftab

copya2ftab — Copie les données d'un vecteur dans une ftable.

## Description

L'opcode *copya2ftab* prend une variable k-tableau et en copie le contenu dans une ftable.

## Syntaxe

```
copya2ftab kArray[], ktab [, koffset]
```

```
copya2ftab iArray[], itab [, ioffset]
```

## Exécution

*kArray[]* -- tableau unidimensionnel pour la source.

*ktab* -- ftable pour la destination.

*koffset* -- décalage dans la ftable (0 par défaut).

## Exemple

Voici un exemple de l'opcode copya2ftab. Il utilise le fichier *copya2ftab.csd* [examples/copya2ftab.csd].

### Exemple 152. Exemple de l'opcode copya2ftab.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

;an 'empty' function table with 10 points
giTable ftgen 0, 0, -10, 2, 0

instr 1

;print initial values of giTable
puts      "\nInitial table content:", 1
indx      =      0
until indx == ftlen(giTable) do
iVal      table      indx, giTable
printf_i  "Table index %d = %f\n", 1, indx, iVal
indx += 1
```

```

    od

;create array
kArr[] init 10

;fill in values
kArr genarray 1, 10

;print array values
printf "%s", 1, "\nArray content:\n"
kndx = 0
until kndx == lenarray(kArr) do
    printf "kArr[%d] = %f\n", kndx+1, kndx, kArr[kndx]
    kndx += 1
od

;copy array values to table
copya2ftab kArr, giTable

;print modified values of giTable
printf "%s", 1, "\nModified table content after copya2ftab:\n"
kndx = 0
until kndx == ftlen(giTable) do
    table kndx, giTable
    printf "Table index %d = %f\n", kndx+1, kndx, kVal
    kndx += 1
od

;turn instrument off
turnoff
endin

</CsInstruments>
<CsScore>
i 1 0 0.1
</CsScore>
</CsSoundSynthesizer>

```

## Voir aussi

*copyf2array*

## Crédits

Auteur : John ffitch  
 Octobre 2011

Nouveau dans la version 5.15 de Csound.

Renommé dans la version 6.00 de Csound

Décalage ajouté dans la version 6.14

# copyf2array

copyf2array — Copie les données d'une ftable dans un vecteur.

## Description

L'opcode *copyf2array* prend une ftable et en copie le contenu dans une t-var.

## Syntaxe

```
copyf2array tab, kftbl
```

## Exécution

*tab* -- table pour la destination.

*kftbl* -- ftable pour la source.

## Exemple

Voici un exemple de l'opcode *copyf2array*. Il utilise le fichier *copyf2array.csd* [examples/copyf2array.csd].

### Exemple 153. Exemple de l'opcode *copyf2array*.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

;8 points sine wave function table
giSine ftgen 0, 0, 8, 10, 1

instr 1
;create array
kArr[] init 8

;copy table values in it
copyf2array kArr, giSine

;print values
kndx = 0
until kndx == lenarray(kArr) do
printf "kArr[%d] = %f\n", kndx+1, kndx, kArr[kndx]
kndx += 1
```

```
od  
  
;turn instrument off  
    turnoff  
endin  
  
</CsInstruments>  
<CsScore>  
i 1 0 0.1  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*copya2ftab*

## Crédits

Auteur : John fitch  
Octobre 2011

Nouveau dans la version 5.15 de Csound.

# COS

cos — Calcule une fonction cosinus.

## Description

Retourne cosinus de  $x$  ( $x$  en radians).

## Syntaxe

`cos(x)` (pas de restriction de taux)

`cos(k/i[])` (k- ou i-tableau)

## Exemples

Voici un exemple de l'opcode cos. Il utilise le fichier *cos.csd* [examples/cos.csd].

### Exemple 154. Exemple de l'opcode cos.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o cos.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
icos1 =      cos(0) ;cosine of 0 is 1
icos2 =      cos($M_PI_2) ;cosine of pi/2 (1.5707...) is 0
icos3 =      cos($M_PI) ;cosine of pi (3.1415...) is -1
icos4 =      cos($M_PI_2 * 3) ;cosine of 3/2pi (4.7123...) is 0
icos5 =      cos($M_PI * 2) ;cosine of 2pi (6.2831...) is 1
icos6 =      cos($M_PI * 4) ;cosine of 4pi is also 1 as it is periodically to 2pi
      print    icos1, icos2, icos3, icos4, icos5, icos6
endin

instr 2 ;cos used in panning, after an example from Hans Mikelson
aout      vco2      0.8, 220 ; sawtooth
kpan      linseg    p4, p3, p5 ;0 = left, 1 = right
kpan      =          kpan*$M_PI_2 ;range 0-1 becomes 0-pi/2
kpanl     =          cos(kpan)
kpanr     =          sin(kpan)
      outs      aout*kpanl, aout*kpanr
endin

</CsInstruments>
```

```
<CsScore>
i 1 0 0
i 2 0 5 0 1 ;move left to right
i 2 6 5 1 0 ;move right to left
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*cosh, cosinv, sin, sinh, sininv, tan, tanh, taninv*

# cosseg

cosseg — Trace une suite de segments entre les points spécifiés avec interpolation cosinusoidale.

## Description

Trace une suite de segments entre les points spécifiés avec interpolation cosinusoidale.

## Syntaxe

```
ares cosseg ia, idur1, ib [, idur2] [, ic] [...]  
kres cosseg ia, idur1, ib [, idur2] [, ic] [...]
```

## Initialisation

*ia* -- valeur initiale.

*ib*, *ic*, etc. -- valeur après *dur1* secondes, etc.

*idur1* -- durée en secondes du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

*idur2*, *idur3*, etc. -- durée en secondes des segments suivants. Une valeur nulle ou négative terminera la phase d'initialisation avec le point précédent, permettant au dernier segment ou à la dernière courbe définis de continuer durant toute l'exécution. La valeur par défaut est zéro.

## Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par 2 ou plus points spécifiés. La somme des valeurs *dur* peut égaier ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, la dernière valeur sera répétée jusqu'à la fin de la note.

## Exemples

Voici un exemple de l'opcode cosseg. Il utilise le fichier *cosseg.csd* [examples/cosseg.csd].

### Exemple 155. Exemple de l'opcode cosseg.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac      ;;realtime audio out  
;-iadc     ;;uncomment -iadc if realtime audio input is needed too  
; For Non-realtime ouput leave only the line below:  
;-o linseg.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>
```



```

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 2^10, 10, 1

instr 1

kcps = cspch(p4)
kenv cosseg 0, 0.25, 1, 0.75, 0 ; together = 1 sec
asig poscil kenv, kcps, giSine
outs asig, asig

endin

instr 2 ; scaling to duration

kcps = cspch(p4)
kenv cosseg 0, p3*0.25, 1, p3*0.75, 0
asig poscil kenv, kcps, giSine
outs asig, asig

endin

instr 3 ; with negative value

kcps = cspch(p4)
aenv cosseg 0, 0.1, 1, 0.5, -0.9, 0.4, 0
asig poscil aenv, kcps, giSine
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 1 7.00 ; = 1 sec, p3 fits exactly
i 1 2 2 7.00 ; = 2 sec, p3 truncated at 1 sec

i 2 4 1 7.00 ; scales to duration
i 2 6 2 7.00 ; of p3

i 3 9 2 7.00
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*expon, expseg, expsegb, expsegr, line, linseg, linsegr transeg transegb*

## Crédits

Auteur : John ffitch  
Août 2012

Nouveau dans la version 5.18

# cossegb

cossegb — Trace une suite de segments entre les points absolus spécifiés, avec interpolation cosinusoidale.

## Description

Trace une suite de segments entre les points absolus spécifiés, avec interpolation cosinusoidale.

## Syntaxe

```
ares cossegb ia, itim1, ib [, itim2] [, ic] [...]  
kres cossegb ia, itim1, ib [, itim2] [, ic] [...]
```

## Initialisation

*ia* -- valeur initiale.

*ib*, *ic*, etc. -- valeur à *tim1* secondes, etc.

*itim1* -- date en secondes de la fin du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

*itim2*, *itim3*, etc. -- date en secondes de la fin des segments suivants.

## Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par 2 ou plus points spécifiés. La dernière valeur *tim* peut égaier ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, la dernière valeur sera répétée jusqu'à la fin de la note.

## Exemples

Voici un exemple de l'opcode cossegb. Il utilise le fichier *cossegb.csd* [examples/cossegb.csd].

### Exemple 156. Exemple de l'opcode cossegb.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac      ;;realtime audio out  
;-iadc     ;;uncomment -iadc if realtime audio input is needed too  
; For Non-realtime ouput leave only the line below:  
;-o linseg.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 32
```

```

nchnls = 2
odbfs = 1

giSine ftgen 0, 0, 2^10, 10, 1

instr 1

kcps = cpspch(p4)
kenv cossegb 0, 0.25, 1, 1, 0
asig poscil kenv, kcps, giSine
outs asig, asig

endin

instr 2 ; scaling to duration

kcps = cpspch(p4)
kenv cossegb 0, p3*0.25, 1, p3, 0
asig poscil kenv, kcps, giSine
outs asig, asig

endin

</CsInstruments>
<CsScore>

i 1 0 1 7.00 ; = 1 sec, p3 fits exactly
i 1 2 2 7.00 ; = 2 sec, p3 truncated at 1 sec

i 2 4 1 7.00 ; scales to duration
i 2 6 2 7.00 ; of p3

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*cosseg, expon, expseg, expsegr, line, linseg linsegb linsegr transeg*

## Credits

Auteur : John ffitich

Août 2012

Nouveau dans la version 5.18

# cossegr

`cossegr` — Trace une suite de segments entre les points spécifiés avec interpolation cosinusoidale, avec un segment de relâchement..

## Description

Trace une suite de segments entre les points spécifiés avec interpolation cosinusoidale, avec un segment de relâchement..

## Syntaxe

```
ares cossegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz  
kres cossegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
```

## Initialisation

*ia* -- valeur initiale.

*ib, ic, etc.* -- valeur après *dur1* secondes, etc.

*idur1* -- durée en secondes du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

*idur2, idur3, etc.* -- durée en secondes des segments suivants. Une valeur nulle ou négative terminera la phase d'initialisation avec le point précédent, permettant au dernier segment ou à la dernière courbe définis de continuer durant toute l'exécution. La valeur par défaut est zéro.

*irel, iz* -- durée en secondes et valeur finale du segment de relâchement de la note.

## Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par 2 ou plus points spécifiés. La somme des valeurs *dur* peut éгалer ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, le dernier segment défini continuera dans la même direction.

*cossegr* fait partie des unités « r » de Csound qui contiennent un détecteur de fin de note et une extension de durée pour le relâchement. Quand la fin d'un événement ou un MIDI noteoff est détecté, la durée d'exécution de l'instrument courant est immédiatement allongée de *irel* secondes, de façon à ce que la valeur *iz* soit atteinte à la fin de cette période (quelque soit le segment dans lequel se trouvait l'unité). Les unités « r » peuvent aussi être modifiées par les vitesses nulles provoquant un message MIDI noteoff. S'il y a plusieurs extensions de durée dans un instrument, c'est la plus longue qui sera choisie.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note off, comme *linenr* et *expsegr*, ou bien l'on peut construire des enveloppes plus complexes au moyen de *xtratim* et de *release*. Noter que qu'il n'est pas nécessaire d'utiliser *xtratim* avec *cossegr*, car la durée est allongée automatiquement.

## Exemples

Voici un exemple de l'opcode *cossegr*. Il utilise le fichier *cossegr.csd* [examples/cossegr.csd].

## Exemple 157. Exemple de l'opcode cossegr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0   ;;realtime audio out and realtime midi in
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o linsegr.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

icps cpsmidi
iamp ampmidi .3

kenv cossegr 1, .05, 0.5, 1, 0
asig pluck kenv, icps, 200, 1, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 4096 10 1 ;sine wave

f0 30 ;runs 30 seconds
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*linenr, linsegr, expsegr, envlpxr, mxadsr, madsr expon, expseg, expsega line, linseg, xtratim, transegr*

## Crédits

Auteur : John ffitch  
Août 2012

Nouveau dans la version 5.18

# cosh

cosh — Calcule une fonction cosinus hyperbolique.

## Description

Retourne cosinus hyperbolique de  $x$  ( $x$  en radians).

## Syntaxe

`cosh(x)` (pas de restriction de taux)

`cosh(k/i[])` ( $k$ - ou  $i$ -tableau)

## Exemples

Voici un exemple de l'opcode cosh. Il utilise le fichier *cosh.csd* [examples/cosh.csd].

### Exemple 158. Exemple de l'opcode cosh.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cosh.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 1
  i1 = cosh(irad)

  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 1.543
```

## Voir aussi

*cos*, *cosinv*, *sin*, *sinh*, *sininv*, *tan*, *tanh*, *taninv*

## Crédits

Auteur : John ffitch

Nouveau dans la version 3.47

Exemple écrit par Kevin Conder.

# cosinv

cosinv — Calcule une fonction arccosinus.

## Description

Retourne arccosinus de  $x$  ( $x$  en radians).

## Syntaxe

`cosinv(x)` (pas de restriction de taux)

`cosinv(k/i[])` (k- ou i-tableau)

## Exemples

Voici un exemple de l'opcode cosinv. Il utilise le fichier *cosinv.csd* [examples/cosinv.csd].

### Exemple 159. Exemple de l'opcode cosinv.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cosinv.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 0.5
  i1 = cosinv(irad)

  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```



Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 1.047
```

## Voir aussi

*cos, cosh, sin, sinh, sininv, tan, tanh, taninv*

## Crédits

Auteur : John ffitch

Nouveau dans la version 3.48

Exemple écrit par Kevin Conder.

# cps2pch

**cps2pch** — Convertit une valeur de classe de hauteur en cycles par seconde (Hz) pour des divisions égales de l'octave.

## Description

Convertit une valeur de classe de hauteur en cycles par seconde (Hz) pour des divisions égales de l'octave.

## Syntaxe

`icps cps2pch ipch, iequal`

## Initialisation

*ipch* -- Nombre en entrée de la forme 8ve.pc, indiquant une octave et quelle note dans l'octave.

*iequal* -- S'il est positif, c'est le nombre d'intervalles égaux de division de l'octave. Doit être inférieur ou égal à 100. S'il est négatif, c'est le numéro d'une table de multiplicateurs de fréquence.



### Note

1. Les lignes suivantes sont équivalentes

```
ia = cpspch(8.02)
ib  cps2pch 8.02, 12
ic  cpsxpch 8.02, 12, 2, 1.02197503906
```

2. C'est un opcode, pas une fonction.
3. Des valeurs négatives pour *ipch* sont permises.

## Exemples

Voici un exemple de l'opcode `cps2pch`. Il utilise le fichier `cps2pch.csd` [examples/cps2pch.csd].

### Exemple 160. Exemple de l'opcode `cps2pch`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o cps2pch.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
```

```

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Use a normal twelve-tone scale.
  ipch = 8.02
  iequal = 12

  icps cps2pch ipch, iequal

  print icps
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra cette ligne :

```
instr 1: icps = 293.666
```

Voici un exemple de l'opcode `cps2pch` qui utilise une table de multiplicateurs de fréquence. Il utilise le fichier `cps2pch_ftable.csd` [examples/cps2pch\_ftable.csd].

### Exemple 161. Exemple de l'opcode `cps2pch` qui utilise une table de multiplicateurs de fréquence.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o cps2pch_ftable.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ipch = 8.02

  ; Use Table #1, a table of frequency multipliers.
  icps cps2pch ipch, -1

  print icps
endin

```

```

</CsInstruments>
<CsScore>

; Table #1: a table of frequency multipliers.
; Creates a 10-note scale of unequal divisions.
f 1 0 16 -2 1 1.1 1.2 1.3 1.4 1.6 1.7 1.8 1.9

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra cette ligne :

```
instr 1: icps = 313.951
```

Voici un exemple de l'opcode `cps2pch` qui utilise une échelle tempérée égale avec 19 divisions de l'octave. Il utilise le fichier `cps2pch_19et.csd` [exemples/cps2pch\_19et.csd].

### Exemple 162. Exemple de l'opcode `cps2pch` qui utilise une échelle tempérée égale avec 19 divisions de l'octave.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o cps2pch_19et.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Use 19ET scale.
ipch = 8.02
iequal = 19

icps cps2pch ipch, iequal

print icps
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra cette ligne :

```
instr 1:  icps = 281.429
```

## Voir aussi

*cpspch*, *cpsxpch*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
1997

Nouveau dans la version 3.492 de Csound

# cpsmidi

`cpsmidi` — Retourne le numéro de note de l'évènement MIDI courant, exprimé en Hz.

## Description

Retourne le numéro de note de l'évènement MIDI courant, exprimé en Hz.

## Syntaxe

`icps cpsmidi`

## Exécution

Retourne le numéro de note de l'évènement MIDI courant, exprimé en Hz, pour traitement local.



### **cpsmidi vs. cpsmidinn**

L'opcode *cpsmidi* ne produit des résultats significatifs qu'avec une note activée par le MIDI (soit en temps réel, soit depuis une partition MIDI avec l'option -F). Avec *cpsmidi*, la valeur du numéro de note MIDI provient de l'évènement MIDI qui est associé en interne avec l'instance de l'instrument. Au contraire, l'opcode *cpsmidinn* peut être utilisé dans n'importe quelle instance d'instrument de Csound, que celle-ci soit activée par un évènement MIDI, un évènement de partition, un évènement en ligne ou depuis un autre instrument. La valeur d'entrée de *cpsmidinn* peut provenir par exemple d'un p-champ dans une partition textuelle ou bien elle peut avoir été extraite au moyen de l'opcode *notnum* de l'évènement MIDI en temps réel qui a activé la note courante.

## Exemples

Voici un exemple de l'opcode `cpsmidi`. Il utilise le fichier *cpsmidi.csd* [examples/cpsmidi.csd].

### **Exemple 163. Exemple de l'opcode cpsmidi.**

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      -M0      ;;RT audio I/O with MIDI in
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime output leave only the line below:
; -o cpsmidi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
```

```
icps cpsmidi
asig oscil 0.6, icps, 1
print icps
outs asig, asig

endin

</CsInstruments>
<CsScore>
f0 20
;sine wave.
f 1 0 16384 10 1

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*aftouch, ampmidi, cpsmidib, cpstmid, midictrl, notnum, octmidi, octmidib, pchbend, pchmidi, pchmidib, veloc, cpsmidinn, octmidinn, pchmidinn*

## Crédits

Auteur : Barry L. Vercoe - Mike Berry  
MIT - Mills  
Mai 1997

# cpsmidib

cpsmidib — Retourne le numéro de note de l'évènement MIDI courant en le modifiant par la valeur courante de pitch-bend, exprimé en Hz.

## Description

Retourne le numéro de note de l'évènement MIDI courant en le modifiant par la valeur courante de pitch-bend, exprimé en Hz.

## Syntaxe

```
icps cpsmidib [irange]
```

```
kcps cpsmidib [irange]
```

## Initialisation

*irange* (facultatif) -- l'étendue du pitch-bend en demi-tons.

## Exécution

Retourne le numéro de note de l'évènement MIDI courant en le modifiant par la valeur courante de pitch-bend, exprimé en Hz. Disponible comme une valeur d'initialisation ou comme une valeur continue de taux-k.

## Exemples

Voici un exemple de l'opcode cpsmidib. Il utilise le fichier *cpsmidib.csd* [examples/cpsmidib.csd].

### Exemple 164. Exemple de l'opcode cpsmidib.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      -M0   ;;RT audio I/O with MIDI in
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o cpsmidi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; move pitch bend wheel while you play

kcps cpsmidib
asig oscil 0.6, kcps, 1
```



```
    printk2 kcps
    outs  asig, asig

    endin

    </CsInstruments>
    <CsScore>
    f0 20
    ;sine wave.
    f 1 0 16384 10 1

    e
    </CsScore>
    </CsoundSynthesizer>
```

## Voir aussi

*aftouch, ampmidi, cpsmidi, midictrl, notnum, octmidi, octmidib, pchbend, pchmidi, pchmidib, veloc*

## Crédits

Auteur : Barry L. Vercoe - Mike Berry  
MIT - Mills  
Mai 1997

# cpsmidinn

`cpsmidinn` — Convertit un numéro de note Midi en cycles par seconde.

## Description

Convertit un numéro de note Midi en cycles par seconde.

## Syntaxe

`cpsmidinn` (MidiNoteNumber) (arguments de taux-i ou -k seulement)

où l'argument entre parenthèses peut être une expression.

## Exécution

`cpsmidinn` est une fonction qui prend une valeur de taux-i ou de taux-k représentant un numéro de note Midi et qui retourne la valeur de fréquence équivalente en cycles par seconde (Hertz). Cette conversion suppose que le do médian est la note Midi numéro 60 et que le la du diapason est accordé à 440 Hz. Les numéros de note Midi sont par définition des nombres entiers compris entre 0 et 127 mais des valeurs fractionnaires ou des valeurs en dehors de cet intervalle seront correctement interprétées.



### `cpsmidinn` vs. `cpsmidi`

L'opcode `cpsmidinn` peut être utilisé dans n'importe quelle instance d'instrument de Csound, que celle-ci soit activée depuis un événement Midi, un événement de partition, un événement en ligne, ou depuis un autre instrument. La valeur d'entrée de `cpsmidinn` peut provenir par exemple d'un p-champ dans une partition textuelle ou bien avoir été retrouvée au moyen de l'opcode `notnum` à partir de l'évènement Midi en temps réel qui a activé la note courante. Le numéro de note Midi à convertir doit être spécifié comme une expression de taux-i ou de taux-k. D'un autre côté, l'opcode `cpsmidi` ne fournit des résultats significatifs qu'avec une note activée par le Midi (soit en temps réel soit à partir d'une partition Midi avec l'option -F). Avec `cpsmidi`, la valeur du numéro de note Midi provient de l'évènement Midi associé à l'instance d'instrument, et aucune source ni aucune expression ne peuvent être spécifiées pour cette valeur.

`cpsmidinn` et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

**Tableau 6. Valeurs de Hauteur et de Fréquence**

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme

deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par contraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.



## Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

## Exemples

Voici un exemple de l'opcode *cpsmidinn*. Il utilise le fichier *cpsmidinn.csd* [examples/cpsmidinn.csd].

### Exemple 165. Exemple de l'opcode *cpsmidinn*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
; Prints a table showing the equivalents of all Midi
; note numbers from 0-127 in cycles-per-second,
; octave.decimal, and octave.pitchclass units.

<CsOptions>
; Select audio/midi flags here according to platform.
; This example produces no audio, so we render in
; non-realtime and turn off sound to disk:
-n
</CsOptions>
<CsInstruments>

instr 1
; i-time loop to print conversion table
imidiNN = 0
loop1:
  icps = cpsmidinn(imidiNN)
  ioct = octmidinn(imidiNN)
  ipch = pchmidinn(imidiNN)
```

```

    print    imidiNN, icps, ioct, ipch

    imidiNN = imidiNN + 1
    if (imidiNN < 128) igoto loop1
endin

instr 2
; test k-rate converters
kMiddleC = 60
kcps = cpsmidinn(kMiddleC)
koct = octmidinn(kMiddleC)
kpch = pchmidinn(kMiddleC)

printks "%d %f %f %f\n", 1.0, kMiddleC, kcps, koct, kpch
endin

</CsInstruments>
<CsScore>
i1 0 0
i2 0 0.1
e

</CsScore>
</CsoundSynthesizer>

```

Voici un autre exemple de l'opcode `cpsmidinn`. Il utilise le fichier `cpsmidinn2.csd` [examples/cpsmidinn2.csd].

### Exemple 166. Second exemple de l'opcode `cpsmidinn`.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
;;;RT audio out, midi in, note=p4 and velocity=p5
-odac -+rtmidi=virtual -M0d --midi-key=4 --midi-velocity-amp=5
;-iadc    ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o cpsmidinn.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

massign 0, 1 ;assign all midi to instr. 1

instr 1 ;play virtual keyboard

inote = p4
icps = cpsmidinn(inote)
asig oscil 0.6, icps, 1
print icps
outs asig, asig

endin

</CsInstruments>
<CsScore>
f0 20
;sine wave.
f 1 0 16384 10 1
;play note from score too

```

```
i1 0 1 60  
e  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*octmidinn, pchmidinn, cpsmidi, notnum, cpspch, cpsoct, octcps, octpch, pchoct*

## Crédits

Dérivé à partir des convertisseurs de valeur originaux de Barry Vercoe.

Nouveau dans la version 5.07

# cpsoct

`cpsoct` — Convertit une valeur octave-point-partie-décimale en cycles par seconde.

## Description

Convertit une valeur octave-point-partie-décimale en cycles par seconde.

## Syntaxe

`cpsoct` (*oct*) (pas de restriction de taux)

où l'argument entre parenthèses peut être une expression.

## Exécution

`cpsoct` et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

**Tableau 7. Valeurs de Hauteur et de Fréquence**

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	<i>pch</i>
octave point partie décimale	<i>oct</i>
cycles par seconde	<i>cps</i>
Numéro de note Midi (0-127)	<i>midinn</i>

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par constraste, la conversion *cpsoct*(8.75 + *k1*) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque *k*-période car c'est le taux de variation de *k1*.



## Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

## Exemples

Voici une exemple de l'opcode *cpsoct*. Il utilise le fichier *cpsoct.csd* [examples/cpsoct.csd].

### Exemple 167. Exemple de l'opcode *cpsoct*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o cpsoct.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; Convert octave-point-decimal value into Hz

ioct = p4
icps = cpsoct(ioct)
print icps
asig oscil 0.7, icps, 1
outs asig, asig
endin

</CsInstruments>
<CsScore>
;sine wave.
f 1 0 16384 10 1

i 1 0 1 8.75
i 1 + 1 8.77
i 1 + 1 8.79
i 1 + .5 6.30

e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra ces lignes :

```
instr 1: icps = 440.000  
instr 1: icps = 446.110  
instr 1: icps = 452.344  
instr 1: icps = 80.521
```

## Voir aussi

*cpspch, octcps, octpch, pchoct, cpsmidinn, octmidinn, pchmidinn*



# cpspch

cpspch — Convertit une valeur de classe de hauteur en cycles par seconde.

## Description

Convertit une valeur de classe de hauteur en cycles par seconde.

## Syntaxe

**cpspch** (pch) (arguments de taux-i ou -k seulement)

où l'argument entre parenthèses peut être une expression.

## Exécution

*cpsoct* et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

**Tableau 8. Valeurs de Hauteur et de Fréquence**

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par constraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.



## Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

Si vous avez besoin de plus de précision de calcul, utilisez plutôt *cps2pch* ou *cpsxpch*.

## Exemples

Voici un exemple de l'opcode *cpspch*. Il utilise le fichier *cpspch.csd* [examples/cpspch.csd].

### Exemple 168. Exemple de l'opcode *cpspch*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o cpspch.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; Convert pitch-class value into Hz

ipch = p4
icps = cpspch(ipch)
print icps
asig oscil 0.7, icps, 1
outs asig, asig

endin

</CsInstruments>
<CsScore>
;sine wave.
f 1 0 16384 10 1

i 1 0 1 8.01
i 1 + 1 8.02
i 1 + 1 8.03
i 1 + .5 5.09

e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra ces lignes :

```
instr 1: icps = 277.167  
instr 1: icps = 293.656  
instr 1: icps = 311.101  
instr 1: icps = 54.995
```

## Voir aussi

*cps2pch, cpsoct, cpsxpch, octcps, octpch, pchoct, cpsmidinn, octmidinn, pchmidinn*

# cpstmid

cpstmid — Retourne un numéro de note MIDI (possibilité d'échelles micro-tonales).

## Description

Cette unité est semblable à *cpsmidi*, mais elle permet de particulariser des échelles micro-tonales.

## Syntaxe

```
icps cpstmid ifn
```

## Initialisation

*ifn* -- table de fonction contenant les paramètres (*numgrades*, *interval*, *basefreq*, *basekeymidi*) et les rapports d'accordage.

## Exécution

Seulement durant l'initialisation.

*cpsmid* nécessite cinq paramètres. Le premier, *ifn*, est le numéro de la table de fonction des rapports d'accordage, et les autres paramètres sont contenus dans la table de fonction elle-même. La table de fonction *ifn* doit être générée par *GEN02*, sans normalisation. Les quatre premières valeurs stockées dans cette fonction sont :

1. *numgrades* -- le nombre de degrés de l'échelle micro-tonale
2. *interval* -- l'intervalle de fréquence couvert avant de répéter les rapports des degrés. Par exemple, 2 pour une octave, 1.5 pour une quinte, etc.
3. *basefreq* -- la fréquence de base de l'échelle en Hz
4. *basekeymidi* -- le numéro de note MIDI auquel *basefreq* est assigné sans modification

Après ces quatre valeurs, on peut commencer à insérer les rapports d'accordage. Par exemple, pour une échelle standard sur 12 notes avec la fréquence de base 261 Hz assignée à la touche numéro 60, l'instruction *f* correspondante dans la partition pour générer la table sera :

```
;      numgrades interval basefreq basekeymidi tuning ratios (equal temp)
f1 0 64 -2  12    2    261    60    1 1.059463094359 1.122462048309 1.189207115003 ...etc...
```

Un autre exemple avec une échelle de 24 notes dont la fréquence de base de 440 Hz est assignée à la touche numéro 48, et un intervalle de répétition de 1.5 :

```
;      numgrades interval basefreq basekeymidi tuning-ratios (equal temp)
f1 0 64 -2  24    1.5    440    48    1 1.01 1.02 1.03 ...etc...
```

## Exemples

Voici un exemple de l'opcode *cpstmid*. Il utilise le fichier *cpstmid.csd* [examples/cpstmid.csd].

## Exemple 169. Exemple de l'opcode `cpstmid`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      -M0   ;;RT audio I/O with MIDI in
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime output leave only the line below:
; -o cpstmid.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
; after an example from Kevin Conder
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

; Table #1, a normal 12-tone equal temperament scale.
; numgrades = 12 (twelve tones)
; interval = 2 (one octave)
; basefreq = 261.659 (Middle C)
; basekeymidi = 60 (Middle C)
gitemp ftgen 1, 0, 64, -2, 12, 2, 261.659, 60, 1.00, \
          1.059, 1.122, 1.189, 1.260, 1.335, 1.414, \
          1.498, 1.588, 1.682, 1.782, 1.888, 2.000

instr 1

ifn = 1
icps cpstmid ifn
print icps
asig oscil 0.6, icps, 2
outs asig, asig

endin

</CsInstruments>
<CsScore>
f 0 20
;sine wave.
f 2 0 16384 10 1

e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*cpsmidi*, *GEN02*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1998

Nouveau dans la version 3.492 de Csound.

# cpstun

cpstun — Retourne des valeurs d'échelle microtonale au taux-k.

## Description

Retourne des valeurs d'échelle microtonale au taux-k.

## Syntaxe

kcps **cpstun** ktrig, kindex, kfn

## Exécution

*kcps* -- Valeur de retour en cycles par seconde.

*ktrig* -- Un signal utilisé pour déclencher l'évaluation.

*kindex* -- Un nombre entier servant d'indice dans l'échelle.

*kfn* -- Table de fonction contenant les paramètres (numgrades, interval, basefreq, basekeymidi) ainsi que les rapports de hauteur.

Cet opcode est similaire à *cpstmid*, mais son fonctionnement ne nécessite pas le MIDI.

*cpstun* travaille au taux-k. Il permet d'obtenir des échelles microtonales personnalisées. Il nécessite le numéro d'une table de fonction qui contient les rapports de hauteur, et quelques autres paramètres stockés dans la table elle-même.

L'argument *kindex* reçoit des nombres entiers indiquant quel degré de l'échelle donnée doit être converti en Hz. Dans *cpstun*, une nouvelle valeur ne sera évaluée que lorsque *ktrig* contiendra une valeur non nulle. La table de fonction *kfn* sera générée par *GEN02*, les quatre premières valeurs stockées dans la table étant des paramètres qui expriment :

- numgrades -- Le nombre de degrés de l'échelle microtonale.
- interval -- L'intervalle de fréquence couvert avant de répéter les rapports des degrés, par exemple 2 pour une octave, 1,5 pour une quinte, etc.
- basefreq -- La fréquence de base de l'échelle en cycles par seconde.
- basekey -- L'indice entier dans l'échelle auquel la fréquence de base sera affectée sans changement.

On peut insérer les rapports de hauteur après ces quatre valeurs. Par exemple, pour une échelle standard de 12 degrés avec une fréquence de base de 261 Hz affectée au numéro de touche 60, l'instruction f de la partition pour générer la table sera :

```
;          numgrades    basefreq    tuning-ratios (eq.temp) .....
;          interval      basekey
f1 0 64 -2  12         2        261    60    1    1.059463 1.12246 1.18920 ..etc...
```

Un autre exemple avec une échelle de 24 degrés et une fréquence de base de 440 affectée au numéro de touche 48, et un intervalle de répétition de 1,5 :

```

;                               numgrades    basefreq    tuning-ratios .....
;                               interval      basekey
f1 0 64 -2                    24      1.5      440      48      1      1.01  1.02  1.03  ..etc...

```

## Exemples

Voici un exemple de l'opcode `cpstun`. Il utilise le fichier `cpstun.csd` [examples/cpstun.csd].

### Exemple 170. Exemple de l'opcode `cpstun`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpstun.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Table #1, a normal 12-tone equal temperament scale.
; numgrades = 12 (twelve tones)
; interval = 2 (one octave)
; basefreq = 261.659 (Middle C)
; basekeymidi = 60 (Middle C)
gitemp ftgen 1, 0, 64, -2, 12, 2, 261.659, 60, 1.00, \
          1.059, 1.122, 1.189, 1.260, 1.335, 1.414, \
          1.498, 1.588, 1.682, 1.782, 1.888, 2.000

; Instrument #1.
instr 1
; Set the trigger.
ktrig init 1

; Use Table #1.
kfn init 1

; If the base key (note #60) is C, then 9 notes
; above it (note #60 + 9 = note #69) should be A.
kindex init 69

k1 cpstun ktrig, kindex, kfn

printk2 k1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

```

```
</CsScore>  
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
i1    440.11044
```

## Voir aussi

*cpstmid*, *cpstuni*, *GEN02*

## Crédits

Exemple écrit par Kevin Conder.



# cpstuni

cpstuni — Retourne des valeurs d'échelle microtonale au taux-i.

## Description

Retourne des valeurs d'échelle microtonale au taux-i.

## Syntaxe

```
icps cpstuni index, ifn
```

## Initialisation

*icps* -- Valeur de retour en cycles par seconde.

*index* -- Un nombre entier servant d'indice dans l'échelle.

*ifn* -- Table de fonction contenant les paramètres (numgrades, interval, basefreq, basekeymidi) ainsi que les rapports de hauteur.

## Exécution

Cet opcode est similaire à *cpstmid*, mais son fonctionnement ne nécessite pas le MIDI.

*cpstuni* travaille au taux-i. Il permet d'obtenir des échelles microtonales personnalisées. Il nécessite le numéro d'une table de fonction qui contient les rapports de hauteur, et quelques autres paramètres stockés dans la table elle-même.

L'argument *index* reçoit un nombre entier indiquant quel degré de l'échelle donnée doit être converti en Hz. La table de fonction *ifn* sera générée par *GEN02*, les quatre premières valeurs stockées dans la table étant des paramètres qui expriment :

- numgrades -- Le nombre de degrés de l'échelle microtonale.
- interval -- L'intervalle de fréquence couvert avant de répéter les rapports des degrés, par exemple 2 pour une octave, 1,5 pour une quinte, etc.
- basefreq -- La fréquence de base de l'échelle en cycles par seconde.
- basekey -- L'indice entier dans l'échelle auquel la fréquence de base sera affectée sans changement.

On peut insérer les rapports de hauteur après ces quatre valeurs. Par exemple, pour une échelle standard de 12 degrés avec une fréquence de base de 261 Hz affectée au numéro de touche 60, l'instruction f de la partition pour générer la table sera :

```
;          numgrades    basefreq    tuning-ratios (eq.temp) .....
;          interval      basekey
f1 0 64 -2 12      2      261      60      1      1.059463 1.12246 1.18920 ..etc...
```

Un autre exemple avec une échelle de 24 degrés et une fréquence de base de 440 affectée au numéro de touche 48, et un intervalle de répétition de 1,5 :

```

;                               numgrades    basefreq    tuning-ratios .....
;                               interval      basekey
f1 0 64 -2                    24      1.5      440      48      1      1.01  1.02  1.03  ..etc...

```

## Exemples

Voici un exemple de l'opcode `cpstuni`. Il utilise le fichier `cpstuni.csd` [examples/cpstuni.csd].

### Exemple 171. Exemple de l'opcode `cpstuni`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpstuni.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Table #1, a normal 12-tone equal temperament scale.
; numgrades = 12 (twelve tones)
; interval = 2 (one octave)
; basefreq = 261.659 (Middle C)
; basekeymidi = 60 (Middle C)
gitemp ftgen 1, 0, 64, -2, 12, 2, 261.659, 60, 1.00, \
          1.059, 1.122, 1.189, 1.260, 1.335, 1.414, \
          1.498, 1.588, 1.682, 1.782, 1.888, 2.000

; Instrument #1.
instr 1
; Use Table #1.
ifn = 1

; If the base key (note #60) is C, then 9 notes
; above it (note #60 + 9 = note #69) should be A.
index = 69

i1 cpstuni index, ifn

print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra cette ligne :

```
instr 1:  il = 440.110
```

## Voir aussi

*cpstmid*, *cpstun*, *GEN02*

## Crédits

Ecrit par Gabriel Maldonado.

Exemple écrit par Kevin Conder.

# cpsxpch

cpsxpch — Convertit une valeur de classe de hauteur en cycles par seconde (Hz) pour des divisions égales de n'importe quel intervalle.

## Description

Convertit une valeur de classe de hauteur en cycles par seconde (Hz) pour des divisions égales de n'importe quel intervalle. Le nombre de divisions ne doit pas dépasser 100.

## Syntaxe

icps **cpsxpch** ipch, iequal, irepeat, ibase

## Initialisation

*ipch* -- Nombre en entrée de la forme 8ve.pc, indiquant une octave et quelle note dans l'octave.

*iequal* -- S'il est positif, c'est le nombre d'intervalles égaux de division de l'« octave ». Doit être inférieur ou égal à 100. S'il est négatif, c'est le numéro d'une table de multiplicateurs de fréquence.

*irepeat* -- Nombre indiquant l'intervalle qui est l'« octave ». Le nombre 2 est utilisé pour des divisions de l'octave, 3 pour une douzième, 4 pour deux octaves, ainsi de suite. Ce nombre ne doit pas forcément être un entier, mais il doit être positif.

*ibase* -- La fréquence qui correspond à la hauteur 0.0



### Note

1. Les lignes suivantes sont équivalentes

```
ia = cpspch(8.02)
ib  cps2pch 8.02, 12
ic  cpsxpch 8.02, 12, 2, 1.02197503906
```

2. C'est un opcode, pas une fonction.

3. Des valeurs négatives sont permises pour *ipch*, mais pas pour *irepeat*, ni pour *iequal* ou *ibase*.

## Exemples

Voici un exemple de l'opcode cpsxpch. Il utilise le fichier *cpsxpch.csd* [examples/cpsxpch.csd].

### Exemple 172. Exemple de l'opcode cpsxpch.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
```

```

; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpsxpch.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Use a normal twelve-tone scale.
ipch = 8.02
iequal = 12
irepeat = 2
ibase = 1.02197503906

icps cpsxpch ipch, iequal, irepeat, ibase

print icps
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra cette ligne :

```
instr 1: icps = 293.666
```

Voici un exemple de l'opcode `cpsxpch` qui utilise une échelle tempérée égale avec 10,5 divisions de l'octave. Il utilise le fichier `cpsxpch_105et.csd` [exemples/cpsxpch\_105et.csd].

**Exemple 173. Exemple de l'opcode `cpsxpch` qui utilise une échelle tempérée égale avec 10,5 divisions de l'octave.**

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpsxpch_105et.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

```

```

; Instrument #1.
instr 1
  ; Use a 10.5ET scale.
  ipch = 4.02
  iequal = 21
  irepeat = 4
  ibase = 16.35160062496

  icps cpsxpch ipch, iequal, irepeat, ibase

  print icps
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra cette ligne :

```
instr 1: icps = 4776.824
```

Voici un exemple de l'opcode cpsxpch qui utilise une échelle de Pierce centrée sur le la médian. Il utilise le fichier *cpsxpch\_pierce.csd* [examples/cpsxpch\_pierce.csd].

#### Exemple 174. Exemple de l'opcode cpsxpch qui utilise une échelle de Pierce centrée sur le la médian.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cpsxpch_pierce.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Use a Pierce scale centered on middle A.
  ipch = 2.02
  iequal = 12
  irepeat = 3
  ibase = 261.62561

  icps cpsxpch ipch, iequal, irepeat, ibase

  print icps
endin

```

```
</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  icps = 2827.762
```

## Voir aussi

*cspch*, *cps2pch*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
1997

Nouveau dans la version 3.492 de Csound

# cpumeter

cpumeter — Retourne le taux d'utilisation cpu soit globalement soit par coeur.

## Description

Retourne le taux d'utilisation de l'unité centrale soit globalement soit par coeur, pour contrôler à quel point le traitement se rapproche des capacités maximales.

## Syntaxe

```
ktot[,kcpu1, kcpu2,...]cpumeter ifreq
```

## Initialisation

*ifreq* est le temps de rafraichissement de la mesure en secondes. S'il est trop court, alors on verra principalement les valeurs zéro ou cent. Une valeur de 0.1 semble acceptable.

## Exécution

*cpumeter* lit la durée totale du temps de repos lors des dernières *ifreq* secondes et retourne le pourcentage d'utilisation. Si le résultat *ktot* n'est pas suffisant, on peut récupérer le même type de résultat pour chaque coeur.

## Exemples

Voici un exemple de l'opcode cpumeter. Il utilise le fichier *cpumeter.csd* [examples/cpumeter.csd].

### Exemple 175. Exemple de l'opcode cpumeter.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o cpumeter.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 1
0dbfs = 1

instr 1 ;cpu metering; stop when too large
k0 cpumeter 0.1
printk2 k0
if k0>70 then
  event "i", 3, 0.1, 1
```



```

        endif
    endin

    instr 2
        event_i    "i", 2, 1, 1000
        asig oscil 0.2, 440, 1
        out asig
    endin

    instr 3
        exitnow
    endin
</CsInstruments>
<CsScore>
f 1 0 32768 10 1 ; sine wave

i 1 0 1000
i 2 0 1000
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*maxalloc, prealloc, cpuprc*

## Crédits

Auteur : John ffitch  
 Mai 2011

Nouveau dans la version 5.14 de Csound, seulement pour Linux/Unix et OSX.

# cpuprc

cpuprc — Contrôle l'allocation des ressources cpu par instrument, pour optimiser la sortie en temps réel.

## Description

Contrôle l'allocation des ressources cpu par instrument, pour optimiser la sortie en temps réel.

## Syntaxe

```
cpuprc insnum, ipercent  
cpuprc Sinsname, ipercent
```

## Initialisation

*insnum* -- numéro de l'instrument

*Sinsname* -- nom de l'instrument

*ipercent* -- pourcentage du temps de traitement cpu à allouer. On peut aussi l'exprimer sous forme fractionnaire.

## Exécution

*cpuprc* fixe le pourcentage du temps de traitement cpu utilisé par un instrument, afin d'éviter un sous-remplissage du tampon dans les exécutions en temps réel. L'utilisateur doit fixer la valeur de *ipercent* pour chaque instrument qui sera activé en temps réel. En supposant que le temps de traitement cpu total soit 100% en théorie, cette valeur de pourcentage ne peut être définie qu'empiriquement, car il y a trop de facteurs qui contribuent à limiter la polyphonie en temps réel sur différents ordinateurs.

Par exemple si *ipercent* est fixé à 5% pour l'instrument 1, le nombre maximum de voix que l'on pourra allouer en temps réel est 20 ( $5\% * 20 = 100\%$ ). Si l'on essaye de jouer une note supplémentaire alors que les 20 notes précédentes sont toujours jouées, Csound empêche l'allocation de cette note et affiche le message d'avertissement suivant :

impossible d'allouer la dernière note car il n'y a plus de temps cpu disponible

Afin d'éviter les sous-remplissages de tampon audio, il est suggéré de fixer le nombre maximum de voix à une valeur légèrement inférieure à la puissance de traitement réelle de l'ordinateur. Parfois, un instrument peut avoir besoin de plus de temps de traitement que la normale. Si, par exemple, l'instrument contient un oscillateur qui lit une table trop grande pour la mémoire cache, il sera plus lent qu'en temps normal. De plus, chaque programme s'exécutant concurremment dans l'environnement multitâche, peut consommer de la puissance processeur à divers degrés.

Au début, tous les instruments reçoivent une valeur par défaut de *ipercent* égale à 0.0% (c'est-à-dire un temps de traitement nul équivalent à une vitesse de processeur infinie). Ce réglage convient très bien pour les sessions en temps différé.

Toutes les instances de *cpuprc* doivent être définies dans la section d'en-tête et non dans le corps de l'instrument.

## Exemples

Voici un exemple de l'opcode `cpuprc`. Il utilise le fichier `cpuprc.csd` [examples/cpuprc.csd].

### Exemple 176. Exemple de l'opcode `cpuprc`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o cpuprc.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

cpuprc 1, 2
cpuprc 2, 30

instr 1 ;cpu processing-time percent usage is set to 2% for each note

asig oscil 0.2, 440, 1
outs asig, asig

endin

instr 2 ;cpu processing-time percent usage is set to 30% for each note
;so the 4 notes of the score exceeds 100% by far
asig oscil 0.2, 440, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 32768 10 1 ; sine wave

i 1 0 1
i 1 0 1
i 1 0 1
i 1 0 1

;too many notes to process,
;check Csound output!
i 2 3 1
i 2 3 1
i 2 3 1
i 2 3 1
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*maxalloc, prealloc*

## Crédits

Auteur : Gabriel Maldonado

Italie

Juillet 1999

Nouveau dans la version 3.57 de Csound ; ajout des instruments nommés dans la version 5.13

# cross2

cross2 — Synthèse croisée au moyen de TFR.

## Description

C'est une implémentation de synthèse croisée au moyen de TFR.

## Syntaxe

```
ares cross2 ain1, ain2, isize, ioverlap, iwin, kbias
```

## Initialisation

*isize* -- Taille de la TFR à effectuer. Plus la taille est grande, meilleure est la réponse en fréquence mais avec une réponse temporelle imprécise.

*ioverlap* -- Facteur de chevauchement des TFR, doit être une puissance de deux. Les meilleurs réglages sont 2 et 4. Un grand chevauchement prend un long temps de calcul.

*iwin* -- Table de fonction contenant la fenêtre à utiliser dans l'analyse. On peut créer cette fenêtre au moyen de la routine *GEN20*.

## Exécution

*ain1* -- Le son d'excitation. Doit contenir des fréquence élevées pour de meilleurs résultats.

*ain2* -- Le son modulant. Doit avoir une réponse en fréquence changeante (comme la parole) pour de meilleurs résultats.

*kbias* -- la proportion de synthèse croisée. 1 est la normale, 0 signifie pas de synthèse croisée.

## Exemples

Voici un exemple de l'opcode cross2. Il utilise les fichiers *cross2.csd* [examples/cross2.csd] et *fox.wav* [examples/fox.wav].

### Exemple 177. Exemple de l'opcode cross2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o cross2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
; after example from Kevin Conder
```

```

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;play audio file

aout soundin "fox.wav"
outs aout, aout
endin

instr 2 ;cross-synthesize

icps = p4
ifn = p5 ; Use the "ahhc.aiff" sound and "eeec.aiff"
ain1 oscil 0.6, p4, ifn
ain2 soundin "fox.wav" ; Use the "fox.wav" as modulator

    isize = 4096
    ioverlap = 2
    iwin = 3
    kbias init 1

aout cross2 ain1, ain2, isize, ioverlap, iwin, kbias
outs aout, aout
endin

</CsInstruments>
<CsScore>
;audio files
f 1 0 128 1 "ahhc.aiff" 0 4 0
f 2 0 128 1 "eeec.aiff" 0 4 0

f 3 0 2048 20 2 ;windowing function

i 1 0 3

i 2 3 3 50 1 ;"eeec.aiff"
i 2 + 3 50 2 ;"ahhc.aiff"
i 2 + 3 100 1 ;"eeec.aiff"
i 2 + 3 100 2 ;"ahhc.aiff"
i 2 + 3 250 1 ;"eeec.aiff"
i 2 + 3 250 2 ;"ahhc.aiff"
i 2 + 3 20 1 ;"eeec.aiff"
i 2 + 3 20 2 ;"ahhc.aiff"
e

</CsScore>
</CsSoundSynthesizer>

```

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1997

# crossfm

crossfm — Deux oscillateurs se modulant mutuellement en fréquence et/ou en phase.

## Description

Deux oscillateurs se modulant mutuellement en fréquence et/ou en phase.

## Syntaxe

```
a1, a2 crossfm xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]
a1, a2 crossfmi xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]
a1, a2 crosspm xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]
a1, a2 crosspmi xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]
a1, a2 crossfmpm xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]
a1, a2 crossfmpmi xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]
```

## Initialisation

*ifn1* -- numéro de la table de fonction pour l'oscillateur 1. Nécessite un point de garde.

*ifn2* -- numéro de la table de fonction pour l'oscillateur 2. Nécessite un point de garde.

*iphs1* (facultatif, 0 par défaut=0) -- phase initiale de la forme d'onde de la table *ifn1*, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation sera ignorée.

*iphs2* (facultatif, 0 par défaut=0) -- phase initiale de la forme d'onde de la table *ifn2*, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation sera ignorée.

## Exécution

*xfrq1* -- un facteur qui, lorsqu'il est multiplié par le paramètre *kcps* donne la fréquence de l'oscillateur 1.

*xfrq2* -- un facteur qui, lorsqu'il est multiplié par le paramètre *kcps* donne la fréquence de l'oscillateur 2.

*xndx1* -- l'indice de modulation de l'oscillateur 2 par l'oscillateur 1.

*xndx2* -- l'indice de modulation de l'oscillateur 1 par l'oscillateur 2.

*kcps* -- un dénominateur commun, en cycles par seconde, pour les fréquences des deux oscillateurs.

*crossfm* implémente un algorithme de modulation de fréquence croisée. La sortie de taux audio de l'oscillateur 1 module l'entrée en fréquence de l'oscillateur 2 tandis que la sortie audio de l'oscillateur 2 module l'entrée en fréquence de l'oscillateur 1. Cette double structure de rétroaction produit des sonorités riches avec parfois un comportement chaotique. *crossfmi* fonctionne comme *crossfm* mais en utilisant l'interpolation linéaire pour la lecture des tables.

*crosspm* et *crosspmi* implémentent la modulation de phase croisée entre deux oscillateurs.

*crossfmpm* et *crossfmpmi* implémentent une modulation de fréquence/phase croisée entre deux oscillateurs. L'oscillateur 1 est modulé en fréquence par l'oscillateur 2 tandis que l'oscillateur 2 est modulé en phase par l'oscillateur 1.

On peut lire mon *article* [<http://www.csoundjournal.com/issue12/crossfm.html>] dans le Csound Journal pour plus d'information.



## Avertissement

Ces opcodes peuvent produire des spectres très riches, particulièrement avec des indices de modulation importants et, dans certains cas des fréquences de repliement peuvent apparaître si le taux d'échantillonnage n'est pas suffisamment élevé. De plus, la sortie audio peut varier en fonction du taux d'échantillonnage à cause de la non-linéarité de l'algorithme. Dans Csound, deux autres opcodes présentent cette caractéristique : *planet* et *chuap*.

## Exemples

Voici un exemple de l'opcode *crossfm*. Il utilise le fichier *crossfm.csd* [examples/crossfm.csd].

### Exemple 178. Exemple de l'opcode *crossfm*.

```
<CsoundSynthesizer>

<CsOptions>
  -d -o dac
</CsOptions>

<CsInstruments>
  sr          =          96000
  ksmpps      =          10
  nchnls      =          2
  0dbfs       =          1

  FLpanel "crossfmForm", 600, 400, 0, 0
    gkfrq1, ihfrq1 FLcount "Freq #1", 0, 20000, 0.001, 1, 1, 200, 30, 20, 50, -1
    gkfrq2, ihfrq2 FLcount "Freq #2", 0, 20000, 0.001, 1, 1, 200, 30, 20, 130, -1
    gkndx1, gkndx2, ihndx1, ihndx2 FLjoy "Indexes", 0, 10, 0, 10, 0, 0, -1, -1, 200, 200, 300, 50

    FLsetVal_i 164.5, ihfrq1
    FLsetVal_i 263.712, ihfrq2
    FLsetVal_i 1.5, ihndx1
    FLsetVal_i 3, ihndx2
  FLpanelEnd
  FLrun

  maxalloc 1, 2

  instr 1
    kamp      linen      0.5, 0.01, p3, 0.5
    a1,a2     crossfm     gkfrq1, gkfrq2, gkndx1, gkndx2, 1, 1, 1
    outs      a1*kamp, a2*kamp
  endin
</CsInstruments>

<CsScore>
  f1 0 16384 10 1 0
  i1 0 60
  e
</CsScore>
</CsoundSynthesizer>
```



Dans cet exemple, on utilise une interface graphique FLTK pour contrôler en temps réel la fréquence des oscillateurs au moyen de deux contrôles *Flcount* et les indices de la modulation croisée avec un contrôle *FLjoy*. Il utilise un taux d'échantillonnage de 96000Hz.

## Voir aussi

Plus d'information sur ces opcodes : <http://www.csoundjournal.com/issue12/crossfm.html>, écrit par François Pinot.

## Crédits

Auteur : François Pinot  
2005-2009

Nouveau dans la version 5.12

# crunch

crunch — Modèle semi-physique d'un son de craquement.

## Description

*crunch* est un modèle semi-physique d'un son de craquement. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntaxe

```
ares crunch iamp, idettack [, inum] [, idamp] [, imaxshake]
```

## Initialisation

*iamp* -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (optional) -- (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 7.

*idamp* (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

$$\text{damping\_amount} = 0,998 + (\text{idamp} * 0,002)$$

La valeur par défaut de *damping\_amount* est 0,99806 ce qui signifie que la valeur par défaut de *idamp* est 0,03. Le maximum de *damping\_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 1,0.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

*imaxshake* (facultatif) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

## Exemples

Voici un exemple de l'opcode crunch. Il utilise le fichier *crunch.csd* [examples/crunch.csd].

### Exemple 179. Exemple de l'opcode crunch.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
```

```
; -o crunch.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
odbfs = 1

instr 1

asig crunch 0.8, 0.1, 7, p4
outs asig, asig

endin

</CsInstruments>
<CsScore>

i1 0 1 .9
i1 1 1 .1

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*cabasa, sandpaper, sekere, stix*

## Crédits

Auteur : Perry Cook, fait partie de PhOLIES (Physically-Oriented Library of Imitated Environmental Sounds)

Adapté par John ffitch

Université de Bath, Codemist Ltd.

Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

# ctrl14

**ctrl14** — Permet un signal MIDI sur 14 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Description

Permet un signal MIDI sur 14 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Syntaxe

```
idest ctrl14 ichan, ictlno1, ictlno2, imin, imax [, ifn]
kdest ctrl14 ichan, ictlno1, ictlno2, kmin, kmax [, ifn]
```

## Initialisation

*idest* -- signal de sortie

*ichan* -- numéro de canal MIDI (1-16)

*ictlno1* -- numéro de contrôleur pour l'octet de poids fort (0-127)

*ictlno2* -- numéro de contrôleur pour l'octet de poids faible (0-127)

*imin* -- la valeur décimale minimale de sortie définie par l'utilisateur

*imax* -- la valeur décimale maximale de sortie définie par l'utilisateur

*ifn* (facultatif) -- la table à lire lorsque l'indexation est requise. La table doit être normalisée. La sortie est mise à l'échelle entre les valeurs *imax* et *imin*.

## Exécution

*kdest* -- signal de sortie

*kmin* -- la valeur décimale minimale de sortie définie par l'utilisateur

*kmax* -- la valeur décimale maximale de sortie définie par l'utilisateur

*ctrl14* (contrôle MIDI sur 14 bit au taux-i et au taux-k) permet un signal MIDI sur 14 bit en nombres décimaux mis à l'échelle entre des limites minimale et maximale. Les valeurs minimale et maximale peuvent être variées au taux-k. Il peut utiliser en option une indexation de table. Il nécessite deux contrôleurs MIDI en entrée.

*ctrl14* est différent de *midic14* parce que il peut être inclu dans des instruments prévus pour une partition sans que Csound ne plante. Il a besoin du paramètre additionnel *ichan* contenant le canal MIDI du contrôleur. Le canal MIDI est le même pour tous les contrôleurs utilisés dans un opcode *ctrl14*.

## Exemples

Voici un exemple de l'opcode *ctrl14*. Il utilise le fichier *ctrl14.csd* [examples/ctrl14.csd].

## Exemple 180. Exemple de l'opcode `ctrl14`.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -M0 ;;RT audio I/O with MIDI in
;-iadc ;;uncomment -iadc if RT audio input is needed too
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; expects MIDI controller input on channel 1
; run and move your midi controller to see result

imax = 1
imin = 0
ichan = 1
ictlno = 7

initc14 1, ictlno, ictlno+1, 1 ; start at max. volume
kamp ctrl14 ichan, ictlno, ictlno+1, imin, imax ; controller 7
printk2 kamp
asig oscil kamp, 220, 1
outs asig, asig

endin

</CsInstruments>
<CsScore>
f 1 0 4096 10 1

i1 0 20

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

`ctrl7`, `ctrl21`, `initc7`, `initc14`, `initc21`, `midic7`, `midic14`, `midic21`

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

# ctrl21

ctrl21 — Permet un signal MIDI sur 21 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Description

Permet un signal MIDI sur 21 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Syntaxe

```
idest ctrl21 ichan, ictlno1, ictlno2, ictlno3, imin, imax [, ifn]
```

```
kdest ctrl21 ichan, ictlno1, ictlno2, ictlno3, kmin, kmax [, ifn]
```

## Initialisation

*idest* -- signal de sortie

*ichan* -- numéro de canal MIDI (1-16)

*ictlno1* -- numéro de contrôleur pour l'octet de poids fort (0-127)

*ictlno2* -- numéro de contrôleur pour l'octet de poids moyen (0-127)

*ictlno3* -- numéro de contrôleur pour l'octet de poids faible (0-127)

*imin* -- la valeur décimale minimale de sortie définie par l'utilisateur

*imax* -- la valeur décimale maximale de sortie définie par l'utilisateur

*ifn* (facultatif) -- la table à lire lorsque l'indexation est requise. La table doit être normalisée. La sortie est mise à l'échelle entre les valeurs *imax* et *imin*.

## Exécution

*kdest* -- signal de sortie

*kmin* -- la valeur décimale minimale de sortie définie par l'utilisateur

*kmax* -- la valeur décimale maximale de sortie définie par l'utilisateur

*ctrl21* (contrôle MIDI sur 21 bit au taux-i et au taux-k) permet un signal MIDI sur 21 bit en nombres décimaux mis à l'échelle entre des limites minimale et maximale. Les valeurs minimale et maximale peuvent être variées au taux-k. Il peut utiliser une indexation de table facultative. Il nécessite trois contrôleurs MIDI en entrée.

*ctrl21* est différent de *midic21* parce qu'il peut être inclu dans des instruments prévus pour une partition sans que Csound ne plante. Il a besoin du paramètre additionnel *ichan* contenant le canal MIDI du contrôleur. Le canal MIDI est le même pour tous les contrôleurs utilisés dans un opcode *ctrl21*.

## Exemples

Voici un exemple de l'opcode `ctrl21`. Il utilise le fichier `ctrl21.csd` [examples/ctrl21.csd].

### Exemple 181. Exemple de l'opcode `ctrl21`.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -M0 ;;;RT audio I/O with MIDI in
;-iadc ;;;uncomment -iadc if RT audio input is needed too
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; expects MIDI controller input on channel 1
; run and move your midi controller to see result

imax = 1
imin = 0
ichan = 1
ictlno = 7

initc21 1, ictlno, ictlno+1, ictlno+2. 1 ; start at max. volume
kamp ctrl21 ichan, ictlno, ictlno+1, ictlno+2, imin, imax ; controller 7
printk2 kamp
asig oscil kamp, 220, 1
outs asig, asig

endin

</CsInstruments>
<CsScore>
f 1 0 4096 10 1

i1 0 20

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

`ctrl7`, `ctrl14`, `initc7`, `initc14`, `initc21`, `midic7`, `midic14`, `midic21`

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

# ctrl7

*ctrl7* — Permet un signal MIDI sur 7 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Description

Permet un signal MIDI sur 7 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Syntaxe

```
idest ctrl7 ichan, ictlno, imin, imax [, ifn]
kdest ctrl7 ichan, ictlno, kmin, kmax [, ifn]
adest ctrl7 ichan, ictlno, kmin, kmax [, ifn] [, icutoff]
```

## Initialisation

*idest* -- signal de sortie

*ichan* -- canal MIDI (1-16)

*ictlno* -- numéro du contrôleur MIDI (0-127)

*imin* -- valeur décimale minimale de sortie définie par l'utilisateur

*imax* -- valeur décimale maximale de sortie définie par l'utilisateur

*ifn* (facultatif) -- la table à lire lorsque l'indexation est requise. La table doit être normalisée. La sortie est mise à l'échelle entre les valeurs *imin* et *imax*.

*icutoff* (facultatif) -- fréquence de coupure du filtre passe-bas pour lisser la sortie au taux-a.

## Exécution

*kdest*, *adest* -- signal de sortie

*kmin* -- valeur décimale minimale de sortie définie par l'utilisateur

*kmax* -- valeur décimale maximale de sortie définie par l'utilisateur

*ctrl7* (contrôle MIDI sur 7 bit au taux-i et au taux-k) permet un signal MIDI sur 7 bit en nombres décimaux mis à l'échelle entre des limites minimale et maximale. Il permet également en option une indexation de table sans interpolation. Les valeurs minimale et maximale peuvent varier au taux-k.

*ctrl7* diffère de *midic7* parce que il peut être inclu dans des instruments prévus pour une partition sans que Csound ne plante. Il a aussi besoin du paramètre additionnel *ichan* contenant le canal MIDI du contrôleur.

La version de *ctrl7* au taux-a fournit en sortie une variable de taux-a, qui est passée par un filtre passe-bas (lissée). Il y a un paramètre facultatif *icutoff*, pour établir la fréquence de coupure du filtre passe-bas. Sa valeur par défaut est 5.



## Exemples

Voici un exemple de l'opcode `ctrl7`. Il utilise le fichier `ctrl7.csd` [examples/ctrl7.csd].

### Exemple 182. Exemple de l'opcode `ctrl7`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -M0 ;;RT audio I/O with MIDI in
;-iadc ;;uncomment -iadc if RT audio input is needed too
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; expects MIDI controller input on channel 1
; run and move your midi controller to see result

imax = 1
imin = 0
ichan = 1
ictlno = 7

initc7 1, 7, 1 ; start at max. volume
kamp ctrl7 ichan, ictlno, imin, imax ; controller 7
printk2 kamp
asig oscil kamp, 220, 1
outs asig, asig

endin

</CsInstruments>
<CsScore>
f 1 0 4096 10 1

i1 0 20

e
</CsScore>
</CsoundSynthesizer>
```



### Note

`ctrl7` ne retourne des k-valeurs qu'après le premier mouvement du contrôleur. Pour fixer une k-valeur initiale, il faut appeler `initc7`. `ctrlinit` ne sert à rien dans ce cas.

## Voir aussi

`ctrl14`, `ctrl21`, `initc7`, `initc14`, `initc21`, `midic7`, `midic14`, `midic21`

## Crédits

Auteur : Gabriel Maldonado

Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

La version de *ctrl7* au taux-a a été ajoutée dans la version 5.06

# ctrlinit

ctrlinit — Initialise les valeurs pour un groupe de contrôleurs MIDI.

## Description

Initialise les valeurs pour un groupe de contrôleurs MIDI.

## Syntaxe

```
ctrlinit ichnl, ictlno1, ival1 [, ictlno2] [, ival2] [, ictlno3] \  
[, ival3] [...ival32]
```

## Initialisation

*ichnl* -- numéro de canal MIDI (1-16)

*ictlno1*, *ictlno1*, etc. -- numéros de contrôleurs MIDI (0-127)

*ival1*, *ival2*, etc. -- valeur initiale pour le contrôleur MIDI de numéro correspondant

## Exécution

Initialise les valeurs pour un groupe de contrôleurs MIDI.

## Voir aussi

*massign*

## Crédits

Auteur : Barry L. Vercoe - Mike Berry  
MIT, Cambridge, Mass.

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

# cuserrnd

cuserrnd — Générateur de nombres aléatoires de distribution continue définie par l'utilisateur.

## Description

Générateur de nombres aléatoires de distribution continue définie par l'utilisateur.

## Syntaxe

```
aout cuserrnd kmin, kmax, ktableNum
iout cuserrnd imin, imax, itableNum
kout cuserrnd kmin, kmax, ktableNum
```

## Initialisation

*imin* -- limite inférieure de l'intervalle

*imax* -- limite supérieure de l'intervalle

*itableNum* -- numéro d'une table contenant la fonction de la distribution aléatoire. Cette table est générée par l'utilisateur. Voir GEN40, GEN41 et GEN42. La longueur de la table peut être différente d'une puissance de 2.

## Exécution

*ktableNum* -- numéro d'une table contenant la fonction de la distribution aléatoire. Cette table est générée par l'utilisateur. Voir GEN40, GEN41 et GEN42. La longueur de la table peut être différente d'une puissance de 2.

*kmin* -- limite inférieure de l'intervalle

*kmax* -- limite supérieure de l'intervalle

*cuserrnd* (Continuous USER-defined-distribution RaNDom generator) génère des nombres aléatoires selon une distribution aléatoire continue créée par l'utilisateur. Dans ce cas la forme de l'histogramme de la distribution peut être dessinée ou générée par n'importe quelle routine GEN. La table contenant la forme d'un tel histogramme doit être transformée ensuite en une fonction de distribution au moyen de GEN40 (voir GEN40 pour plus de détails). Cette fonction doit ensuite être allouée à l'argument *XtableNum* de *cuserrnd*. L'intervalle de sortie sera ensuite mis à l'échelle selon les arguments *Xmin* et *Xmax*. *cuserrnd* faisant une interpolation linéaire entre les éléments de la table, il n'est pas recommandé pour les distributions discrètes (GEN41 et GEN42).

Pour un tutoriel sur les histogrammes et les fonctions de distribution aléatoires consulter :

- D. Lorrain. "A panoply of stochastic cannons". In C. Roads, ed. 1989. Music machine. Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode *cuserrnd*. Il utilise le fichier *cuserrnd.csd* [examples/cuserrnd.csd].

### Exemple 183. Exemple de l'opcode `cuserrrnd`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o cuserrrnd.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; every run time same values

kuser cuserrrnd 0, 100, 1
  printk .2, kuser
asig poscil .5, 220+kuser, 3
  outs asig, asig
endin

instr 2 ; every run time different values

  seed 0
kuser cuserrrnd 0, 100, 1
  printk .2, kuser
asig poscil .5, 220+kuser, 3
  outs asig, asig
endin
</CsInstruments>
<CsScore>
f 1 0 16 -7 1 4 0 8 0 4 1 ;distrubution using GEN07
f 2 0 16384 40 1 ;GEN40 is to be used with cuserrrnd
f 3 0 8192 10 1 ;sine

i 1 0 2
i 2 3 2
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
i 1 1 time 0.00067: 53.14918
i 1 1 time 0.20067: 0.00000
i 1 1 time 0.40067: 0.00000
i 1 1 time 0.60067: 96.80406
i 1 1 time 0.80067: 94.20729
i 1 1 time 1.00000: 0.00000
i 1 1 time 1.20067: 86.13032
i 1 1 time 1.40067: 31.37096
i 1 1 time 1.60067: 70.35889
i 1 1 time 1.80000: 0.00000
i 1 1 time 2.00000: 49.18914
```

WARNING: Seeding from current time 2006647442

i	2	time	3.00067:	21.45002
i	2	time	3.20067:	44.32333
i	2	time	3.40067:	46.05420
i	2	time	3.60000:	0.00000
i	2	time	3.80067:	41.32175
i	2	time	4.00000:	0.00000
i	2	time	4.20000:	63.72019
i	2	time	4.40067:	0.00000
i	2	time	4.60067:	0.00000
i	2	time	4.80067:	0.00000
i	2	time	5.00000:	74.49330

## Voir aussi

*dusernd, urd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.16

# dam

dam — Un compresseur/expander dynamique.

## Description

Cet opcode modifie dynamiquement une valeur de gain appliquée à l'entrée *ain* en comparant son niveau de puissance à un seuil de niveau donné. Le signal est compressé ou élargi de différents facteurs selon qu'il est au-dessus ou en-dessous du seuil.

## Syntaxe

```
ares dam asig, kthreshold, icompl, icompl2, irtime, iftime
```

## Initialisation

*icompl1* -- rapport de compression pour la zone supérieure.

*icompl2* -- rapport de compression pour la zone inférieure

*irtime* -- montée du gain en secondes. Durée au-delà de laquelle le facteur de gain peut augmenter d'une unité.

*iftime* -- chute du gain en secondes. Durée au-delà de laquelle le facteur de gain peut diminuer d'une unité. of one unit.

## Exécution

*asig* -- signal d'entrée à modifier

*kthreshold* -- niveau du signal d'entrée qui agit comme seuil. Il peut changer au taux-k (par exemple pour le ducking)

Note sur les taux de compression : un taux de compression de un laisse le son inchangé. Avec un rapport inférieur à un, le signal sera compressé (réduction de son volume) tandis qu'avec un rapport supérieur à un, le signal sera élargi (augmentation de son volume).

## Exemples

Les résultats de l'opcode *dam* pouvant être subtils, je recommande de les regarder dans un éditeur de sons graphique comme *audacity*. *audacity* existe pour Linux, Windows, et MacOS et on peut le télécharger à <http://audacity.sourceforge.net> [<http://audacity.sourceforge.net/>].

Voici un exemple de l'opcode dam. Il utilise les fichiers *dam.csd* [examples/dam.csd] et *beats.wav* [examples/beats.wav].

### Exemple 184. Un exemple de l'opcode dam compressant un signal audio.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o dam.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;normal audio

asig disk2 "beats.wav", 1, 0, 1
outs asig, asig

endin

instr 2 ; compressed audio

kthreshold = 0.2
icompl = 0.8
icomp2 = 0.2
irtime = 0.01
iftime = 0.5
asig disk2 "beats.wav", 1, 0, 1
asig dam asig, kthreshold, icompl, icomp2, irtime, iftime
outs asig, asig

endin

</CsInstruments>
<CsScore>

i 1 0 2
i 2 2.5 8.5

e
</CsScore>
</CsoundSynthesizer>

```

Cet exemple compresse le fichier audio « beats.wav ». On y entend un schéma de batterie répété deux fois. La deuxième fois, le son est moins fort (compressé) que la première fois.

Voici un autre exemple de l'opcode dam. Il utilise les fichiers *dam\_expanded.csd* [examples/dam\_expanded.csd] et *beats.wav* [examples/beats.wav].

### Exemple 185. Un exemple de l'opcode dam élargissant un signal audio.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o dam_expanded.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100

```



```
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

asig diskin2 "beats.wav", 1, 0, 1
outs asig, asig

endin

instr 2 ;expanded audio

kthreshold = .5
icomp1 = 2
icomp2 = 3
irtime = 0.01
ifetime = 0.1

asig diskin2 "beats.wav", 1, 0, 1
asig dam asig, kthreshold, icomp1, icomp2, irtime, iftime
outs asig*.2, asig*.2 ;adjust volume of expanded beat

endin

</CsInstruments>
<CsScore>

i 1 0 2
i 2 2.5 6.5

e
</CsScore>
</CsoundSynthesizer>
```

Cet exemple élargit le fichier audio « beats.wav ». On y entend un motif de batterie répété deux fois. La deuxième fois, le son est plus fort (élargi) que la première fois. Pour éviter une distorsion le volume du signal a été diminué.

## Voir aussi

*compress*

## Crédits

Auteur : Marc Resibois  
Belgique  
1997

Nouveau dans la version 3.47

# date

date — Retourne le nombre de secondes écoulées depuis une date de base.

## Description

Opcode du greffon cs\_date.

Retourne le nombre de secondes écoulées depuis une date de base, en lisant l'horloge du système d'exploitation. La base est le 1<sup>er</sup> janvier 1970 pour la version de Csound utilisant des nombres flottants en double précision et le 1<sup>er</sup> janvier 2010 pour les versions utilisant des nombres flottants en simple précision. Sur les systèmes d'exploitation ayant une résolution suffisante, la date comprend les fractions de secondes.

## Syntaxe

```
ir[ , inano] date
```

```
kr[ , knano] date
```

## Initialisation et exécution

*ir* -- valeur en secondes à l'initialisation de la note, de l'horloge système depuis le début de l'epoch.

*kr* -- valeur en secondes durant une période de contrôle, de l'horloge système depuis le début de l'epoch.

*inano* -- valeur en nanosecondes à l'initialisation depuis la dernière seconde.

*knano* -- valeur en nanosecondes durant une période de contrôle depuis la dernière seconde.

Noter que la date de base était à l'origine 1970, mais depuis la version 5.14 elle a été changée pour les nombres flottants en simple précision car ceux-ci sont insuffisants pour indiquer les changements.

La réponse facultative avec une résolution comprenant les fractions de seconde a été introduite dans la version 6.07.

## Exemples

Voici un exemple de l'opcode date. Il utilise le fichier *date.csd* [examples/date.csd].

### Exemple 186. Exemple de l'opcode date.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o date.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
instr 1
  ii,ij date
  print ii
```

```

    print ij
    Sa dates ii
    prints Sa
    Ss dates -1
    prints Ss
    St dates 1
    prints St
endin

</CsInstruments>

<CsScore>
i 1 0 1
e
</CsScore>

</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

instr 1:  ii = 1447337826.778
instr 1:  ij = 778279830.000
Thu Nov 12 14:17:07 2015
Thu Nov 12 14:17:06 2015
Thu Jan  1 01:00:01 1970

```

## Voir aussi

*dates*

## Crédits

Auteur : John ffitch  
 Université de Bath/Codemist Ltd.  
 Bath, UK  
 Décembre 2006

Nouveau dans la version 5.05 de Csound.

Modifié dans la version 5.14 de Csound.

# dates

dates — Retourne sous forme de chaîne de caractères la date et l'heure spécifiées.

## Description

Opcodes du greffon cs\_date.

Retourne sous forme de chaîne de caractères la date et l'heure spécifiées.

## Syntaxe

```
Sir dates [ itime]
```

## Initialisation

*itime* -- le temps écoulé en secondes depuis le début de l'epoch. S'il est omis ou s'il est négatif, le temps courant est utilisé.

*Sir* -- la date et l'heure sous forme de chaîne de caractères.

## Exemples

Voici un exemple de l'opcode dates. Il utilise le fichier *dates.csd* [examples/dates.csd].

### Exemple 187. Exemple de l'opcode dates.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o dates.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kmps = 32
nchnls = 2
0dbfs = 1

seed      0 ;each time different seed

instr 1
;;generating a different filename each time csound renders
itim      date
Stim      dates      itim
Syear     strsub     Stim, 20, 24
Smonth    strsub     Stim, 4, 7
Sday      strsub     Stim, 8, 10
iday      strtod     Sday
Shor      strsub     Stim, 11, 13
Smin      strsub     Stim, 14, 16
Ssec      strsub     Stim, 17, 19
```

```

Sfilnam sprintf "%s_%s_%02d_%s_%s_%s.wav", Syear, Smonth, iday, Shor, Smin, Ssec
;;rendering with random frequency, amp and pan, and writing to disk
ifreq random 400, 1000
iamp random .1, 1
ipan random 0, 1
asin oscils iamp, ifreq, 0
aL, aR pan2 asin, ipan
fout Sfilnam, 14, aL, aR
outs aL, aR
printf_i "File '%s' written to the same directory as this CSD file is!\n", 1, Sfilnam

endin

</CsInstruments>
<CsScore>
i 1 0 1
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

File '2011_Jan_05_19_14_46.wav' written to the same directory as this CSD file is!
Closing file '/home/user/csound/Output/2011_Jan_05_19_14_46.wav'...

```

## Voir aussi

*date*

## Crédits

Auteur : John ffitch  
 Université de Bath/Codemist Ltd.  
 Bath, UK  
 Décembre 2006

Nouveau dans la version 5.05 de Csound

# db

db — Retourne l'amplitude équivalente pour une valeur donnée en décibels.

## Description

Retourne l'amplitude équivalente pour une valeur donnée en décibels. Cet opcode est le même que *ampdb*.

## Syntaxe

`db(x)`

Cette fonction fonctionne aux taux-i, -k et -a.

## Initialisation

*x* -- une valeur exprimée en décibels.

## Exécution

Retourne l'amplitude équivalente pour une valeur donnée en décibels.

## Exemples

Voici un exemple de l'opcode db. Il utilise le fichier *db.csd* [examples/db.csd].

### Exemple 188. Exemple de l'opcode db.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o db.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1

idec = p4
iamp = db(idec)
print iamp
asig vco2 iamp, 110 ;sawtooth
outs asig, asig

endin
```

```
</CsInstruments>
<CsScore>

i 1 0 1 50
i 1 + 1 >
i 1 + 1 >
i 1 + 1 85
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra ces lignes :

```
instr 1: iamp = 316.252
      instr 1: iamp = 1211.582
      instr 1: iamp = 4641.643
      instr 1: iamp = 17782.420
```

## Voir aussi

*ampdb, cent, octave, semitone*

Nouveau dans la version 4.16

# dbamp

dbamp — Retourne l'équivalent en décibel de l'amplitude  $x$ .

## Description

Retourne l'équivalent en décibel de l'amplitude  $x$ .

## Syntaxe

**dbamp**( $x$ ) (arguments de taux- $i$  ou  $-k$  seulement)

## Exemples

Voici un exemple de l'opcode dbamp. Il utilise le fichier *dbamp.csd* [examples/dbamp.csd].

### Exemple 189. Exemple de l'opcode dbamp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o dbamp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1

iamp = p4
idb = dbamp(iamp)
print idb
asig vco2 iamp, 110 ;sawtooth
outs asig, asig

endin

</CsInstruments>
<CsScore>

i 1 0 1 100
i 1 + 1 1000
i 1 + 1 10000
i 1 + 1 20000
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra ces lignes :



```
instr 1: idb = 40.000
instr 1: idb = 60
instr 1: idb = 80
instr 1: idb = 86.021
```

## Voir aussi

*ampdb, ampdbfs, dbfsamp*

# dbfsamp

dbfsamp — Retourne l'équivalent en décibel de l'amplitude  $x$ , relative à l'amplitude maximale.

## Description

Retourne l'équivalent en décibel de l'amplitude  $x$ , relative à l'amplitude maximale. L'amplitude maximale est supposée être codée en 16 bit. Nouveau dans la version 4.10.

## Syntaxe

**dbfsamp**( $x$ ) (arguments de `taux-i` ou `-k` seulement)

## Exemples

Voici un exemple de l'opcode dbfsamp. Il utilise le fichier *dbfsamp.csd* [examples/dbfsamp.csd].

### Exemple 190. Exemple de l'opcode dbfsamp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o dbfsamp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1

iamp = p4
idb = dbfsamp(iamp)
print idb
asig vco2 iamp, 110 ;sawtooth
outs asig, asig

endin

</CsInstruments>
<CsScore>

i 1 0 1 1
i 1 + 1 100
i 1 + 1 1000
i 1 + 1 10000
i 1 + 1 30000
e

</CsScore>
```

</CsoundSynthesizer>

Sa sortie contiendra ces lignes :

```
instr 1: idb = -90.309
      instr 1: idb = -50.309
      instr 1: idb = -30.309
      instr 1: idb = -10.309
      instr 1: idb = -0.767
```

## Voir aussi

*ampdb, ampdbfs, dbamp*

# dcblock

dcblock — Un filtre bloqueur de composante continue.

## Description

Implémente le filtre bloqueur de composante continue

$$Y[i] = X[i] - X[i-1] + (\text{igain} * Y[i-1])$$

Basé sur un travail de Perry Cook.

## Syntaxe

```
ares dcblock ain [, igain]
```

## Initialisation

*igain* -- le gain du filtre qui vaut 0.99 par défaut.

## Exécution

*ain* -- signal audio en entrée.



### Note

Le nouvel opcode *dcblock2* présente une méthode améliorée de suppression de la composante continue d'un signal audio.

## Exemples

On peut voir le résultat dans un éditeur graphique de fichiers audio comme *audacity*. *audacity* est disponible pour Linux, Windows et Mac OS et on peut le télécharger depuis <http://audacity.sourceforge.net> [<http://audacity.sourceforge.net/>].

Voici un exemple de l'opcode dcblock. Il utilise les fichiers *dcblock.csd* [examples/dcblock.csd] et *beats.wav* [examples/beats.wav].

### Exemple 191. Exemple de l'opcode dcblock.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o dcblock.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1 ;add DC to "beats.wav"

asig soundin "beats.wav"
asig = asig+5000 ;adds DC of 5000
outs asig, asig
endin

instr 2 ;dcblock audio

asig soundin "beats.wav"
asig = asig+5000 ;adds DC
adc dcblock asig ;remove DC again
outs adc, adc

endin

</CsInstruments>
<CsScore>

i 1 0 2
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*dcblock2*

## Crédits

Auteur : John ffitch  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 3.49 de Csound.

Février 2003 : la formule a été corrigée grâce à une note d'Anders Andersson.

# dcblock2

dcblock2 — Un filtre bloqueur de composante continue.

## Description

Implémente un filtre bloqueur de composante continue avec une atténuation améliorée de la composante continue.

## Syntaxe

```
ares dcblock2 ain [, iorder] [, iskip]
```

## Initialisation

*iorder* -- ordre du filtre, au minimum 4ème ordre, vaut par défaut 128.

*iskip* -- s'il vaut 1, l'initialisation est ignorée (0 par défaut).

## Exécution

*ares* -- signal audio filtré

*ain* -- signal audio en entrée



### Note

Avec l'utilisation d'une valeur inférieure à *ksmps* pour *iorder*, la réduction du décalage dû à la composante continue ne sera pas efficace.

## Exemples

On peut voir le résultat dans un éditeur graphique de fichiers audio comme *audacity*. *audacity* est disponible pour Linux, Windows et Mac OS et on peut le télécharger depuis <http://audacity.sourceforge.net> [<http://audacity.sourceforge.net/>].

Voici un exemple de l'opcode dcblock2. Il utilise les fichiers *dcblock2.csd* [examples/dcblock2.csd] et *beats.wav* [examples/beats.wav].

### Exemple 192. Exemple de l'opcode dcblock2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o dcblock2.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1 ;add DC to "beats.wav"

asig soundin "beats.wav"
asig = asig+5000 ;adds DC of 5000
outs asig, asig
endin

instr 2 ;dcblock audio

asig soundin "beats.wav"
asig = asig+5000 ;adds DC
adc dcblock2 asig ;remove DC again
outs adc, adc

endin

</CsInstruments>
<CsScore>

i 1 0 2
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*dcblock*

## Crédits

Par Victor Lazzarini

Nouveau dans la version 5.09 de Csound

# dconv

dconv — Un opcode de convolution directe.

## Description

Un opcode de convolution directe.

## Syntaxe

```
ares dconv asig, isize, ifn
```

## Initialisation

*isize* -- la taille du tampon de convolution à utiliser. Si la taille du tampon est inférieure à celle de *ifn*, alors seules les premières *isize* valeurs de la table seront utilisées.

*ifn* -- numéro de la table d'une fonction stockée contenant la réponse impulsionnelle pour la convolution.

## Exécution

Plutôt que d'utiliser la méthode d'analyse/resynthèse de l'opcode *convolve*, *dconv* utilise la convolution directe pour créer le résultat. Avec de petites tables, il peut faire cela avec une certaine efficacité, alors que des tables plus grandes nécessitent bien plus de temps de calcul. *dconv* effectue (*isize* \* *ksmps*) multiplications à chaque cycle-k. C'est pourquoi les effets de réverbération et de délai sont mieux réalisés par d'autre opcodes (à moins que les durées soient courtes).

*dconv* a été conçu pour travailler avec des tables variant dans le temps pour faciliter de nouvelles possibilités de filtrage en temps réel.

## Exemples

Voici un exemple de l'opcode *dconv*. Il utilise le fichier *dconv.csd* [examples/dconv.csd].

### Exemple 193. Exemple de l'opcode dconv.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o dconv.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```



```

#define RANDI(A) #kout randi 1, kfq, $A*.001+iseed, 1
                tablew kout, $A, itable#

instr 1
itable init 1
iseed init .6
isize init ftlen(itable)
kfq line 1, p3, 10

$RANDI(0)
$RANDI(1)
$RANDI(2)
$RANDI(3)
$RANDI(4)
$RANDI(5)
$RANDI(6)
$RANDI(7)
$RANDI(8)
$RANDI(9)
$RANDI(10)
$RANDI(11)
$RANDI(12)
$RANDI(13)
$RANDI(14)
$RANDI(15)

asig rand 10000, .5, 1
asig butlp asig, 5000
asig dconv asig, isize, itable

out asig *.5
endin

</CsInstruments>
<CsScore>

f1 0 16 10 1
i1 0 10
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*pconvolve, convolve, fconv*

## Crédits

Auteur : William « Pete » Moss  
2001

Nouveau dans la version 4.12

# dct

dct — Transformée en cosinus discrète d'un tableau d'échantillons (DCT-II).

## Description

Applique une transformée en cosinus discrète à un tableau unidimensionnel produisant un tableau de même taille qui contient la transformée. Pour le moment seules les tailles puissance de deux sont implémentées. Disponible en versions de taux-i et de taux-k.

## Syntaxe

```
kout[] dct kin[]
```

```
iout[] dct iin[]
```

## Initialisation

*iout[]* -- tableau de sortie contenant la DCT. Il est créé s'il n'existait pas.

*iin[]* -- tableau d'entrée.

## Exécution

*kout[]* -- tableau de sortie contenant la DCT. Il est créé s'il n'existait pas.

*kin[]* -- tableau d'entrée.

## Exemples

Voici un exemple de l'opcode `dct`. Il utilise le fichier *dct.csd* [examples/dct.csd].

### Exemple 194. Exemple de l'opcode `dct`.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

instr 1

iArr[] fillarray 1,2,3,4
iDCT[] dct iArr
iDCT[] dctinv iDCT

prints "%.1f %.1f %.1f %.1f => %.1f %.1f %.1f %.1f\n",
      iArr[0], iArr[1], iArr[2], iArr[3],
      iDCT[0], iDCT[1], iDCT[2], iDCT[3]

endin
</CsInstruments>
```

```
<CsScore>  
i1 0 0  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux, *dtinv*.

## Crédits

Auteur : Victor Lazzarini  
NUI Maynooth  
2014

Nouveau dans la version 6.04

# dctinv

dctinv — Transformée en cosinus discrète inverse d'un tableau d'échantillons (DCT-III).

## Description

Applique une transformée en cosinus discrète inverse à un tableau unidimensionnel produisant un tableau de même taille qui contient la transformée. Pour le moment seules les tailles puissance de deux sont implémentées. Disponible en versions de taux-i et de taux-k.

## Syntaxe

```
kout[] dctinv kin[]  
iout[] dctinv iin[]
```

## Initialisation

*iout[]* -- tableau de sortie contenant la IDCT. Il est créé s'il n'existait pas.

*iin[]* -- tableau d'entrée.

## Exécution

*kout[]* -- tableau de sortie contenant la IDCT. Il est créé s'il n'existait pas.

*kin[]* -- tableau d'entrée.

## Exemples

Voici un exemple de l'opcode `dctinv`. Il utilise le fichier *dctinv.csd* [examples/dctinv.csd].

### Exemple 195. Exemple de l'opcode `dctinv`.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
</CsOptions>  
<CsInstruments>  
  
instr 1  
  
iArr[] fillarray 1,2,3,4  
iDCT[] dct iArr  
iiDCT[] dctinv iDCT  
  
prints "%.1f %.1f %.1f %.1f => %.1f %.1f %.1f %.1f\n",  
        iArr[0], iArr[1], iArr[2], iArr[3],  
        iiDCT[0], iiDCT[1], iiDCT[2], iiDCT[3]  
  
endin  
</CsInstruments>
```

```
<CsScore>  
i1 0 0  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux, *dct*

## Crédits

Auteur : Victor Lazzarini  
NUI Maynooth  
2014

Nouveau dans la version 6.04

# deinterleave

deinterleave — Désentrelace un tableau en prenant alternativement les données de son entrée.

## Description

Prend un tableau en entrée et désentrelace ses données en choisissant les données depuis les positions alternées.

## Syntaxe

```
kout1[], kout2[] deinterleave kin[]
```

## Exécution

*kout1[], kout2[]* -- tableaux en sortie contenant les données désentrelacées. Ils sont créés s'ils n'existent pas.

*kin[]* -- tableau en entrée contenant les valeurs à désentrelacer.

## Exemples

Voici un exemple de l'opcode deinterleave. Il utilise le fichier *deinterleave.csd* [exemples/deinterleave.csd].

### Exemple 196. Exemple de l'opcode deinterleave.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-n
</CsOptions>
<CsInstruments>

instr 1

kInt[] fillarray 1,2,3,4,5,6,7,8

kout1[],kout2[] deinterleave kInt

printf "input: \n%d %d %d %d %d %d %d %d\n", 1,
      kInt[0], kInt[1], kInt[2], kInt[3],
      kInt[4], kInt[5], kInt[6], kInt[7]

printf "de-interleaved:\n%d %d %d %d \n%d %d %d %d\n", 1,
      kout1[0], kout1[1], kout1[2], kout1[3],
      kout2[0], kout2[1], kout2[2], kout2[3]

endin

</CsInstruments>
<CsScore>
i1 0 1
```

```
e  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

Vectorial opcodes, array opcodes

## Crédits

Auteur : Victor Lazzarini  
NUI Maynooth  
2018

Nouveau dans la version 6.12

# delay

delay — Retarde un signal d'entrée d'une certaine durée.

## Description

Un signal peut être lu ou écrit dans une ligne à retard, ou il peut être retardé automatiquement d'une certaine durée.

## Syntaxe

```
ares delay asig, idlt [, iskip]
```

## Initialisation

*idlt* -- délai demandé en secondes. Il peut être aussi grand que la mémoire disponible le permet. L'espace requis pour *n* secondes de délai est de  $4n * sr$  octets. Il est alloué lorsque l'instrument est initialisé pour la première fois, et retourne dans le pool à la fin d'une section de partition.

*iskip* (facultatif, 0 par défaut) -- disposition initiale de l'espace des données de la boucle de retard (voir *reson*). La valeur par défaut est 0.

## Exécution

*asig* -- signal audio

*delay* est un composé de *delayr* et de *delayw*, écrivant et lisant à la fois dans son propre espace de stockage. Il peut ainsi accomplir un décalage temporel du signal, bien que la rétroaction variable ne soit pas possible. Il n'y a pas de durée de délai minimale.

## Exemples

Voici un exemple de l'opcode delay. Il utilise le fichier *delay.csd* [examples/delay.csd].

### Exemple 197. Exemple de l'opcode delay.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o delay.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 32
nchnls  = 2
0dbfs   = 1
```



```

instr      1

adel init 0
ilev      = p4      ;level of direct sound
idelay    = p5 *.001 ;Delay in ms
ifd = p6      ;feedback

ain diskin2 "fox.wav", 1, 1
adel delay ain + (adel*ifd), idelay;ifd = amount of feedback
asig moogvcf adel, 1500, .6, 1 ;color feedback
outs      asig*ilev, ain

endin

</CsInstruments>
<CsScore>
;Delay is in ms
i 1 0 15 2 200 .95 ;with feedback
i 1 4 5 2 20 .95
i 1 + 3 2 5 .95
i 1 + 3 3 5 0 ;no feedback

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*delayl, delayr, delayw*

# delay1

delay1 — Retarde un signal d'entrée d'un échantillon.

## Description

Retarde un signal d'entrée d'un échantillon.

## Syntaxe

```
ares delay1 asig [, iskip]
```

## Initialisation

*iskip* (facultatif, 0 par défaut) -- disposition initiale de l'espace des données de la boucle de retard (voir *reson*). La valeur par défaut est 0.

## Exécution

*delay1* est une forme spéciale de délai qui sert à retarder le signal audio *asig* d'un seul échantillon. Il est ainsi fonctionnellement équivalent à l'opcode *delay* mais il est plus efficace à la fois en temps et en espace. Cette unité est particulièrement utile dans la fabrication de filtres non récursifs généralisés.

## Exemples

Voici un exemple des opcodes *delay* et *delay1*. Il utilise le fichier *delay1.csd* [exemples/delay1.csd].

### Exemple 198. Exemple de l'opcode *delay1*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; For Non-realtime ouput leave only the line below:
-o delay.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
odbfs = 1

instr 1
; Make white noise.
a0 random -1, 1

; Simple Lowpass filter
a1 delay1 a0
aout = (a0+0.99*a1)/2
```

```
    ; output white and filtered
    outs      aout, a0
endin

</CsInstruments>
<CsScore>
; Play Instrument #1.
i 1 0.0 3

e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*delay, delayr, delayw*

## Crédits

Auteur : Barry Vercoe

Exemple écrit par John ffitich.

# delayk

delayk — Retarde un signal d'entrée d'une certaine durée.

## Description

Opcodes de retard de taux-k.

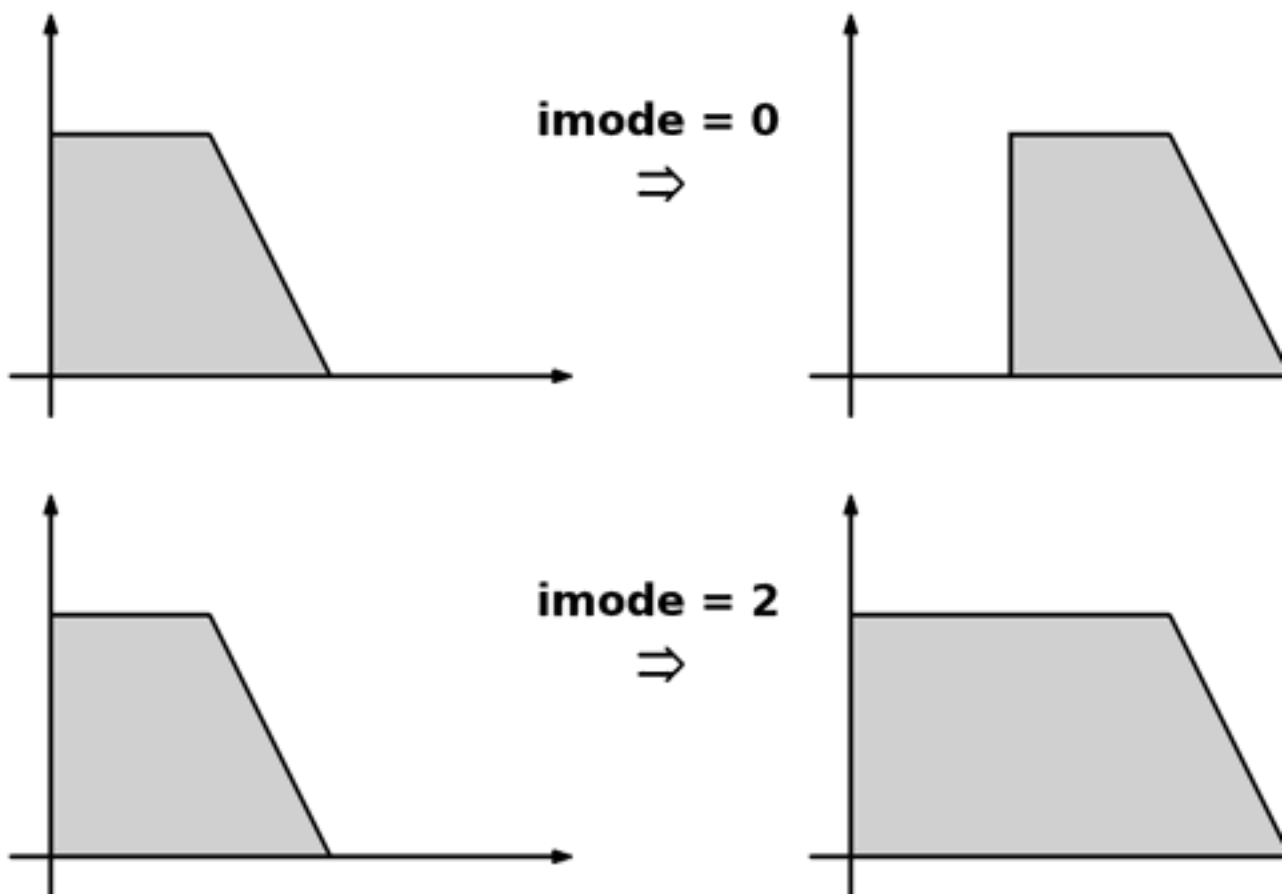
## Syntaxe

```
kr delayk    ksig, idel[, imode]
kr vdel_k    ksig, kdel, imdel[, imode]
```

## Initialisation

*idel* -- délai (en secondes) pour *delayk*. Il est arrondi au multiple entier le plus proche d'un cycle-k (c-à-d  $1/kr$ ).

*imode* -- somme de 1 pour ignorer l'initialisation (par exemple pour les notes liées) et de 2 pour maintenir la première valeur d'entrée durant le délai initial au lieu de sortir des zéros. Cela est utile principalement pour retarder des enveloppes qui ne commencent pas à zéro.



*imdel* -- délai maximum pour *vdel\_k*, en secondes.

## Exécution

*kr* -- le signal de sortie. Note : ces opcodes n'interpolent pas leur sortie.

*ksig* -- le signal d'entrée.

*kdel* -- délai (en secondes) pour *vdel\_k*. Il est arrondi au multiple entier le plus proche d'un cycle-k (c-à-d  $1/kr$ ).

## Exemples

Voici un exemple de l'opcode delayk. Il utilise le fichier *delayk.csd* [examples/delayk.csd].

### Exemple 199. Exemple de l'opcode delayk.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o delayk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
;example shows "delayk" for fm index and
;a second "delayk" for panning
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisin ftgen 0, 0, 2^10, 10, 1

instr 1

kenv1 transeg 0, .02, 0, 1, 3.98, -6, 0 ;envelope
kenv2 delayk kenv1, 2 ;delayed by two seconds
kindx expon 5, p3, 1 ;fm index decreasing over p3
asig foscili .6, 400, 1, 11/4, kindx, gisin
kpan1 linseg 0, 4, 1 ;panning for first sound
kpan2 linseg 1, 4, 0 ;panning for second sound ...
kpan2 delayk kpan2, 2 ;delayed by two seconds
a1 = asig * kenv1
a2 = asig * kenv2

aL1,aR1 pan2 a1, kpan1
aL2,aR2 pan2 a2, kpan2
outs aL1+aL2, aR1+aR2

endin
</CsInstruments>
<CsScore>

i 1 0 6
e
</CsScore>
```

`</CsoundSynthesizer>`

## Crédits

Istvan Varga.

# delayr

delayr — Lit depuis une ligne à retard numérique établie automatiquement.

## Description

Lit depuis une ligne à retard numérique établie automatiquement.

## Syntaxe

```
ares delayr idlt [, iskip]
```

## Initialisation

*idlt* -- délai demandé en secondes. Il peut être aussi grand que la mémoire disponible le permet. L'espace requis pour n secondes de délai est de  $4n * sr$  octets. Il est alloué lorsque l'instrument est initialisé pour la première fois, et retourne dans le pool à la fin d'une section de partition.

*iskip* (facultatif, 0 par défaut) -- disposition initiale de l'espace des données de la boucle de retard (voir *reson*). La valeur par défaut est 0.

## Exécution

*delayr* lit depuis une ligne à retard numérique établie automatiquement, dans laquelle le signal restitué est resté pendant *idlt* secondes. Cette unité doit être appariée avec une unité *delayw* qu'elle précède. Il peut y avoir d'autres opcodes de Csound entre les deux.

## Exemples

Voici un exemple de l'opcode delayr. Il utilise le fichier *delayr.csd* [examples/delayr.csd].

### Exemple 200. Exemple de l'opcode delayr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime output leave only the line below:
; -o delayr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gasig  init 0
gidel  = 1 ;delay time in seconds
```

```

instr 1

ain  pluck .7, 440, 1000, 0, 1
    outs ain, ain

vincr gasig, ain ;send to global delay
endin

instr 2

ifedback = p4

abuf2 delayr gidel
adelL  deltap .4 ;first tap (on left channel)
adelM  deltap 1 ;second tap (on middle channel)
    delayw gasig + (adelL * ifedback)

abuf3 delayr gidel
kdel  line 1, p3, .01 ;vary delay time
adelR  deltap .65 * kdel ;one pitch changing tap (on the right chn.)
    delayw gasig + (adelR * ifedback)
;make a mix of all deayed signals
    outs adelL + adelM, adelR + adelM

clear gasig
endin

</CsInstruments>
<CsScore>

i 1 0 1
i 1 3 1
i 2 0 3 0 ;no feedback
i 2 3 8 .8 ;lots of feedback
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*delay, delayl, delayw*



# delayw

delayw — Écrit le signal audio dans une ligne à retard numérique.

## Description

Écrit le signal audio dans une ligne à retard numérique.

## Syntaxe

```
delayw asig
```

## Exécution

*delayw* écrit *asig* dans l'espace du délai établi par l'unité *delayr* précédente. Cette paire d'unités permet la formation de boucles de rétroaction modifiées, etc. Cependant, il y a une limite inférieure à *idlt*, qui doit valoir au moins une période de contrôle (ou  $1/kr$ ).

## Exemples

Voici un exemple de l'opcode *delayw*. Il utilise le fichier *delayw.csd* [examples/delayw.csd].

### Exemple 201. Exemple de l'opcode *delayw*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o delayw.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gasig  init 0
gidel  = 1 ;delay time in seconds

instr 1

ain  pluck .7, 440, 1000, 0, 1
     outs ain, ain

vincr gasig, ain ;send to global delay
endin

instr 2

ifedback = p4
```

```

abuf2 delayr gidel
adelL deltap .4 ;first tap (on left channel)
adelM deltap 1 ;second tap (on middle channel)
delayw gasig + (adelL * ifeedback)

abuf3 delayr gidel
kdel line 1, p3, .01 ;vary delay time
adelR deltap .65 * kdel ;one pitch changing tap (on the right chn.)
delayw gasig + (adelR * ifeedback)
;make a mix of all deayed signals
outs adelL + adelM, adelR + adelM

clear gasig
endin

</CsInstruments>
<CsScore>

i 1 0 1
i 1 3 1
i 2 0 3 0 ;no feedback
i 2 3 8 .8 ;lots of feedback
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*delay, delayl, delayr*

# deltap

deltap — Lit une ligne à retard avec des délais variables.

## Description

Lit une ligne à retard avec des délais variables.

## Syntaxe

```
ares deltap kdlt
```

## Exécution

*kdlt* -- spécifie le délai de lecture en secondes. Chaque valeur est comprise entre 1 période de contrôle et le délai total de la paire lecture/écriture ; cependant, comme il n'y a pas de vérification interne du respect de cet intervalle, l'utilisateur est entièrement responsable. Chaque argument peut être une constante, une variable ou un signal variant dans le temps.

*deltap* extrait le son en lisant directement les échantillons stockés.

Cet opcode peut lire directement dans une paire *delayr/delayw* en extrayant les données audio retardées des *idlt* secondes de son stocké. Il peut y avoir n'importe quel nombre d'unités *deltap* et/ou *deltapi* entre une paire lecture/écriture. Chacune reçoit un extrait audio sans changement de l'amplitude originale.

Ces opcodes peuvent fournir de multiples lectures de délai pour des lignes à retard arbitraires et des réseaux à rétroaction. Ils peuvent délivrer des délais constants ou variables, et sont utiles pour construire des effets de chorus, harmonizer et Doppler. Les délais constants (et certains de ceux qui varient lentement) ne nécessitent pas d'interpolation ; *deltap* leur convient très bien. Les délais variant à moyenne vitesse ou rapidement nécessiteront cependant les services supplémentaires de *deltapi*.

Les paires *delayr/delayw* peuvent être entrelacées. Pour associer une unité de lecture de délai à une unité *delayr*, elle doit non seulement être située entre ce *delayr* et le *delayw* approprié, mais aussi précéder tous les *delayr* suivants. Voir l'exemple 2. (Cette possibilité fut ajoutée dans la version 3.57 de Csound par Jens Groh et John fitch).

*N.B.* Les délais de taux-k ne sont pas interpolés en interne, mais déroulent plutôt des décalages temporels d'échantillons audios ; c'est adéquat pour des délais changeant lentement. Cependant, pour les changements à vitesse moyenne ou rapides, il faut fournir en entrée des valeurs de délai avec une plus grande résolution de taux audio.

## Exemples

### Exemple 202. Exemple n°1 de deltap

```
asource  buzz      1, 440, 20, 1
atime    linseg     1, p3/2,.01, p3/2,1 ; trace a distance in secs
ampfac   =          1/atime/atime       ; and calc an amp factor
adump    delayr     1                   ; set maximum distance
amove    deltapi    atime                ; move sound source past
         delayw     asource              ; the listener
out      amove * ampfac
```

**Exemple 203. Exemple n°2 de deltap**

```

ainput1 = .....
ainput2 = .....
kdlyt1  = .....
kdlyt2  = .....

;Read delayed signal, first delayr instance:
adump   delayr 4.0
adly1   deltap kdlyt1      ; associated with first delayr instance

;Read delayed signal, second delayr instance:
adump   delayr 4.0
adly2   deltap kdlyt2      ; associated with second delayr instance

;Do some cross-coupled manipulation:
afdbk1  =      0.7 * adly1 + 0.7 * adly2 + ainput1
afdbk2  =     -0.7 * adly1 + 0.7 * adly2 + ainput2

;Feed back signal, associated with first delayr instance:
delayw  afdbk1

;Feed back signal, associated with second delayr instance:
delayw  afdbk2
outs    adly1, adly2

```

Voici un autre exemple de l'opcode deltap. Il utilise le fichier *deltap.csd* [examples/deltap.csd].

**Exemple 204. Exemple de l'opcode deltap.**

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o deltap.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gasig  init 0
gidel  = 1 ;delay time in seconds

instr 1

ain  pluck .7, 440, 1000, 0, 1
outs ain, ain

vincr gasig, ain ;send to global delay
endin

instr 2

ifedback = p4

abuf2 delayr gidel

```

```
adell deltap .4 ;first tap (on left channel)
adelM deltap 1 ;second tap (on middle channel)
delayw gasig + (adell * ifeedback)

abuf3 delayr gidel
kdel line 1, p3, .01 ;vary delay time
adelR deltap .65 * kdel ;one pitch changing tap (on the right chn.)
delayw gasig + (adelR * ifeedback)
;make a mix of all deayed signals
outs adell + adelM, adelR + adelM

clear gasig
endin

</CsInstruments>
<CsScore>

i 1 0 1
i 1 3 1
i 2 0 3 0 ;no feedback
i 2 3 8 .8 ;lots of feedback
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*deltap3, deltapi, deltapi*

# deltap3

deltap — Lit une ligne à retard avec des délais variables et interpolation cubique.

## Description

Lit une ligne à retard avec des délais variables et interpolation cubique.

## Syntaxe

```
ares deltap3 xdlr
```

## Exécution

*xdlr* -- spécifie le délai de lecture en secondes. Chaque valeur est comprise entre 1 période de contrôle et le délai total de la paire lecture/écriture ; cependant, comme il n'y a pas de vérification interne du respect de cet intervalle, l'utilisateur est entièrement responsable. Chaque argument peut être une constante, une variable ou un signal variant dans le temps ; l'argument *xdlr* de *deltap3* implique qu'une valeur de délai variant au taux audio est autorisée ici.

*deltap3* est expérimental, et utilise l'interpolation cubique. (Nouveau dans la version 3.50 de Csound).

Cet opcode peut lire directement dans une paire *delayr/delayw* en extrayant les données audio retardées des *idlr* secondes de son stocké. Il peut y avoir n'importe quel nombre d'unités *deltap* et/ou *deltapi* entre une paire lecture/écriture. Chacune reçoit un extrait audio sans changement de l'amplitude originale.

Ces opcodes peuvent fournir de multiples lectures de délai pour des lignes à retard arbitraires et des réseaux à rétroaction. Ils peuvent délivrer des délais constants ou variables, et sont utiles pour construire des effets de chorus, harmonizer et Doppler. Les délais constants (et certains de ceux qui varient lentement) ne nécessitent pas d'interpolation ; *deltap* leur convient très bien. Les délais variant à moyenne vitesse ou rapidement nécessiteront cependant les services supplémentaires de *deltapi*.

Les paires *delayr/delayw* peuvent être entrelacées. Pour associer une unité de lecture de délai à une unité *delayr*, elle doit non seulement être située entre ce *delayr* et le *delayw* approprié, mais aussi précéder tous les *delayr* suivants. Voir l'exemple 2. (Cette possibilité fut ajoutée dans la version 3.57 de Csound par Jens Groh et John ffitch).

*N.B.* Les délais de taux-k ne sont pas interpolés en interne, mais déroulent plutôt des décalages temporels d'échantillons audios ; c'est adéquat pour des délais changeant lentement. Cependant, pour les changements à vitesse moyenne ou rapides, il faut fournir en entrée des valeurs de délai avec une plus grande résolution de taux audio.

## Exemples

### Exemple 205. Exemple n°1 de deltap

```
asource  buzz      1, 440, 20, 1
atime    linseg     1, p3/2,.01, p3/2,1 ; trace a distance in secs
ampfac   =          1/atime/atime       ; and calc an amp factor
adump    delayr     1                   ; set maximum distance
amove    deltapi    atime               ; move sound source past
         delayw     asource             ; the listener
```

```
out      amove * ampfac
```

## Exemple 206. Exemple n°2 de deltap

```
ainput1 = .....
ainput2 = .....
kdlyt1  = .....
kdlyt2  = .....

;Read delayed signal, first delayr instance:
adump    delayr 4.0
adly1    deltap kdlyt1      ; associated with first delayr instance

;Read delayed signal, second delayr instance:
adump    delayr 4.0
adly2    deltap kdlyt2      ; associated with second delayr instance

;Do some cross-coupled manipulation:
afdbk1   =      0.7 * adly1 + 0.7 * adly2 + ainput1
afdbk2   =     -0.7 * adly1 + 0.7 * adly2 + ainput2

;Feed back signal, associated with first delayr instance:
delayw   afdbk1

;Feed back signal, associated with second delayr instance:
delayw   afdbk2
outs     adly1, adly2
```

Voici un autre exemple de l'opcode deltap3. Il utilise le fichier *deltap3.csd* [examples/deltap3.csd].

## Exemple 207. Exemple de l'opcode deltap3.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o deltap3.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gasig    init 0
gidel    = 1 ;delay time in seconds

instr 1

ain    pluck .7, 440, 1000, 0, 1
outs  ain, ain

vincr   gasig, ain ;send to global delay
endin

instr 2
```

```

ifedback = p4

abuf2 delayr gidel
adelL deltap3 .4 ;first tap (on left channel)
adelM deltap3 1 ;second tap (on middle channel)
delayw gasig + (adelL * ifedback)

abuf3 delayr gidel
kdel line 1, p3, .01 ;vary delay time
adelR deltap3 .65 * kdel ;one pitch changing tap (on the right chn.)
delayw gasig + (adelR * ifedback)
;make a mix of all deayed signals
outs adelL + adelM, adelR + adelM

clear gasig
endin

</CsInstruments>
<CsScore>

i 1 0 1
i 1 3 1
i 2 0 3 0 ;no feedback
i 2 3 8 .8 ;lots of feedback
e
</CsScore>
</CsSoundSynthesizer>

```

## Voir aussi

*deltap, deltapi, deltapi*



# deltapi

deltapi — Lit une ligne à retard avec des délais variables et interpolation.

## Description

Lit une ligne à retard avec des délais variables et interpolation.

## Syntaxe

```
ares deltapi xdl
```

## Exécution

*xdl* -- spécifie le délai de lecture en secondes. Chaque valeur est comprise entre 1 période de contrôle et le délai total de la paire lecture/écriture ; cependant, comme il n'y a pas de vérification interne du respect de cet intervalle, l'utilisateur est entièrement responsable. Chaque argument peut être une constante, une variable ou un signal variant dans le temps ; l'argument *xdl* de *deltapi* implique qu'une valeur de délai variant au taux audio est autorisée ici.

*deltapi* extrait le son par lecture interpolée. En interpolant entre deux échantillons voisins stockés *deltapi* restitue un délai particulier avec plus de précision, mais nécessite deux fois plus de temps de calcul.

Cet opcode peut lire directement dans une paire *delayr/delayw* en extrayant les données audio retardées des *idlt* secondes de son stocké. Il peut y avoir n'importe quel nombre d'unités *deltap* et/ou *deltapi* entre une paire lecture/écriture. Chacune reçoit un extrait audio sans changement de l'amplitude originale.

Ces opcodes peuvent fournir de multiples lectures de délai pour des lignes à retard arbitraires et des réseaux à rétroaction. Ils peuvent délivrer des délais constants ou variables, et sont utiles pour construire des effets de chorus, harmoniser et Doppler. Les délais constants (et certains de ceux qui varient lentement) ne nécessitent pas d'interpolation ; *deltap* leur convient très bien. Les délais variant à moyenne vitesse ou rapidement nécessiteront cependant les services supplémentaires de *deltapi*.

Les paires *delayr/delayw* peuvent être entrelacées. Pour associer une unité de lecture de délai à une unité *delayr*, elle doit non seulement être située entre ce *delayr* et le *delayw* approprié, mais aussi précéder tous les *delayr* suivants. Voir l'exemple 2. (Cette possibilité fut ajoutée dans la version 3.57 de Csound par Jens Groh et John fitch).

*N.B.* Les délais de taux-k ne sont pas interpolés en interne, mais déroulent plutôt des décalages temporels d'échantillons audios ; c'est adéquat pour des délais changeant lentement. Cependant, pour les changements à vitesse moyenne ou rapides, il faut fournir en entrée des valeurs de délai avec une plus grande résolution de taux audio.

## Exemples

### Exemple 208. Exemple n°1 de *deltap*

```
asource  buzz      1, 440, 20, 1
atime    linseg     1, p3/2,.01, p3/2,1 ; trace a distance in secs
ampfac   =          1/atime/atime      ; and calc an amp factor
adump    delayr     1                  ; set maximum distance
```

```

amove    deltapi    atime                ; move sound source past
         delayw     asource              ; the listener
         out        amove * ampfac

```

## Exemple 209. Exemple n°2 de deltapi

```

ainput1 = .....
ainput2 = .....
kdlyt1  = .....
kdlyt2  = .....

;Read delayed signal, first delayr instance:
adump    delayr 4.0
adly1    deltapi kdlyt1          ; associated with first delayr instance

;Read delayed signal, second delayr instance:
adump    delayr 4.0
adly2    deltapi kdlyt2          ; associated with second delayr instance

;Do some cross-coupled manipulation:
afdbk1   =      0.7 * adly1 + 0.7 * adly2 + ainput1
afdbk2   =     -0.7 * adly1 + 0.7 * adly2 + ainput2

;Feed back signal, associated with first delayr instance:
         delayw    afdbk1

;Feed back signal, associated with second delayr instance:
         delayw    afdbk2
         outs      adly1, adly2

```

Voici un autre exemple de l'opcode deltapi. Il utilise le fichier *deltapi.csd* [examples/deltapi.csd].

## Exemple 210. Exemple de l'opcode deltapi.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o deltapi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gasig    init 0
gidel    = 1 ;delay time in seconds

instr 1

ain    pluck .7, 440, 1000, 0, 1
      outs ain, ain

vincr    gasig, ain ;send to global delay
endin

```

```

instr 2

ifeedback = p4

abuf2 delayr gidel
adell deltapi .4 ;first tap (on left channel)
adelm deltapi 1 ;second tap (on middle channel)
delayw gasig + (adell * ifeedback)

abuf3 delayr gidel
kdel line 1, p3, .01 ;vary delay time
adelR deltapi .65 * kdel ;one pitch changing tap (on the right chn.)
delayw gasig + (adelR * ifeedback)
;make a mix of all deayed signals
outs adell + adelm, adelR + adelm

clear gasig
endin

</CsInstruments>
<CsScore>

i 1 0 1
i 1 3 1
i 2 0 3 0 ;no feedback
i 2 3 8 .8 ;lots of feedback
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*deltap, deltap3, deltapn*

# deltapn

deltapn — Lit une ligne à retard avec des délais variables.

## Description

Lit une ligne à retard avec des délais variables.

## Syntaxe

```
ares deltapn xnumsamps
```

## Exécution

*xnumsamps* -- spécifie le délai de lecture en secondes. Chaque valeur est comprise entre 1 période de contrôle et le délai total de la paire lecture/écriture ; cependant, comme il n'y a pas de vérification interne du respect de cet intervalle, l'utilisateur est entièrement responsable. Chaque argument peut être une constante, une variable ou un signal variant dans le temps.

*deltapn* est identique à *deltapi*, sauf que la durée du retard est exprimée en échantillons plutôt qu'en secondes (Hans Mikelson).

Cet opcode peut lire directement dans une paire *delayr/delayw* en extrayant les données audio retardées des *idlt* secondes de son stocké. Il peut y avoir n'importe quel nombre d'unités *deltap* et/ou *deltapi* entre une paire lecture/écriture. Chacune reçoit un extrait audio sans changement de l'amplitude originale.

Ces opcodes peuvent fournir de multiples lectures de délai pour des lignes à retard arbitraires et des réseaux à rétroaction. Ils peuvent délivrer des délais constants ou variables, et sont utiles pour construire des effets de chorus, harmonizer et Doppler. Les délais constants (et certains de ceux qui varient lentement) ne nécessitent pas d'interpolation ; *deltap* leur convient très bien. Les délais variant à moyenne vitesse ou rapidement nécessiteront cependant les services supplémentaires de *deltapi*.

Les paires *delayr/delayw* peuvent être entrelacées. Pour associer une unité de lecture de délai à une unité *delayr*, elle doit non seulement être située entre ce *delayr* et le *delayw* approprié, mais aussi précéder tous les *delayr* suivants. Voir l'exemple 2. (Cette possibilité fut ajoutée dans la version 3.57 de Csound par Jens Groh et John ffitch).

*N.B.* Les délais de taux-k ne sont pas interpolés en interne, mais déroulent plutôt des décalages temporels d'échantillons audios ; c'est adéquat pour des délais changeant lentement. Cependant, pour les changements à vitesse moyenne ou rapides, il faut fournir en entrée des valeurs de délai avec une plus grande résolution de taux audio.

## Exemples

Voici un exemple de l'opcode *deltapn*. Il utilise le fichier *deltapn.csd* [examples/deltapn.csd].

### Exemple 211. Exemple de l'opcode *deltapn*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o deltap3.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gasig init 0
gidel = 1 ;delay time in seconds

instr 1

ain pluck .7, 440, 1000, 0, 1
outs ain, ain

vincr gasig, ain ;send to global delay
endin

instr 2

ifedback = p4

abuf2 delayr gidel
adelL deltapn 4000 ;first tap (on left channel)
adelM deltapn 44100 ;second tap (on middle channel)
delayw gasig + (adelL * ifedback)

abuf3 delayr gidel
kdel line 100, p3, 1 ;vary delay time
adelR deltapn 100 * kdel ;one pitch changing tap (on the right chn.)
delayw gasig + (adelR * ifedback)
;make a mix of all deayed signals
outs adelL + adelM, adelR + adelM

clear gasig
endin

</CsInstruments>
<CsScore>

i 1 0 1
i 1 3 1
i 2 0 3 0 ;no feedback
i 2 3 8 .8 ;lots of feedback
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*deltap, deltap3, deltapi*

# deltapx

deltapx — Lit depuis ou écrit dans une ligne à retard avec interpolation.

## Description

*deltapx* est semblable à *deltapi* ou à *deltap3*. Cependant, il permet une meilleure qualité d'interpolation. Cet opcode peut lire depuis et écrire dans une ligne à retard *delayr/delayw* avec interpolation.

## Syntaxe

aout **deltapx** adel, iwsiz

## Initialisation

*iwsiz* -- taille de la fenêtre d'interpolation en échantillons. Les valeurs permises sont des multiples entiers de 4 compris entre 4 et 1024. *iwsiz* = 4 utilise l'interpolation cubique. Des valeurs croissantes de *iwsiz* améliorent la qualité sonore au prix d'une utilisation plus intensive du CPU, et d'une durée de délai minimale.

## Exécution

*aout* -- Signal de sortie.

*adel* -- Délai en secondes.

```
a1      delayr  idlr
        deltapxw a2, adl1, iws1
a3      deltapi adl2, iws2
        deltapxw a4, adl3, iws3
        delayw  a5
```

Durées de délai minimales et maximales :

$idlr \geq 1/kr$	Longueur de la ligne à retard
$adl1 \geq (iws1/2)/sr$	Ecriture avant la lecture
$adl1 \leq idlr - (1 + iws1/2)/sr$	(permet des délais plus courts)
$adl2 \geq 1/kr + (iws2/2)/sr$	Temps de lecture
$adl2 \leq idlr - (1 + iws2/2)/sr$	
$adl2 \geq adl1 + (iws1 + iws2) / (2*sr)$	
$adl2 \geq 1/kr + adl3 + (iws2 + iws3) / (2*sr)$	
$adl3 \geq (iws3/2)/sr$	Ecriture après lecture
$adl3 \leq idlr - (1 + iws3/2)/sr$	(permet une rétroaction)



### Note

Les tailles de fenêtres des autres opcodes que *deltapx* sont : *deltap*, *deltapn* : 1, *deltapi* : 2 (linéaire), *deltap3* : 4 (cubique).

## Exemples

Voici un exemple de l'opcode `deltapx`. Il utilise le fichier `deltapx.csd` [examples/deltapx.csd].

### Exemple 212. Exemple de l'opcode `deltapx`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o deltapx.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 32
nchnls  = 2
0dbfs   = 1

instr 1

a1      phasor 300
a1      = a1 - 0.5
a_      delayr 1
adel    phasor 4
adel    = sin(2 * 3.14159265 * adel) * 0.01 + 0.2
deltapxw a1, adel, 32
adel    phasor 2
adel    = sin(2 * 3.14159265 * adel) * 0.01 + 0.2
deltapxw a1, adel, 32
adel    = 0.3
a2      deltapx adel, 32
a1      = 0
        delayw a1
        outs  a2*.7, a2*.7

endin
</CsInstruments>
<CsScore>

i1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*deltapxw*

## Crédits

Auteur : Istvan Varga  
Août 2001

Nouveau dans la version 4.13

# deltapxw

deltapxw — Mélange le signal d'entrée dans une ligne à retard.

## Description

*deltapxw* mélange le signal d'entrée dans une ligne à retard. Cet opcode peut être utilisé avec les unités de lecture (*deltap*, *deltapn*, *deltapi*, *deltap3* et *deltapx*) dans n'importe quel ordre ; la durée du délai étant la différence entre les dates de lecture et d'écriture. Cet opcode peut lire depuis et écrire dans une ligne à retard *delayr/delayw* avec interpolation.

## Syntaxe

**deltapxw** ain, adel, iwsiz

## Initialisation

*iwsiz* -- taille de la fenêtre d'interpolation en échantillons. Les valeurs permises sont des multiples entiers de 4 compris entre 4 et 1024. *iwsiz* = 4 utilise l'interpolation cubique. Des valeurs croissantes de *iwsiz* améliorent la qualité sonore au prix d'une utilisation plus intensive du CPU, et d'une durée de délai minimale.

## Exécution

*ain* -- Signal d'entrée.

*adel* -- Délai en secondes.

```
a1      delayr      idlr
        deltapxw a2, adl1, iws1
a3      deltapx  adl2, iws2
        deltapxw a4, adl3, iws3
        delayw      a5
```

Durées de délai minimales et maximales :

$idlr \geq 1/kr$	Longueur de la ligne à retard
$adl1 \geq (iws1/2)/sr$	Ecriture avant lecture
$adl1 \leq idlr - (1 + iws1/2)/sr$	(permet des délais plus courts)
$adl2 \geq 1/kr + (iws2/2)/sr$	Temps de lecture
$adl2 \leq idlr - (1 + iws2/2)/sr$	
$adl2 \geq adl1 + (iws1 + iws2) / (2*sr)$	
$adl2 \geq 1/kr + adl3 + (iws2 + iws3) / (2*sr)$	
$adl3 \geq (iws3/2)/sr$	Ecriture après lecture
$adl3 \leq idlr - (1 + iws3/2)/sr$	(permet une rétroaction)



### Note

Les tailles de fenêtres des autres opcodes que *deltapx* sont : *deltap*, *deltapn* : 1, *deltapi* : 2 (linéaire), *deltap3* : 4 (cubique).



## Exemples

Voici un exemple de l'opcode `deltapxw`. Il utilise le fichier `deltapxw.csd` [examples/deltapxw.csd].

### Exemple 213. Exemple de l'opcode `deltapxw`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o deltapxw.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 32
nchnls  = 2
0dbfs   = 1

instr 1

a1      phasor 300
a1      = a1 - 0.5
a_      delayr 1
adel    phasor 4
adel    = sin(2 * 3.14159265 * adel) * 0.01 + 0.2
deltapxw a1, adel, 32
adel    phasor 2
adel    = sin(2 * 3.14159265 * adel) * 0.01 + 0.2
deltapxw a1, adel, 32
adel    = 0.3
a2      deltapx adel, 32
a1      = 0
        delayw a1
        outs  a2*.7, a2*.7

endin
</CsInstruments>
<CsScore>

i1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*deltapx*

## Crédits

Auteur : Istvan Varga  
Août 2001

Nouveau dans la version 4.13

# denorm

denorm — Ajoute du bruit de bas niveau à une liste de signaux de taux-a.

## Description

Ajoute du bruit de bas niveau (~1e-20 pour les flottants et ~1e-56 pour les doubles) à une liste de signaux de taux-a. On peut l'utiliser avant les filtres RII et les réverbérations pour éviter les nombres dénormalisés qui peuvent sinon produire un accroissement significatif de l'utilisation des ressources CPU.

## Syntaxe

```
denorm a1[, a2[, a3[, ... ]]]
```

## Exécution

*a1[, a2[, a3[, ... ]]]* -- signaux auxquels on ajoute du bruit.

Certaines architectures de processeur (particulièrement les Pentium IV) sont très lentes pour traiter les très petits nombres. Ces petits nombres peuvent résulter de certains traitements à rétroaction décroissante comme la réverbération et les filtres RII. On peut ajouter du bruit de faible niveau afin que les nombres très petits ne soient jamais atteints et soient "absorbés" par le "fond bruiteux".

Si l'utilisation du CPU atteint les 100% en queue de réverbération ou si l'on obtient des discontinuités audio dans un traitement qui ne devrait pas beaucoup utiliser le CPU, l'utilisation de *denorm* avant l'opcode ou le traitement fautif peut résoudre le problème.

## Exemples

Voici un exemple de l'opcode denorm. Il utilise le fichier *denorm.csd* [examples/denorm.csd].

### Exemple 214. Exemple de l'opcode denorm.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o denorm.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
; slightly simplified example from Istvan Varga 2006
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

garvb init 0

instr 1
```

```

a1 oscils 0.6, 440, 0
a2 linsegr 0, 0.005, 1, 3600, 1, 0.08, 0
a1 = a1 * a2
    vincr garvb, a1
    outs a1, a1
endin

instr 99 ;"Always on"

    denorm garvb
aL, aR reverbsc garvb * 0.5, garvb * 0.5, 0.92, 10000
    clear garvb
    outs aL, aR
endin

</CsInstruments>
<CsScore>

i 99 0 -1 ;held by a negative p3, means "always on"
i 1 0 0.5
i 1 4 0.5
e 8 ;8 extra seconds after the performance

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Istvan Varga  
2005

# diff

diff — Modifie un signal par différentiation.

## Description

Modifie un signal par différentiation.

## Syntaxe

```
ares diff asig [, iskip]
kres diff ksig [, iskip]
```

## Initialisation

*iskip* (facultatif) -- état initial de l'espace mémoire interne (voir *reson*). La valeur par défaut est 0.

## Exécution

*integ* et *diff* réalisent l'intégration et la différentiation d'un signal de contrôle ou audio en entrée. Ils sont mutuellement inverses et l'application des deux reconstruit le signal original. Comme ces unités sont des cas particuliers de filtres passe-bas et passe-haut, ils produisent une sortie pondérée (et modifiée en phase) en fonction de la fréquence. Ainsi *diff* d'un sinus produit un cosinus dont l'amplitude vaut  $2 * \pi * \text{Hz} / \text{sr}$  de l'original (pour chaque partiel) ; *integ* affectera inversement les amplitudes de ses composants en entrée. Sachant cela, ces unités peuvent fournir d'utiles modifications de signal.

## Exemples

Voici un exemple de l'opcode diff. Il utilise le fichier *diff.csd* [examples/diff.csd].

### Exemple 215. Exemple de l'opcode diff.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o diff.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

asig diskin2 "fox.wav", 1
```

```

        outs asig, asig

    endin

    instr 2 ; with diff

    asig diskin2 "fox.wav", 1
    ares diff asig
        outs ares, ares

    endin

    instr 3 ; with integ

    asig diskin2 "fox.wav", 1
    aint integ asig
    aint = aint*.05 ;way too loud
        outs aint, aint

    endin

    instr 4 ; with diff and integ

    asig diskin2 "fox.wav", 1
    ares diff asig
    aint integ ares
        outs aint, aint

    endin

</CsInstruments>
<CsScore>

i 1 0 1
i 2 1 1
i 3 2 1
i 4 3 1

e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*downsamp, integ, interp, samphold, upsamp*

# diode\_ladder

`diode_ladder` — Implémentation d'un filtre en échelle à diodes à 4 pôles avec rétroaction sans retard.

## Description

Implémentation d'un filtre passe-bas à diodes à 4 pôles (24 dB/oct) avec rétroaction sans retard. Ce type de filtre fut utilisé à l'origine dans l'EMS VCS3 et il fut le filtre résonnant du Roland TB-303.

## Syntaxe

```
asig diode_ladder ain, xcf, xk [, inlp, isaturation, istor]
```

## Initialisation

*istor* --état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal de sortie passe-bas.

*ain* -- signal d'entrée.

*xcf* -- fréquence de coupure du filtre (taux-i-, k ou a).

*xk* -- valeur de rétroaction du filtre (taux-i-, k ou a) qui contrôle la résonance. Dans l'intervalle 0.0-17.0. L'auto-oscillation survient à 17.0.

*knlp* (facultatif, 0 par défaut) -- méthode de traitement non linéaire. 0 = pas de traitement, 1 = TNL de grande qualité, 2 = TNL de faible qualité (plus rapide). La méthode 1 utilise  $(1.0 / \tanh(ksaturation)) * \tanh(ksaturation * input)$ . La méthode 2 utilise  $\tanh(ksaturation * input)$ . L'activation du TNL peut pousser la sortie globale du filtre au-delà de l'unité et doit être compensée pour l'environnement externe du filtre.

*ksaturation* (facultatif, 1 par défaut) -- quantité de saturation à utiliser pour le traitement non linéaire. Les valeurs > 1 augmentent la pente de la courbe de TNL.

## Exemples

Voici un exemple de l'opcode `diode_ladder`. Il utilise le fichier `diode_ladder.csd` [examples/diode\_ladder.csd].

### Exemple 216. Exemple de l'opcode `diode_ladder`.

```
<CsoundSynthesizer>
<CsOptions>
-o dac
</CsOptions>
<CsInstruments>
sr=44100
ksmps=16
nchnls=2
```

```

Odbfs=1

gi_sine ftgen 0, 0, 65537, 10, 1

gkcut init 6000

instr modulation
    gkcut = lfo(4000, 0.1) + 6000
endin

instr bass

    iamp = ampdbfs(-12)
    ipch = cps2pch(p4, 12)

    asig = vco2(0.5, ipch, 0)

    acut = expon:a(i(gkcut), p3, 200)
    aout = diode_ladder(asig, acut, 8, 1, 4)

    aout *= expseg:a(1.0, p3 - 0.05, 1.0, 0.05, 0.001)

    aout = limit(aout, -1.0, 1.0)

    outc(aout, aout)

endin

gipat[] init 8
gipat[0] = 6.00
gipat[1] = 7.00
gipat[2] = 6.00
gipat[3] = 7.00
gipat[4] = 5.07
gipat[5] = 6.07
gipat[6] = 5.08
gipat[7] = 6.08

instr player
    indx = p4

    ;; play instrument
    if(gipat[indx] > 0) then
        schedule("bass", 0, 0.2, gipat[indx])
    endif

    ;; temporal recursion
    schedule("player", 0.2, 0.1, (indx + 1) % lenarray(gipat))

endin

schedule("modulation", 0, -1)
schedule("player", 0, 0.1, 0)
event_i("e", 0, 0.1 * 128)

</CsInstruments>

<CsScore>
</CsScore>

</CsoundSynthesizer>

```

## Références

Ce filtre est basé sur les travaux de Will Pirkle qui emploie le travail de Vadim Zavalishin pour créer des implémentations de filtres analogiques à transformation préservant la topologie (TPT), avec des transformations bilinéaires.

1. Pirkle, Will. Designing Software Synthesizer Plug-ins in C++: For RackAFX, VST3, and Audio Units. CRC Press, 2014.
2. Pirkle, Will. AN-6: Virtual Analog (VA) DiodeLadder Filter. 2013.
3. Zavalishin, Vadim. "The Art of VA filter design." Native Instruments, 2012.

## Crédits

Auteur : Steven Yi  
Avril 2017

Nouveau dans Csound 6.09.0



# directory

`directory` — Lit un répertoire et restitue la liste des noms de fichiers dans un tableau.

## Description

Lit un répertoire pour les noms de fichiers et les passe dans un tableau de chaînes de caractères. On peut choisir le type de fichier en passant une extension de nom de fichier sous forme de chaîne de caractères.

## Syntaxe

```
SFiles[] directory SDirectory[, SExtention]
```

## Initialisation

*SDirectory* -- une chaîne de caractères identifiant le répertoire dont on veut lister les fichiers.

*SExtention* -- Facultatif. Fixe le type de fichier désiré. S'il n'est pas utilisé, tous les noms de fichiers sont listés.

## Exécution

*SFiles[]* -- Un tableau de chaînes de caractères contenant les noms de tous les fichiers d'un type donné trouvés dans le répertoire.



### Note

Ne fonctionne qu'au taux-i et ne tient pas compte des changements effectués dans le répertoire après le début de l'exécution.

## Exemple

Cet exemple montre comment on peut utiliser *directory* pour lister les fichiers .wav à un endroit donné. *printf\_i* est ensuite utilisé pour afficher les noms des fichiers. Il utilise le fichier *directory.csd* [exemples/directory.csd].

### Exemple 217. Exemple de l'opcode `directory`.

```
<CsoundSynthesizer>
<CsOptions>
-n
</CsOptions>
<CsInstruments>

;browse for text files in current directory
instr 1
iCnt init 0
SFileNames[] directory ".", ".txt"
iNumberOfFiles lenarray SFileNames

until iCnt>=iNumberOfFiles do
  printf_i "Filename = %s \n", 1, SFileNames[iCnt]
```

```
    iCnt = iCnt+1
  od
endin

</CsInstruments>
<CsScore>
i1 0 1
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Rory Walsh  
2016

# diskgrain

diskgrain — Synthèse granulaire synchrone, utilisant un fichier son comme source.

## Description

*diskgrain* implémente la synthèse granulaire synchrone. La source sonore des grains est obtenue en lisant un fichier son contenant les échantillons de la forme d'onde source.

## Syntaxe

```
asig diskgrain Sfname, kamp, kfreq, kpitch, kgrsize, kprate, \  
ifun, iolaps [,imaxgrsize , ioffset]
```

## Initialisation

*Sfilename* -- fichier son source.

*ifun* -- table de fonction de l'enveloppe de grain.

*iolaps* -- nombre maximum de chevauchements,  $\max(kfreq) \cdot \max(kgrsize)$ . Une grande valeur d'estimation ne devrait pas affecter l'exécution, mais le dépassement de cette valeur aura probablement des conséquences désastreuses.

*imaxgrsize* -- taille de grain maximale en secondes (par défaut 1.0).

*ioffset* -- décalage initial en secondes à partir du début du fichier (par défaut 0).

## Exécution

*kamp* -- pondération de l'amplitude

*kfreq* -- fréquence de génération des grains, ou densité, en grains/sec.

*kpitch* -- transposition de hauteur des grains (1 = hauteur normale, < 1 plus bas, > 1 plus haut ; négatif, lecture à l'envers)

*kgrsize* -- taille de grain en secondes.

*kprate* -- vitesse du pointeur de lecture, en grains. Une valeur de 1 avancera le pointeur de lecture d'un grain dans la table source. Des valeurs supérieures provoqueront une compression temporelle et des valeurs inférieures une expansion temporelle du signal source. Avec des valeurs négatives, le pointeur progressera à l'envers et zéro l'immobilisera.

Le générateur de grain contrôle complètement la fréquence (grains/sec), l'amplitude globale, la hauteur de grain (un incrément de l'échantillonnage) et la taille de grain (en secondes), comme paramètres fixes ou variant dans le temps (signaux). La vitesse du pointeur de grain est un paramètre supplémentaire qui contrôle la position à laquelle le générateur commencera à lire les échantillons dans le fichier pour chaque grain successif. Elle est mesurée en fraction de la taille de grain, si bien qu'une valeur de 1 (par défaut) provoquera la lecture de chaque grain successif à partir de l'endroit où le grain précédent s'est terminé. Avec une valeur de 0.5 le grain suivant commencera à la position médiane entre le début et la fin du grain précédent, etc... Avec une valeur de 0 le générateur lit toujours à partir d'une position fixe (quelque soit l'endroit où il

se trouvait précédemment). Une valeur négative décrémentera les positions du pointeur. Ce contrôle donne plus de flexibilité pour créer des modifications de l'échelle temporelle pendant la resynthèse.

*Diskgrain* générera n'importe quel nombre de flux de grain parallèles (en fonction de la densité/fréquence de grain) borné par la valeur de *iolaps* (par défaut 100). Le nombre de flux (grains se chevauchant) est déterminé par  $kgrsize * kfreq$ . Plus il y aura de chevauchements, plus il y aura de calculs ce qui pourra empêcher la synthèse en temps réel (selon la puissance du processeur).

*Diskgrain* peut simuler une synthèse formantique à la FOF, si on emploie une forme adéquate comme enveloppe de grain et une forme d'onde sinus comme onde de grain. Pour cette utilisation, on peut choisir des tailles de grain d'environ 0.04 secondes. La fréquence centrale du formant est déterminée par la hauteur de grain. Comme celle-ci est exprimée en incrément d'échantillonnage, il faut pondérer cette valeur par  $tablesize/sr$  pour obtenir une fréquence en Hz. La fréquence de grain déterminera le fondamental.

Cet opcode est une variation sur l'opcode *syncgrain*.



### Note

*diskgrain* n'effectue aucun ré-échantillonnage, ce qui produira une transposition de hauteur si le taux d'échantillonnage du fichier source est différent de celui de Csound.

## Exemples

Voici un exemple de l'opcode *diskgrain*. Il utilise le fichier *diskgrain.csd* [examples/diskgrain.csd].

### Exemple 218. Exemple de l'opcode *diskgrain*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime output leave only the line below:
; -o diskgrain.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 32
nchnls  = 2
0dbfs   = 1

instr 1

iolaps  = 2
igrsize = 0.04
ifreq   = iolaps/igrsize
ips     = 1/iolaps

istr = p4 /* timescale */
ipitch = p5 /* pitchscale */

a1 diskgrain "mary.wav", 1, ifreq, ipitch, igrsize, ips*istr, 1, iolaps
outs a1, a1

endin
```

```
</CsInstruments>
<CsScore>
f 1 0 8192 20 2 1 ;Hanning function

;          timescale  pitchscale
i 1 0 5 1 1
i 1 + 5 2 1
i 1 + 5 1 0.75
i 1 + 5 1.5 1.5
i 1 + 5 0.5 1.5

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
Mai 2007  
Nouveau dans Csound 5.06

# diskin

diskin — Lit des données audio d'un périphérique ou d'un flot externe et peut altérer leur hauteur.

## Description

Lit des données audio d'un périphérique ou d'un flot externe et peut altérer leur hauteur.

## Syntaxe

```
ar1 [, ar2 [, ar3 [, ... arN]]] diskin ifilcod[, kpitch[, iskiptim \
    [, iwraparound[, iformat[, iskipinit]]]]]
```

Noter que N valait 24 dans les versions antérieures à la 5.14 et qu'il vaut maintenant 40.

```
ar1[] diskin ifilcod[, kpitch[, iskiptim] \
    [, iwraparound[, iformat[, iskipinit]]]]]
```

(dans cette version, le nombre de canaux de sortie n'est pas limité.)

## Initialisation

*ifilcod* -- entier ou chaîne de caractères donnant le nom du fichier son source. Un entier indique le fichier soundin.filcod ; une chaîne de caractères (entre guillemets, espaces autorisés) donne le nom de fichier lui-même, éventuellement un nom de chemin complet. Si ce n'est pas un nom de chemin complet, le fichier nommé est d'abord cherché dans le répertoire courant, puis dans celui qui est donné par la variable d'environnement *SSDIR* (si elle est définie) puis par *SFDIR*. Voir aussi *GEN01*.

*iskiptim* (facultatif) -- portion du son en entrée à ignorer, exprimée en secondes. La valeur par défaut est 0.

*iformat* (facultatif) -- ignoré s'il est positif, mais s'il est négatif il spécifie le format des données audio d'un fichier brut :

- -1 = caractères signés sur 8 bit (les 8 bit de poids fort d'un entier sur 16 bit)
- -2 = octets sur 8 bit A-law
- -3 = octets sur 8 bit U-law
- -4 = entiers courts sur 16 bit
- -5 = entiers longs sur 32 bit
- -6 = flottants sur 32 bit
- -7 = entiers non signés sur 8 bit (non disponible dans les versions de Csound antérieures à la 5.00)
- -8 = entiers sur 24 bit (non disponible dans les versions de Csound antérieures à la 5.00)
- -9 = doubles sur 64 bit (non disponible dans les versions de Csound antérieures à la 5.00)

*iwraparound* -- 1 = activé, 0 = désactivé (parcours cyclique du fichier dans les deux directions, ce qui permet les boucles)

*iskipinit* -- supprime toute initialisation s'il est non nul (vaut 0 par défaut). Fut introduit dans la version 4\_23f13 et dans csound5.

Si *iformat* = 0, il est déduit de l'en-tête du fichier, et s'il n'y a pas d'en-tête, de l'option de ligne de commande *-o* de Csound. La valeur par défaut est 0.

## Exécution

*a1 ... a24* -- signaux de sortie, dans l'intervalle -0dbfs, 0dbfs. Tous les échantillons avant le début (position négative) et après la fin du fichier sont supposés nuls, à moins que *iwrap* soit différent de zéro. Le nombre d'arguments en sortie doit être le même que le nombre de canaux dans le fichier, que l'on peut déterminer avec l'opcode *filenchnls*, sinon il y aura une erreur.

*ar1[]* --- signaux de sortie, dans un tableau de taux-a de taille N, où N est le nombre de canaux dans le fichier. Les tableaux sont automatiquement alloués avec la taille correcte.

*kpitch* -- N'importe quel nombre réel. Un nombre négatif signifie une restitution à l'envers. Ce nombre est un rapport de hauteur où :

- 1 = hauteur normale (par défaut)
- 2 = 1 octave plus haut
- 3 = 12ème plus haut, etc.
- .5 = 1 octave plus bas
- .25 = 2 octaves plus bas, etc.
- -1 = hauteur normale à l'envers
- -2 = 1 octave plus haut à l'envers, etc.



### Note pour les utilisateurs de Windows

Les utilisateurs de Windows utilisent normalement des anti-slash, « \ », pour spécifier les chemins de leurs fichiers. Par exemple un utilisateur de Windows pourra utiliser le chemin « c:\music\samples\loop001.wav ». Ceci pose problème car les anti-slash servent habituellement à spécifier des caractères spéciaux.

Pour spécifier correctement ce chemin dans Csound on peut utiliser :

- soit le *slash* : c:/music/samples/loop001.wav
- soit le *caractère spécial d'anti-slash*, « \\ » : c:\\music\\samples\\loop001.wav

## Exemples

Voici un exemple de l'opcode *diskin*. Il utilise les fichiers *diskin.csd* [examples/diskin.csd] et *beats.wav* [examples/beats.wav].

### Exemple 219. Exemple de l'opcode *diskin*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o diskin.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 32
nchnls  = 2
0dbfs   = 1

instr 1

ktrans linseg 1, 5, 2, 10, -2
a1      diskin "beats.wav", ktrans, 0, 1, 0, 32
outs a1, a1

endin

</CsInstruments>
<CsScore>

i 1 0 15
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*in, inh, ino, inq, ins, soundin* and *diskin2*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

Nouveau dans la version 3.46

*kpitch* devient facultatif dans la version 6.06

Avertissement pour les utilisateurs de Windows ajouté par Kevin Conder, avril 2002



# diskin2

diskin2 — Lit des données audio depuis un fichier, et peut altérer leur hauteur au moyen d'un des types d'interpolation disponibles ainsi que convertir le taux d'échantillonnage pour s'accorder à celui de l'orchestre.

## Description

Lit des données audio depuis un fichier, et peut altérer leur hauteur au moyen d'un des types d'interpolation disponibles ainsi que convertir le taux d'échantillonnage pour s'accorder à celui de l'orchestre. *diskin2* peut également lire des fichiers multicanaux dont le nombre de canaux est compris entre 1 et 24 pour les versions de Csound antérieures à la 5.14, et entre 1 et 40 pour les versions suivantes.

## Syntaxe

```
a1[, a2[, ... aN]] diskin2 ifilcod[, kpitch[, iskiptim \
[, iwrap[, iformat[, iwsizel[, ibufsize[, iskipinit]]]]]]]
```

*diskin2* offre plus de contrôle et une meilleure qualité de son que *diskin* mais au prix d'une utilisation plus intensive des ressources CPU.

```
ar1[] diskin2 ifilcod[, kpitch[, iskiptim \
[, iwrap[, iformat[, iwsizel[, ibufsize[, iskipinit]]]]]]]
```

(dans la version de la sortie dans un tableau, le nombre de canaux de sortie n'a pas de limite supérieure.)

## Initialisation

*ifilcod* -- entier ou chaîne de caractères donnant le nom du fichier son source. Un entier indique le fichier soundin.filcod ; une chaîne de caractères (entre guillemets, espaces autorisés) donne le nom de fichier lui-même, éventuellement un nom de chemin complet. Si ce n'est pas un nom de chemin complet, le fichier nommé est d'abord cherché dans le répertoire courant, puis dans celui qui est donné par la variable d'environnement *SSDIR* (si elle est définie) puis par *SFDIR*. Voir aussi *GEN01*. Note : il est possible que les fichiers contenant plus de  $2^{31}-1$  trames d'échantillons ne soient pas joués correctement sur les plates-formes 32 bit ; cela donne une longueur maximale d'environ trois heures avec un taux d'échantillonnage de 192000 Hz.

*iskiptim* (facultatif, zéro par défaut) -- portion du son en entrée à ignorer, exprimée en secondes, en supposant que *kpitch*=1. Peut être négatif, pour ajouter *-iskiptim/kpitch* secondes de délai au lieu de d'ignorer une partie du son.



### Note

Si *iwrap* est différent de 0 (lecture cyclique), *iskiptim* ne retardera pas le son si une valeur négative est utilisée. Au lieu de cela, la lecture commencera du même décalage avant la fin du fichier.

*iwrap* (facultatif, zéro par défaut) -- s'il a n'importe quelle valeur non nulle, les positions de lecture négatives ou au-delà de la fin du fichier sont ramenées à l'intérieur de la durée du fichier son au lieu d'être remplacées par des échantillons nuls. Pratique pour jouer un fichier en boucle.



### Note

Si *iwrap* est activé, la longueur du fichier ne doit pas être inférieure à la taille de la fenêtre d'interpolation (voir ci-dessous), sinon il pourra y avoir des craquements dans le son de sortie.

*iformat* (facultatif, zéro par défaut) -- fixe le format d'échantillon pour les fichiers bruts (sans en-tête). Ce paramètre doit être nul si le fichier a un en-tête. Les valeurs admises pour les fichiers bruts sont :

- 1 : entiers courts sur 16 bit
- 2 : caractères signés sur 8 bit (les 8 bit de poids fort d'un entier sur 16 bit)
- 3 : octets sur 8 bit A-law
- 4 : octets sur 8 bit U-law
- 5 : entiers courts sur 16 bit
- 6 : entiers longs sur 32 bit
- 7 : flottants sur 32 bit
- 8 : entiers non signés sur 8 bit
- 9 : entiers sur 24 bit
- 10 : doubles sur 64 bit



### Note

Cette liste n'est pas la même que celle qui est dans *GEN01*.

*iwsiz*e (facultatif, zéro par défaut) -- taille de la fenêtre d'interpolation, en échantillons. Peut prendre une de ces valeurs :

- 1 : arrondi à l'échantillon le plus proche (pas d'interpolation, pour *kpitch*=1)
- 2 : interpolation linéaire
- 4 : interpolation cubique
- >= 8 : interpolation par sinc de *iwsiz*e points avec anti-aliasing (lent)

Zéro ou des valeurs négatives sélectionnent la valeur par défaut, qui est l'interpolation cubique.



### Note

S'il y a interpolation, *kpitch* est automatiquement mis à l'échelle par le rapport des taux d'échantillonnage du fichier et de l'orchestre, afin que le fichier soit toujours joué à la hauteur originale si *kpitch* vaut 1. Cependant, la conversion du taux d'échantillonnage est désactivée lorsque *iwsiz*e vaut 1.

*ibufsize* (facultatif, zéro par défaut) -- taille du tampon en échantillons mono (pas en trames d'échantillons). Ce n'est que la valeur suggérée, la valeur retenue étant arrondie afin que le nombre de trames d'échantillons soit une puissance entière de deux et soit comprise entre 128 (ou *iwsiz*e s'il est supérieur à 128) et 1048576. La valeur par défaut, 4096, choisie par zéro ou une valeur négative, sera adéquate dans la plupart des cas, mais lors du mélange de plusieurs fichiers son de grande taille en temps différé, une grande taille de tampon est recommandée pour améliorer l'efficacité des lectures sur disque. Pour une sortie en temps réel, la lecture des fichiers depuis un RAM disque rapide (sur les plates-formes qui le permettent) avec une petite taille de tampon est préférable.

*iskipinit* (facultatif, zéro par défaut) -- supprime l'initialisation s'il est non nul.

## Exécution

*a1 ... a24* -- signaux de sortie, dans l'intervalle allant de -0dbfs à 0dbfs. Tous les échantillons avant le début du fichier (positions négatives) et après la fin du fichier prennent la valeur zéro, à moins que *iwrap* soit non nul. Le nombre d'arguments de sortie doit être le même que le nombre de canaux du fichier son - lequel peut être déterminé avec l'opcode *filenchnls*, sinon il y aura une erreur d'initialisation.

*ar1[]* --- signaux de sortie, dans un tableau de taux-a de taille N, où N est le nombre de canaux dans le fichier. Les tableaux sont automatiquement alloués avec la taille correcte.



### Note

Il est plus efficace de lire un seul fichier avec plusieurs canaux, que plusieurs fichiers à un seul canal, spécialement avec de grandes valeurs de *iwsiz*e.

*kpitch* -- transpose la hauteur du son en entrée par ce facteur (par exemple 0.5 signifie une octave plus bas, 2 une octave plus haut, et 1 la hauteur originale, qui est la valeur par défaut). Des valeurs fractionnaires et négatives sont permises (les dernières provoquant une lecture à l'envers, cependant, dans ce cas, *iskiptim* doit prendre une valeur positive, par exemple la longueur du fichier, ou bien *iwrap* doit être non nul, sinon rien ne sera joué). S'il y a interpolation et que le taux d'échantillonnage du fichier est différent de celui de l'orchestre, le rapport de transposition est automatiquement ajusté de façon à ce que *kpitch*=1 joue à la hauteur originale. Un *iwsiz*e élevé (40 ou plus) peut améliorer significativement la qualité du son lors d'une transposition vers l'aigu, au prix d'une utilisation plus intensive des ressources CPU.

## Exemples

Voici un exemple de l'opcode *diskin2*. Il utilise les fichiers *diskin2.csd* [examples/diskin2.csd] et *beats.wav* [examples/beats.wav].

### Exemple 220. Exemple de l'opcode *diskin2*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o diskin2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 32
nchnls  = 2
0dbfs   = 1

instr 1

ktrans  linseg 1, 5, 2, 10, -2
a1      diskin2 "beats.wav", ktrans, 0, 1, 0, 32
outs    a1, a1

endin

</CsInstruments>
```

```
<CsScore>  
  
i 1 0 15  
e  
  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*in, inh, ino, inq, ins, soundin et disk*

## Crédits

Auteur : Istvan Varga  
2005

Nouveau dans la version 5.00

*kpitch* devient facultatif dans la version 6.06

# dispffft

displayfft — Affiche la transformée de Fourier d'un signal audio ou de contrôle.

## Description

Ces unités affichent les valeurs d'initialisation de l'orchestre ou produisent un affichage graphique de signaux de contrôle ou audio de l'orchestre. Des fenêtres X11 sont utilisées s'il est activé, sinon (ou si l'option `-g` est positionnée) on a un affichage approximatif en caractères ASCII.

## Syntaxe

```
dispffft xsig, iprd, iwsiz [, iwtyp] [, idbout] [, iwtflg] [,imin] [,imax]
```

## Initialisation

*iprd* -- la période d'affichage en secondes.

*iwsiz* -- taille de la fenêtre d'entrée en échantillons. Une fenêtre de *iwsiz* points produira une transformée de Fourier de *iwsiz*/2 points, répartis linéairement en fréquence de 0 à *sr*/2. *iwsiz* doit être une puissance de 2, comprise entre 16 et 4096. Les fenêtres peuvent se chevaucher.

*iwtyp* (facultatif, 0 par défaut) -- type de fenêtre. 0 = rectangulaire, 1 = Hanning. La valeur par défaut est 0 (rectangulaire).

*idbout* (facultatif, 0 par défaut) -- unité d'affichage des coefficients de Fourier. 0 = magnitude, 1 = décibels. La valeur par défaut est 0 (magnitude).

*iwtflg* (facultatif, 0 par défaut) -- indicateur de maintien. S'il est différent de zéro, chaque affichage est maintenu jusqu'à ce que l'utilisateur le libère. La valeur par défaut est 0 (pas de maintien).

*imin* (facultatif, 0 par défaut) -- bin minimum de la FFT à afficher.

*imax* (facultatif, *winsize*/2 par défaut) -- bin maximum de la FFT à afficher.

## Exécution

*dispffft* -- affiche la transformée de Fourier d'un signal audio ou de contrôle (*asig* ou *ksig*) chaque *iprd* secondes au moyen de la méthode de transformée de Fourier rapide.

## Exemples

Voici un exemple de l'opcode *dispffft*. Il utilise le fichier *dispffft.csd* [examples/dispffft.csd].

### Exemple 221. Exemple de l'opcode *dispffft*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>
```

```

; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o dispfft.wav -W ;; for file output any platform
</CsOptions>;be sure to NOT have -d in the CsOptions...
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kcps = 110
ifn = 1

knh line p4, p3, p5
asig buzz 1, kcps, knh, ifn
outs asig, asig

dispfft asig, .1, 2048, 0, 1

endin
</CsInstruments>
<CsScore>
;sine wave.
f 1 0 16384 10 1

i 1 0 3 20 20
i 1 + 3 3 3
i 1 + 3 150 1
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*display, print*

# display

display — Affiche un signal audio ou de contrôle sur un graphique amplitude/temps.

## Description

Ces unités affichent les valeurs d'initialisation de l'orchestre ou produisent un affichage graphique de signaux de contrôle ou audio de l'orchestre. Des fenêtres X11 sont utilisées s'il est activé, sinon (ou si l'option `-g` est positionnée) on a un affichage approximatif en caractères ASCII.

## Syntaxe

```
display xsig, iprd [, inprds] [, iwtflg]
```

## Initialisation

*iprd* -- la période d'affichage en secondes.

*inprds* (facultatif, 1 par défaut) -- Nombre de périodes d'affichage retenues dans chaque graphique. Les valeurs supérieures ou égales à 2 donneront une perspective plus étendue du mouvement du signal. La valeur par défaut est 1 (chaque graphique est entièrement renouvelé). *inprds* est un facteur d'échelle pour la forme d'onde affichée, qui contrôle combien de trames d'échantillon de longueur *iprd* sont dessinées dans la fenêtre (la valeur par défaut qui est aussi la valeur minimale est 1.0). Des valeurs supérieures de *inprds* provoquent un dessin plus lent (plus de points à dessiner) mais feront défiler la forme d'onde à travers la fenêtre, ce qui est utile avec de faibles valeurs de *iprd*.

*iwtflg* (facultatif, 0 par défaut) -- indicateur de maintien. S'il est différent de zéro, chaque affichage est maintenu jusqu'à ce que l'utilisateur le libère. La valeur par défaut est 0 (pas de maintien).

## Exécution

*display* -- affiche le signal audio ou de contrôle *xsig* chaque *iprd* secondes, sur un graphique amplitude/temps.

## Exemples

Voici un exemple de l'opcode display. Il utilise le fichier *display.csd* [exemples/display.csd].

### Exemple 222. Exemple de l'opcode display.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o display.wav -W ;; for file output any platform
</CsOptions>;be sure to NOT have -d in the CsOptions...
<CsInstruments>
```

```
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kcps = 110
ifn = 1

knh line p4, p3, p5
asig buzz 1, kcps, knh, ifn
outs asig, asig

display asig, .1, 3

endin
</CsInstruments>
<CsScore>
;sine wave.
f 1 0 16384 10 1

i 1 0 3 20 20
i 1 + 3 3 3
i 1 + 3 150 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*dispfft, print*

## Crédits

Commentaires sur le paramètre *inprds* par Rasmus Ekman.



# distort

distort — Distorsion non-linéaire d'un signal audio avec écrêtage optionnel.

## Description

Distorsion non-linéaire d'un signal audio avec écrêtage optionnel.

## Syntaxe

```
ar distort asig, kdist, ifn[, ihp, istor]
```

## Initialisation

*ifn* -- numéro de table d'une fonction de distortion non-linéaire avec point de garde. La fonction peut avoir n'importe quelle forme, mais elle doit traverser 0, milieu de la table, avec une pente positive. Il n'est pas nécessaire d'avoir une table de grande taille car la lecture se fait avec interpolation.

*ihp* -- (facultatif) point à mi-puissance (en Hz) du filtre passe-bas interne. La valeur par défaut est 10.

*istor* -- (facultatif) état initial de la mémoire interne (voir *reson*). La valeur par défaut est 0.

## Exécution

*asig* -- Signal audio à traiter

*kdist* -- Taux de distortion (habituellement entre 0 et 1)

Cette unité déforme un signal entrant au moyen d'une fonction de distortion non-linéaire *ifn* et un indice de distortion *kdist*. Le signal d'entrée est d'abord compressé en prenant sa valeur efficace puis passé par une fonction de distorsion non-linéaire qui peut modifier sa forme et son spectre. Finalement il est remis à l'échelle la plus proche de sa puissance originale.

Le taux de distorsion dépend de la nature de la fonction déformante et de la valeur de *kdist*, qui est généralement compris entre 0 et 1. On souhaite que pour les faibles valeurs de *kdist* la fonction déformante laisse le signal quasi inchangé. Ceci sera le cas si, au point médian de la table, la fonction déformante est presque linéaire et traverse 0 avec une pente positive. Une fonction segment de droite entre -1 et +1 satisfait à ces exigences ; une sigmoïde (sinusoïde entre 270 et 90 degrés) également. Lorsque *kdist* augmente, le signal compressé est dilaté pour rencontrer plus de parties de la fonction déformante, et si celle-ci devient non-linéaire, le signal est de plus en plus *tordu* lors de son parcours pour obtenir une distorsion.

Lorsque *kdist* devient suffisamment grand, le parcours va éventuellement atteindre les limites de la table. La table n'est pas lue de manière cyclique, mais se « bloque » sur les points extrêmes lorsque le signal entrant les dépasse ; cela introduit de l'écrêtage, une forme supplémentaire de distorsion du signal. Le point où l'écrêtage commence dépend de la complexité (différence entre la valeur efficace et le pic) du signal entrant. Pour une sinusoïde pure, l'écrêtage ne commence que si *kdist* dépasse 0.7 ; pour une entrée plus complexe, l'écrêtage peut commencer avec *kdist* à 0.5 ou beaucoup moins. *kdist* peut dépasser le point d'écrêtage de n'importe quelle quantité et peut être supérieur à 1.

La fonction déformante peut être rendue arbitrairement complexe pour plus d'effets. Elle doit généralement être continue bien que ce ne soit pas nécessaire. Elle doit aussi être régulière près du point médian et répartie à peu près également entre les valeurs positives et les valeurs négatives sur l'ensemble, sinon un

décalage dû à une composante continue excessive peut apparaître. On peut expérimenter avec des fonctions plus agressives selon les besoins. Une pente généralement positive permet de mélanger le signal transformé à la source sans annulation de phase.

*distort* est utile comme processeur d'effets et est habituellement combiné avec de la réverbération et du chorus sur les bus d'effets. Cependant, on peut également l'utiliser comme effet dans un instrument unique.

## Exemples

Voici un exemple de l'opcode *distort*. Il utilise le fichier *distort.csd* [examples/distort.csd].

### Exemple 223. Exemple de l'opcode *distort*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o distort.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 32
nchnls  = 2
0dbfs   = 1

gifn ftgen 0,0, 257, 9, .5,1,270 ; define a sigmoid, or better
;gifn ftgen 0,0, 257, 9, .5,1,270,1.5,.33,90,2.5,.2,270,3.5,.143,90,4.5,.111,270

instr 1

kdist line 0, p3, 2 ; and over 10 seconds
asig poscil 0.3, 440, 1
aout distort asig, kdist, gifn ; gradually increase the distortion
outs aout, aout

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1
i 1 0 10
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Ecrit par Barry L. Vercoe pour Extended Csound et incorporé dans Csound5.

# distort1

distort1 — Distorsion par tangente hyperbolique modifiée.

## Description

Implémentation de la distorsion par tangente hyperbolique modifiée. *distort1* peut être utilisé pour générer une distorsion non-linéaire basée sur une modification de la fonction *tanh*.

$$\text{aout} = \frac{\exp(\text{asig} * (\text{shape1} + \text{pregain})) - \exp(\text{asig} * (\text{shape2} - \text{pregain}))}{\exp(\text{asig} * \text{pregain}) + \exp(-\text{asig} * \text{pregain})}$$

## Syntaxe

```
ares distort1 asig, kpregain, kpostgain, kshape1, kshape2[, imode]
```

## Initialisation

*imode* (Csound version 5.00 et suivantes seulement ; facultatif, 0 par défaut) -- met à l'échelle *kpregain*, *kpostgain*, *kshape1* et *kshape2* pour une utilisation avec des signaux audio entre -32768 et 32768 (*imode*=0), entre -0dbfs et 0dbfs (*imode*=1), ou désactive la mise à l'échelle de *kpregain* et de *kpostgain* et pondère *kshape1* par *kpregain* et *kshape2* par *-kpregain* (*imode*=2).

## Exécution

*asig* -- est le signal d'entrée.

*kpregain* -- détermine le gain appliqué au signal avant la distorsion. Une valeur de 1 donne une légère distorsion.

*kpostgain* -- détermine le gain appliqué au signal après la distorsion.

*kshape1* -- détermine la forme de la partie positive de la courbe. Une valeur de 0 donne un palier, de petites valeurs positives donnent une forme pentue.

*kshape2* -- détermine la forme de la partie négative de la courbe.

## Exemples

Voici un exemple de l'opcode *distort1*. Il utilise le fichier *distort1.csd* [examples/distort1.csd].

### Exemple 224. Exemple de l'opcode *distort1*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```

-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o distort1.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gadist init 0

instr 1
  iamp = p4
  ifgc = cpspch(p5)
  asig pluck iamp, ifgc, ifgc, 0, 1
  gadist = gadist + asig
endin

instr 50
  kpre init p4
  kpost init p5
  kshap1 init p6
  kshap2 init p7
  aout distort1 gadist, kpre, kpost, kshap1, kshap2, 1

  outs aout, aout

  gadist = 0
endin

</CsInstruments>
<CsScore>

; Sta Dur Amp Pitch
i1 0.0 3.0 0.5 6.00
i1 0.5 2.5 0.5 7.00
i1 1.0 2.0 0.5 7.07
i1 1.5 1.5 0.5 8.00

; Sta Dur PreGain PostGain Shape1 Shape2
i50 0 4 2 .5 0 0
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Hans Mikelson  
 Décembre 1998

Nouveau dans la version 3.50 de Csound

# divz

divz — Division protégée de deux nombres.

## Syntaxe

```
ares divz xa, xb, ksubst  
ires divz ia, ib, isubst  
kres divz ka, kb, ksubst
```

## Description

Division protégée de deux nombres.

## Initialisation

Lorsque  $b$  est différent de zéro, le résultat reçoit la valeur de  $a / b$  ; si  $b$  est égal à zéro, le résultat prend la valeur de *subst*.

## Exemples

Voici un exemple de l'opcode divz. Il utilise le fichier *divz.csd* [examples/divz.csd].

### Exemple 225. Exemple de l'opcode divz.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
; Audio out   Audio in  
-odac        -iadc      ;;RT audio I/O  
; For Non-realtime ouput leave only the line below:  
; -o divz.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
; Initialize the global variables.  
sr = 44100  
kr = 4410  
ksmps = 10  
nchnls = 1  
  
; Instrument #1.  
instr 1  
; Define the numbers to be divided.  
ka init 200  
; Linearly change the value of kb from 200 to 0.  
kb line 0, p3, 200  
; If a "divide by zero" error occurs, substitute -1.  
ksubst init -1  
  
; Safely divide the numbers.
```

```

kresults divz ka, kb, ksubst

; Print out the results.
printks "%f / %f = %f\\n", 0.1, ka, kb, kresults
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme :

```

200.000000 / 0.000000 = -1.000000
200.000000 / 19.999887 = 10.000056
200.000000 / 40.000027 = 4.999997

```

## Voir aussi

*=, init, tival*

## Crédits

Auteur : John ffitch d'après une idée de Barry L. Vercoe

Exemple écrit par Kevin Conder.

# doppler

**doppler** — Une méthode rapide et robuste d'approximation de propagation du son, produisant un effet Doppler convaincant sans résolution d'équations.

## Description

Opcodes du greffon doppler.

Une méthode rapide et robuste d'approximation de propagation du son, produisant un effet Doppler convaincant sans résolution d'équations. La méthode calcule des décalages de fréquence basés sur la lecture d'une ligne à retard en entrée avec un retard calculé à partir de la distance entre la source et le microphone, et de la vitesse du son. Il faut une instance de l'opcode pour chaque dimension de l'espace dans lequel le son évolue. Si la source sonore se déplace à vitesse constante depuis l'avant du microphone jusqu'à l'arrière du microphone en passant par le microphone, la sortie sera transposée en fréquence à une fréquence constante au-dessus de la fréquence de la source durant l'approche de la source, puis basculera de manière discontinue sous la fréquence de la source à une fréquence constante comme la source s'éloigne du microphone. Si la source sonore passe à vitesse constante en un point situé sur un côté du microphone, la vitesse du changement de position ne sera pas constante et l'on entendra le décalage en fréquence typique de l'effet Doppler familier d'une sirène ou d'un moteur approchant et s'éloignant le long d'une route située sur le côté de l'auditeur.

## Syntaxe

```
ashifted doppler asource, ksourceposition, kmicposition [, isoundspeed, ifiltercutoff]
```

## Initialisation

*isoundspeed* (facultatif, 340.29 par défaut) -- Vitesse du son en mètres/seconde.

*ifiltercutoff* (facultatif, 6 par défaut) -- taux de mise à jour du filtre de lissage de la position, en Hz.

## Exécution

*asource* -- Signal d'entrée de la source sonore.

*ksourceposition* -- Position de la source sonore en mètres. La distance entre la source et le micro ne doit pas changer plus vite qu'environ 3/4 de la vitesse du son.

*kmicposition* -- Position du microphone enregistreur en mètres. of the recording microphone in meters. La distance entre la source et le micro ne doit pas changer plus vite qu'environ 3/4 de la vitesse du son.

## Exemples

Voici un exemple de l'opcode doppler. Il utilise le fichier *doppler.csd* [examples/doppler.csd].

### Exemple 226. Exemple de l'opcode doppler.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime output leave only the line below:
; -o doppler.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 48000
ksmps = 128
nchnls = 2
0dbfs = 1

instr 1

iattack      init      0.05
irelease     init      0.05
isustain     init      p3
p3           init      iattack + isustain + irelease
kdamping     linseg    0.0, iattack, 1.0, isustain, 1.0, irelease, 0.0
kmic         init      4
              ; Position envelope, with a changing rate of change of position.
;           transeg a  dur  ty b      dur  ty c      dur  ty d
kposition    transeg  4, p3*.4, 0, 120,  p3*.3, -3, 50,  p3*.3, 2, 4
ismoothinghz init      6
ispeedofsound init     340.29
asignal      vco2      0.5, 110
aoutput      doppler   asignal, kposition, kmic, ispeedofsound, ismoothinghz
              outs      aoutput*kdamping, aoutput * kdamping

endin

</CsInstruments>
<CsScore>

i1 0.0 20
e1
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur de l'algorithme : Peter Brinkmann  
Auteur de l'opcode : Michael Gogins  
Janvier 2010

Nouveau dans la version 5.11 de Csound.



# dot

dot — Calcule le produit scalaire de deux tableaux.

## Description

Prend deux tableaux numériques (taux-k ou i) et calcule le produit scalaire.

## Syntaxe

```
kres/iresdot karr1[]/iarr1[], karr2[]/iarr2[] (k- or i-arrays )
```

## Exemples

Voici un exemple de l'opcode dot. Il utilise le fichier *dota.csd* [examples/dot.csd].

### Exemple 227. Exemple de l'opcode dot.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-n
</CsOptions>
<CsInstruments>

instr 1
  kArr1[] fillarray 1,3,2,7,4
  kArr2[] fillarray 2,3,1,2,5
  kd dot kArr1,kArr2
  printk2 kd
  turnoff
endin

</CsInstruments>
<CsScore>
i1 0 1
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*sumarray product*

## Crédits

Auteur : Victor Lazzarini

Nouveau dans la version 6.09

# downsamp

downsamp — Modifie un signal par sous-échantillonnage.

## Description

Modifie un signal par sous-échantillonnage.

## Syntaxe

```
kres downsamp asig [, iwlen]
```

## Initialisation

*iwlen* (facultatif) -- longueur en échantillons de la fenêtre sur laquelle est prise la valeur moyenne du signal audio pour déterminer une valeur sous-échantillonnée. La longueur maximale est *ksmps* ; 0 et 1 impliquent pas de fenêtre de moyenne. La valeur par défaut est 0.

## Exécution

*downsamp* convertit un signal audio en signal de contrôle par sous-échantillonnage. Il produit une *kval* pour chaque période audio de contrôle. La fenêtre optionnelle invoque un simple procédé de moyenne pour supprimer le repliement.

## Exemples

Voici un exemple de l'opcode *downsamp*. Il utilise le fichier *downsamp.csd* [examples/downsamp.csd].

### Exemple 228. Exemple de l'opcode *downsamp*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o downsamp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq = cpspch(p4)
ain diskin2 "beats.wav", 1
aenv follow ain, .001 ;take the amplitude every 1/1000th of a second
alow tone aenv, 25 ;lowpass-filter (25 Hz) for a clean signal
```

```

kenv downsamp allow
asig pluck kenv, ifrq, 15, 0, 1
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 2 9
i 1 + . 7
i 1 + . 5

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*diff, integ, interp, samphold, upsamp*

# dripwater

dripwater — Modèle semi-physique d'une goutte d'eau.

## Description

*dripwater* est un modèle semi-physique d'une goutte d'eau. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntaxe

```
ares dripwater kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \  
    [, ifreq1] [, ifreq2]
```

## Initialisation

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 10.

*idamp* (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

$$\text{damping\_amount} = 0,996 + (\text{idamp} * 0,002)$$

La valeur par défaut de *damping\_amount* est 0,996 ce qui signifie que la valeur par défaut de *idamp* est 0. Le maximum de *damping\_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 2,0.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale. Rasmus Ekman proposee un intervalle de 1,4 à 1,75. Il suggère aussi une valeur maximale de 1,9 au lieu de la limite théorique de 2,0.

*imaxshake* (facultatif, 0 par défaut) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

*ifreq* (facultatif) -- la fréquence de résonance principale. La valeur par défaut est 450.

*ifreq1* (facultatif) -- la première fréquence de résonance. La valeur par défaut est 600.

*ifreq2* (facultatif) -- La seconde fréquence de résonance. La valeur par défaut est 750.

## Exécution

*kamp* -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

## Exemples

Voici un exemple de l'opcode *dripwater*. Il utilise le fichier *dripwater.csd* [examples/dripwater.csd].

## Exemple 229. Exemple de l'opcode dripwater.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o dripwater.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

imaxshake = p4
ifreq      = p5
ifreq1     = p6
ifreq2     = p7

;low amplitude
adrp dripwater .1, 0.09, 10, .9, imaxshake, ifreq, ifreq1, ifreq2
asig clip adrp, 2, 0.9 ; avoid drips that drip too loud
outs asig, asig

endin
</CsInstruments>
<CsScore>

{100 CNT

i1 [0.1 * $CNT] 0.5 0.5 430 1000 800

}

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*bamboo, guiro, sleighbells, tambourine*

## Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)

Adapté par John ffitich

Université de Bath, Codemist Ltd.

Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

# dssiactivate

dssiactivate — Active ou désactive un greffon DSSI ou LADSPA.



## Note

Greffon : nécessite le greffon *dssi4cs*

## Syntaxe

```
dssiactivate ihandle, ktoggle
```

## Description

Opcode du greffon dssi4cs.

*dssiactivate* est utilisé pour activer ou désactiver un greffon DSSI ou LADSPA. Il appelle les fonctions *activate()* et *deactivate()* du greffon si elles sont disponibles.

## Initialisation

*ihandle* - le numéro qui identifie le greffon, généré par *dssiinit*.

## Exécution

*ktoggle* - Choix entre l'activation (*ktoggle* = 1) et la désactivation (*ktoggle* = 0).

*dssiactivate* est utilisé pour activer ou désactiver des greffons si ceux-ci proposent cette option. Cela peut aider à maintenir le traitement CPU dans certains cas. Pour être consistant, tous les greffons doivent être activés pour produire du son. Un greffon inactif reste silencieux.

En fonction de l'implémentation du greffon, cela peut causer des interruptions dans le processus audio en temps réel. Il faut donc l'utiliser avec précaution.

*dssiactivate* pouvant provoquer des interruptions du flux audio en temps réel, il est recommandé de charger tous les greffons que l'on veut utiliser avant l'exécution.



## Avertissement

Noter que même si un greffon ne possède pas les fonctions *activate()* et *deactivate()*, *dssiactivate* doit être appelé pour que le greffon produise du son.

## Exemples

Voici un exemple de l'opcode *dssiactivate*. Il utilise le fichier *dssiactivate.csd* [exemples/dssiactivate.csd].

### Exemple 230. Exemple de l'opcode dssiactivate.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
```

```

<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o dssiactivate.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

gihandle dssiinit "caps.so", 19, 1 ; = mono phaser and
gaout init 0 ; verbose about all ports

instr 1 ; activate DSSI

ktoggle = p4
dssiactivate gihandle, ktoggle
endin

instr 2

ain1 diskin2 "beats.wav", 1,0,1 ; loop
ain1 = ain1*.5
outs ain1, ain1
gaout = gaout+ain1
endin

instr 3

dssictls gihandle, 0, 1, 1 ; range -1 to 1
dssictls gihandle, 1, 2, 1 ; rate 0 to 10
dssictls gihandle, 2, .8, 1 ; depth 0 to 1
dssictls gihandle, 3, 3, 1 ; spread 0 to 3.14
dssictls gihandle, 4, .9, 1 ; feedback 0 to 0.999

endin

instr 4

aout1 dssiaudio gihandle, gaout ;get beats.wav, mono out
outs aout1,aout1

gaout = 0
endin

</CsInstruments>
<CsScore>
i 1 0 4 1
i 1 + . 0
i 1 + . 1
i 1 + . 0
i 1 + . 1
i 2 1 20
i 3 1 20
i 4 0 20

e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

2005

Par Andrés Cabrera

Utilise du code provenant du sdk de LADSPA par Richard Furse.



# dssiaudio

dssiaudio — Traitement audio au moyen d'un greffon LADSPA ou DSSI.



## Note

Greffon : nécessite le greffon *dssi4cs*

## Syntaxe

```
[aout1, aout2, ..., aout9] dssiaudio ihandle, [ain1, ain2, ..., ain9]
```

## Description

Opcode du greffon dssi4cs.

*dssiaudio* génère des données audio en traitant un signal d'entrée dans un greffon LADSPA ou DSSI.

## Initialisation

*ihandle* - le numéro qui identifie le greffon, généré par *dssiinit*.

## Exécution

*aout1, aout2, etc* - sortie audio générée par le greffon.

*ain1, ain2, etc* - entrée audio fournie au greffon pour traitement.

*dssiaudio* exécute un greffon sur la source audio et produit une sortie audio. Actuellement il peut y avoir jusqu'à quatre entrées et sorties. Il faut fournir un signal à toutes les entrées audio du greffon, sinon le résultat peut être imprévisible. Si le greffon n'a pas d'entrée (par exemple un générateur de bruit), il faut quand même fournir au moins une variable d'entrée qui sera ignorée avec un message.

Il ne faut exécuter qu'un seul *dssiaudio* à la fois par greffon, sinon des résultats étranges peuvent survenir.

## Exemples

Voici un exemple de l'opcode *dssiaudio*. Il utilise le fichier *dssiaudio.csd* [examples/dssiaudio.csd].

### Exemple 231. Exemple de l'opcode *dssiaudio*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o DSSIplay_mono.wav -W ;;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

  sr = 44100
  ksmps = 32
  nchnls = 2

  gihandle dssiinit "caps.so", 19, 1 ; = mono phaser and
  gaout    init 0 ; verbose about all ports

  instr 1 ; activate DSSI

  dssiactivate gihandle, 1
  endin

  instr 2
  ain1 disk2 "beats.wav", 1,0,1 ; loop

  gaout = gaout+(ain1*.5)
  endin

  instr 3

  dssictls gihandle, 0, .8, 1 ; range -1 to 1
  dssictls gihandle, 1, .05, 1 ; rate 0 to 10
  dssictls gihandle, 2, .8, 1 ; depth 0 to 1
  dssictls gihandle, 3, 2, 1 ; spread 0 to 3.14
  dssictls gihandle, 4, .7, 1 ; feedback 0 to 0.999

  endin

  instr 4

  aout1 dssiaudio gihandle, gaout ;get beats.wav, mono out
  outs aout1,aout1

  gaout = 0

  endin
</CsInstruments>
<CsScore>
i 1 0 20
i 2 1 20
i 3 1 20
i 4 0 20

e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

2005

Par Andrés Cabrera

Utilise du code provenant du sdk de LADSPA par Richard Furse.

# dssictls

dssictls — Envoie une information de contrôle à un greffon LADSPA ou DSSI.



## Note

Greffon : nécessite le greffon *dssi4cs*

## Syntaxe

```
dssictls ihandle, iport, kvalue, ktrigger
```

## Description

Opcode du greffon dssi4cs.

*dssictls* envoie des valeurs de contrôle sur le port de contrôle d'un greffon.

## Initialisation

*ihandle* - le numéro qui identifie le greffon, généré par *dssiinit*.

*iport* - numéro du port de contrôle

## Exécution

*kvalue* - valeur à assigner au port.

*ktrigger* - détermine si l'information de contrôle doit être envoyée (*ktrigger* = 1) ou non. Utile pour réduire l'information de contrôle, si l'on génère *ktrigger* avec *metro*.

*dssictls* envoie de l'information de contrôle sur le port de contrôle d'un greffon LADSPA ou DSSI. Les ports de contrôle valides et les valeurs autorisées sont donnés par *dssiinit*. Si l'on utilise des valeurs hors limite, il peut y avoir un comportement indéfini.

## Exemples

Voici un exemple de l'opcode dssictls. Il utilise le fichier *dssictls.csd* [examples/dssictls.csd].

### Exemple 232. Exemple de l'opcode dssictls.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o dssictls.wav -W ;;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

gihandle dssiinit "caps.so", 1, 1 ; = equaliser and
gaoutl init 0 ; verbose about all ports
gaoutr init 0

instr 1 ; activate DSSI

dssiactivate gihandle, 1
endin

instr 2
ainl disk2 "beats.wav", 1,0,1 ; loop

gaoutl = gaoutl+(ainl*.1) ; temper input
gaoutr = gaoutr+(ainl*.1)
endin

instr 3

dssictls gihandle, 2, -48, 1 ; 31 Hz range -48 to 24
dssictls gihandle, 3, -48, 1 ; 63 Hz range -48 to 24
dssictls gihandle, 4, -48, 1 ; 125 Hz range -48 to 24
dssictls gihandle, 5, 20, 1 ; 250 Hz range -48 to 24
dssictls gihandle, 6, -48, 1 ; 500 Hz range -48 to 24
dssictls gihandle, 7, -48, 1 ; 1 kHz range -48 to 24
dssictls gihandle, 8, -48, 1 ; 2 kHz range -48 to 24
dssictls gihandle, 9, 24, 1 ; 4 kHz range -48 to 24
dssictls gihandle, 10, 24, 1 ; 8 kHz range -48 to 24
dssictls gihandle, 11, 24, 1 ; 16 kHz range -48 to 24

endin

instr 4

aout1, aout2 dssiaudio gihandle, gaoutl, gaoutr ;get beats.wav, mono out
outs aout1,aout2

gaoutl = 0
gaoutr = 0
endin

</CsInstruments>
<CsScore>
i 1 0 20
i 2 1 20
i 3 1 20
i 4 0 20

e
</CsScore>
</CsSoundSynthesizer>

```

## Crédits

2005

Par Andrés Cabrera

Utilise du code provenant du sdk de LADSPA par Richard Furse.

# dssiinit

dssiinit — Charge un greffon DSSI ou LADSPA.



## Note

Greffon : nécessite le greffon *dssi4cs*

## Syntaxe

```
ihandle dssiinit ilibraryname, igreffondex [, iverbose]
```

## Description

Opcode du greffon dssi4cs.

*dssiinit* est utilisé pour charger en mémoire un greffon DSSI ou LADSPA pour une utilisation avec les autres opcodes dssi4cs. On peut utiliser des effets LADSPA ainsi que des instruments DSSI.

## Initialisation

*ihandle* - le numéro qui identifie le greffon, à passer au autres opcodes dssi4cs.

*ilibraryname* - le nom du fichier .so (objet partagé) à charger.

*igreffondex* - l'index du greffon à utiliser, supérieur ou égal à zéro.

*iverbose* (facultatif) - montre l'information et les paramètres du greffon lors du chargement. (1 par défaut)

*dssiinit* recherche *ilibraryname* dans les chemins définis par LADSPA\_PATH et DSSI\_PATH. Une de ces variables doit être définie sinon *dssiinit* retourne une erreur. Les bibliothèques LADSPA et DSSI peuvent contenir plus d'un greffon, ceux-ci étant référencés par un index. *dssiinit* tente alors de trouver le greffon d'index *igreffondex* dans la bibliothèque et le charge en mémoire s'il le trouve. Pour savoir quels greffons sont disponibles et quels sont leurs numéros d'index, on peut utiliser *dssilist*.

Si *iverbose* est différent de 0 (par défaut), une information sur les caractéristiques détaillées du greffon et sur ses ports est affichée. Cette information est importante pour les opcodes comme *dssictls*.

Les greffons sont inactifs par défaut. Il *\*faut\** donc utiliser *dssiactivate* pour que le greffon produise du son. Ceci est obligatoire, même si le greffon ne possède pas de fonction activate().

*dssiinit* pouvant provoquer des interruptions du flux audio en temps réel, il est recommandé de charger tous les greffons que l'on veut utiliser avant l'exécution.

## Exemples

Voici un exemple de l'opcode dssinit. Il utilise le fichier *dssiinit.csd* [examples/dssiinit.csd].

**Exemple 233. Exemple de l'opcode dssinit. (Penser à changer le nom de la bibliothèque)**

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime output leave only the line below:
; -o dssiinit.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

gihandle dssiinit "g2reverb.so", 0, 1
gaout    init 0

instr 1 ; activate DSSI

dssiactivate gihandle, 1
endin

instr 2

ain1 diskin2 "beats.wav", 1

gaout = gaout+(ain1*.3)
endin

instr 3

dssictls gihandle, 4, 100, 1 ; room 10 to 150
dssictls gihandle, 5, 10, 1 ; reverb time 1 to 20
dssictls gihandle, 6, .5, 1 ; input bandwidth 0 to 1
dssictls gihandle, 7, .25, 1 ; damping 0 to 1
dssictls gihandle, 8, 0, 1 ; dry -80 to 0
dssictls gihandle, 9, -10, 1 ; reflections -80 to 0
dssictls gihandle, 10, -15, 1 ; rev. tail -80 to 0
endin

instr 4

aout1, aout2 dssiaudio gihandle, gaout, gaout ;get beats.wav and
           outs aout1,aout2 ; stereo DSSI plugin

gaout = 0
endin
</CsInstruments>
<CsScore>
i 1 0 2
i 2 1 10
i 3 1 10
i 4 0 10
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

2005

Par Andrés Cabrera

Utilise du code provenant du sdk de LADSPA par Richard Furse.

# dssilist

dssilist — Liste tous les greffons DSSI et LADSPA disponibles.



## Note

Greffon : nécessite le greffon *dssi4cs*

## Syntaxe

`dssilist`

## Description

Opcode du greffon dssi4cs.

*dssilist* vérifie les variables DSSI\_PATH et LADSPA\_PATH et liste tous les greffons disponibles dans les bibliothèques accessibles par ces chemins.

Les bibliothèques LADSPA et DSSI peuvent contenir plus d'un greffon, ceux-ci devant être référencés par l'index fournit par *dssilist*.

Cet opcode produisant un long listing qui peut interrompre la sortie audio en temps réel, il vaut mieux l'utiliser au début de l'exécution.

## Exemples

Voici un exemple de l'opcode dssilist. Il utilise le fichier *dssilist.csd* [exemples/dssilist.csd].

**Exemple 234. Exemple de l'opcode dssilist. (RPenser à changer le nom de la bibliothèque)**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out
-odac

</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1 ; list all DSSI and LADSPA plugins

dssilist

endin
</CsInstruments>
<CsScore>
i 1 0 0
```

```
e  
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

2005

Par Andrés Cabrera

Utilise du code provenant du sdk de LADSPA par Richard Furse.



# dumpk

**dumpk** — Ecrit périodiquement la valeur d'un signal de contrôle de l'orchestre dans un fichier externe.

## Description

Ecrit périodiquement la valeur d'un signal de contrôle de l'orchestre dans un fichier externe, dans un format spécifique.

## Syntaxe

```
dumpk ksig, ifilename, iformat, iprd
```

## Initialisation

*ifilename* -- chaîne de caractères (entre guillemets, espaces autorisés) contenant le nom du fichier externe. Peut être un nom de chemin complet avec un répertoire cible ou un simple nom de fichier à créer dans le répertoire courant.

*iformat* -- spécifie le format des données de sortie :

- 1 = caractères signés sur 8 bit (les 8 bit d'ordre supérieur d'un entier sur 16 bit)
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers longs en ASCII
- 8 = flottants en ASCII (2 positions décimales)

Noter que les sorties A-law et U-law ne sont pas disponibles, et que tous les formats sauf les deux derniers sont binaires. Le fichier de sortie ne contient pas d'information d'en-tête.

*iprd* -- la période de la sortie *ksig* en secondes, arrondie à la période de contrôle de l'orchestre la plus proche. Une valeur de 0 implique une période de contrôle (le minimum imposé), qui créera un fichier de sortie échantillonné au taux de contrôle de l'orchestre.

## Exécution

*ksig* -- un signal au taux de contrôle

Cet opcode permet de sauvegarder la valeur d'un signal généré au taux de contrôle dans un fichier externe. Le fichier ne contient pas d'information auto-descriptive en en-tête. Mais il contient une suite temporelle échantillonnée régulièrement, appropriée pour une entrée ultérieure ou une analyse. Il peut y avoir n'importe quel nombre d'opcodes *dumpk* dans un instrument ou dans un orchestre mais chacun doit écrire dans un fichier différent.

## Exemples

Voici un exemple de l'opcode *dumpk*. Il utilise le fichier *dumpk.csd* [exemples/dumpk.csd]. Noter que l'exemple doit être exécuté depuis un répertoire dans lequel l'écriture est autorisée.

### Exemple 235. Exemple de l'opcode dumpk.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac        -iadc    ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o dumpk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 20
nchnls = 1

; By Andres Cabrera 2008

instr 1
; Write fibonacci numbers to file "fibonacci.txt"
; as ascii long integers (mode 7), using the orchestra's
; control rate (iprd = 0)

knumber init 0
koldnumber init 1
ktrans init 1
ktrans = knumber
knumber = knumber + koldnumber
koldnumber = ktrans
dumpk knumber, "fibonacci.txt", 7, 0
printk2 knumber
endin

</CsInstruments>
<CsScore>

;Write to the file for 1 second. Since control rate is 20, 20 values will be written
i 1 0 1

</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode dumpk. Il utilise le fichier *dumpk-2.csd* [examples/dumpk-2.csd].

### Exemple 236. Exemple 2 de l'opcode dumpk.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac        ;;realtime audio out
;-iadc       ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o dumpk-2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 2^10, 10, 1

instr 1 ;writes a control signal to a file
kfreq randh 100, 1, 2, 1, 500 ;generates one random number between 400 and 600 per second
dumpk kfreq, "dumpk.txt", 8, 1 ;writes the control signal
printk 1, kfreq ;prints it
endin

instr 2 ;reads the file written by instr 1
kfreq readk "dumpk.txt", 8, 1
printk 1, kfreq ;prints it
aout poscil .2, kfreq, giSine
outs aout, aout
endin

</CsInstruments>
<CsScore>
i 1 0 5
i 2 5 5
e
</CsScore>
</CsoundSynthesizer>

```

La sortie contiendra des lignes comme celles-ci :

WARNING: Seeding from current time 683384022

```

i 1 time 1.00033: 463.64510
i 1 time 2.00000: 463.64510
i 1 time 3.00000: 483.14200
i 1 time 4.00000: 567.55973
i 1 time 5.00000: 576.37060
i 1 time 6.00000: 460.66550

i 2 time 6.00033: 463.64510
i 2 time 7.00000: 463.64510
i 2 time 8.00000: 483.14200
i 2 time 9.00000: 567.55970
i 2 time 10.00000: 576.37060
i 2 time 11.00000: 460.66550

```

## Voir aussi

*dumpk2, dumpk3, dumpk4, readk, readk2, readk3, readk4*

## Crédits

Par : John ffitich et Barry L. Vercoe

1999 ou avant

# dumpk2

`dumpk2` — Ecrit périodiquement les valeurs de deux signaux de contrôle de l'orchestre dans un fichier externe.

## Description

Ecrit périodiquement les valeurs de deux signaux de contrôle de l'orchestre dans un fichier externe, dans un format spécifique.

## Syntaxe

```
dumpk2 ksig1, ksig2, ifilename, iformat, iprd
```

## Initialisation

*ifilename* -- chaîne de caractères (entre guillemets, espaces autorisés) contenant le nom du fichier externe. Peut être un nom de chemin complet avec un répertoire cible ou un simple nom de fichier à créer dans le répertoire courant.

*iformat* -- spécifie le format des données de sortie :

- 1 = caractères signés sur 8 bit (les 8 bit d'ordre supérieur d'un entier sur 16 bit)
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers longs en ASCII
- 8 = flottants en ASCII (2 positions décimales)

Noter que les sorties A-law et U-law ne sont pas disponibles, et que tous les formats sauf les deux derniers sont binaires. Le fichier de sortie ne contient pas d'information d'en-tête.

*iprd* -- la période de la sortie *ksig* en secondes, arrondie à la période de contrôle de l'orchestre la plus proche. Une valeur de 0 implique une période de contrôle (le minimum imposé), qui créera un fichier de sortie échantillonné au taux de contrôle de l'orchestre.

## Exécution

*ksig1*, *ksig2* -- signaux au taux de contrôle.

Cet opcode permet de sauvegarder les valeurs de deux signaux générés au taux de contrôle dans un fichier externe. Le fichier ne contient pas d'information auto-descriptive en en-tête. Mais il contient une suite temporelle échantillonnée régulièrement, appropriée pour une entrée ultérieure ou une analyse. Il peut y avoir n'importe quel nombre d'opcodes *dumpk2* dans un instrument ou dans un orchestre mais chacun doit écrire dans un fichier différent.

## Exemples

Voici un exemple de l'opcode `dumpk2`. Il utilise le fichier `dumpk2.csd` [exemples/dumpk2.csd]. Noter que l'exemple doit être exécuté depuis un répertoire dans lequel l'écriture est autorisée.

### Exemple 237. Exemple de l'opcode `dumpk2`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o dumpk2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 2^10, 10, 1

instr 1 ;writes two control signals to a file
kfreq randh 100, 1, 2, 1, 500 ;generates one random number between 400 and 600 per second
kdb randh 12, 1, 2, 1, -12 ;amplitudes in dB between -24 and 0
dumpk2 kfreq, kdb, "dumpk2.txt", 8, 1 ;writes the control signals
prints "WRITING:\n"
printks "kfreq = %f, kdb = %f\n", 1, kfreq, kdb ;prints them
endin

instr 2 ;reads the file written by instr 1
kf,kdb readk2 "dumpk2.txt", 8, 1
prints "READING:\n"
printks "kfreq = %f, kdb = %f\n", 1, kf, kdb ;prints again
kdb .1 ;smoothing amp transition
aout poscil ampdb(kdb), kf, giSine
outs aout, aout
endin

</CsInstruments>
<CsScore>
i 1 0 5
i 2 5 5
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
kfreq = 429.202551, kdb = -20.495694
kfreq = 429.202551, kdb = -20.495694
kfreq = 407.275258, kdb = -23.123776
kfreq = 475.264472, kdb = -9.300846
kfreq = 569.979181, kdb = -7.315527
kfreq = 440.103457, kdb = -0.058331

kfreq = 429.202600, kdb = -20.495700
```

```
kfreq = 429.202600, kdb = -20.495700  
kfreq = 407.275300, kdb = -23.123800  
kfreq = 475.264500, kdb = -9.300800  
kfreq = 569.979200, kdb = -7.315500  
kfreq = 440.103500, kdb = -0.058300
```

## Voir aussi

*dumpk, dumpk3, dumpk4, readk, readk2, readk3, readk4*

## Crédits

Par : John ffitch et Barry L. Vercoe

1999 ou avant

# dumpk3

`dumpk3` — Ecrit périodiquement les valeurs de trois signaux de contrôle de l'orchestre dans un fichier externe.

## Description

Ecrit périodiquement les valeurs de trois signaux de contrôle de l'orchestre dans un fichier externe, dans un format spécifique.

## Syntaxe

```
dumpk3 ksig1, ksig2, ksig3, ifilename, iformat, iprd
```

## Initialisation

*ifilename* -- chaîne de caractères (entre guillemets, espaces autorisés) contenant le nom du fichier externe. Peut être un nom de chemin complet avec un répertoire cible ou un simple nom de fichier à créer dans le répertoire courant.

*iformat* -- spécifie le format des données de sortie :

- 1 = caractères signés sur 8 bit (les 8 bit d'ordre supérieur d'un entier sur 16 bit)
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers longs en ASCII
- 8 = flottants en ASCII (2 positions décimales)

Noter que les sorties A-law et U-law ne sont pas disponibles, et que tous les formats sauf les deux derniers sont binaires. Le fichier de sortie ne contient pas d'information d'en-tête.

*iprd* -- la période de la sortie *ksig* en secondes, arrondie à la période de contrôle de l'orchestre la plus proche. Une valeur de 0 implique une période de contrôle (le minimum imposé), qui créera un fichier de sortie échantillonné au taux de contrôle de l'orchestre.

## Exécution

*ksig1*, *ksig2*, *ksig3* -- signaux au taux de contrôle

Cet opcode permet de sauvegarder les valeurs de trois signaux générés au taux de contrôle dans un fichier externe. Le fichier ne contient pas d'information auto-descriptive en en-tête. Mais il contient une suite temporelle échantillonnée régulièrement, appropriée pour une entrée ultérieure ou une analyse. Il peut y avoir n'importe quel nombre d'opcodes *dumpk3* dans un instrument ou dans un orchestre mais chacun doit écrire dans un fichier différent.

## Exemples

Voici un exemple de l'opcode `dumpk3`. Il utilise le fichier `dumpk3.csd` [examples/dumpk3.csd]. Noter que l'exemple doit être exécuté depuis un répertoire dans lequel l'écriture est autorisée.

### Exemple 238. Exemple de l'opcode `dumpk3`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o dumpk3.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 2^10, 10, 1

instr 1 ;writes three control signals to a file
kfreq randh 100, 1, 2, 1, 500 ;generates one random number between 400 and 600 per second
kdb randh 12, 1, 2, 1, -12 ;amplitudes in dB between -24 and 0
kpan randh .5, 1, 2, 1, .5 ;panning between 0 and 1
dumpk3 kfreq, kdb, kpan, "dumpk3.txt", 8, 1 ;writes the control signals
prints "WRITING:\n"
printks "kfreq = %f, kdb = %f, kpan = %f\n", 1, kfreq, kdb, kpan ;prints them
endin

instr 2 ;reads the file written by instr 1
kf,kdb,kp readk3 "dumpk3.txt", 8, 1
prints "READING:\n"
printks "kfreq = %f, kdb = %f, kpan = %f\n", 1, kf, kdb, kp ;prints again
kdb lineto kdb, .1 ;smoothing amp transition
kp lineto kp, .1 ;smoothing pan transition
aout poscil ampdb(kdb), kf, giSine
aL, aR pan2 aout, kp
outs aL, aR
endin

</CsInstruments>
<CsScore>
i 1 0 5
i 2 5 5
e
</CsScore>
</CsoundSynthesizer>
```

La sortie contiendra des lignes comme celles-ci :

```
WRITING:
kfreq = 473.352855, kdb = -15.197657, kpan = 0.366764
kfreq = 473.352855, kdb = -15.197657, kpan = 0.366764
kfreq = 441.426368, kdb = -19.026206, kpan = 0.207327
kfreq = 452.965140, kdb = -21.447486, kpan = 0.553270
```



```
kfreq = 585.106328, kdb = -11.903852, kpan = 0.815665  
kfreq = 482.056760, kdb = -4.046744, kpan = 0.876537
```

READING:

```
kfreq = 473.352900, kdb = -15.197700, kpan = 0.366800  
kfreq = 473.352900, kdb = -15.197700, kpan = 0.366800  
kfreq = 441.426400, kdb = -19.026200, kpan = 0.207300  
kfreq = 452.965100, kdb = -21.447500, kpan = 0.553300  
kfreq = 585.106300, kdb = -11.903900, kpan = 0.815700  
kfreq = 482.056800, kdb = -4.046700, kpan = 0.876500
```

## Voir aussi

*dumpk, dumpk2, dumpk4, readk, readk2, readk3, readk4*

## Crédits

Par : John ffitch et Barry L. Vercoe

1999 ou avant

# dumpk4

`dumpk4` — Ecrit périodiquement les valeurs de quatre signaux de contrôle de l'orchestre dans un fichier externe.

## Description

Ecrit périodiquement les valeurs de quatre signaux de contrôle de l'orchestre dans un fichier externe, dans un format spécifique.

## Syntaxe

```
dumpk4 ksig1, ksig2, ksig3, ksig4, ifilename, iformat, iprd
```

## Initialisation

*ifilename* -- chaîne de caractères (entre guillemets, espaces autorisés) contenant le nom du fichier externe. Peut être un nom de chemin complet avec un répertoire cible ou un simple nom de fichier à créer dans le répertoire courant.

*iformat* -- spécifie le format des données de sortie :

- 1 = caractères signés sur 8 bit (les 8 bit d'ordre supérieur d'un entier sur 16 bit)
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers longs en ASCII
- 8 = flottants en ASCII (2 positions décimales)

Noter que les sorties A-law et U-law ne sont pas disponibles, et que tous les formats sauf les deux derniers sont binaires. Le fichier de sortie ne contient pas d'information d'en-tête.

*iprd* -- la période de la sortie *ksig* en secondes, arrondie à la période de contrôle de l'orchestre la plus proche. Une valeur de 0 implique une période de contrôle (le minimum imposé), qui créera un fichier de sortie échantillonné au taux de contrôle de l'orchestre.

## Exécution

*ksig1*, *ksig2*, *ksig3*, *ksig4* -- signaux au taux de contrôle

Cet opcode permet de sauvegarder les valeurs de quatre signaux générés au taux de contrôle dans un fichier externe. Le fichier ne contient pas d'information auto-descriptive en en-tête. Mais il contient une suite temporelle échantillonnée régulièrement, appropriée pour une entrée ultérieure ou une analyse. Il peut y avoir n'importe quel nombre d'opcodes *dumpk4* dans un instrument ou dans un orchestre mais chacun doit écrire dans un fichier différent.

## Exemples

Voici un exemple de l'opcode `dumpk4`. Il utilise le fichier `dumpk4.csd` [examples/dumpk4.csd]. Noter que l'exemple doit être exécuté depuis un répertoire dans lequel l'écriture est autorisée.

### Exemple 239. Exemple de l'opcode `dumpk4`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o dumpk4.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 2^10, 10, 1

instr 1 ;writes four control signals to a file
kcf      randh    950, 1, 2, 1, 1050 ;generates one random number between 100 and 2000 per second
kq       randh    10, 1, 2, 1, 11 ;generates another random number between 1 and 21 per second
kdb      randh    9, 1, 2, 1, -15 ;amplitudes in dB between -24 and -6
kpan     randh    .5, 1, 2, 1, .5 ;panning between 0 and 1
          dumpk4   kcf, kq, kdb, kpan, "dumpk4.txt", 8, 1 ;writes the control signals
          prints    "WRITING:\n"
          printks   "kcf = %f, kq = %f, kdb = %f, kpan = %f\n", 1, kcf, kq, kdb, kpan ;prints them
endin

instr 2 ;reads the file written by instr 1
kcf,kq,kdb,kp readk4 "dumpk4.txt", 8, 1
          prints    "READING:\n"
          printks   "kcf = %f, kq = %f, kdb = %f, kpan = %f\n", 1, kcf, kq, kdb, kp ;prints values
kdb      lineto    kdb, .1 ;smoothing amp transition
kp       lineto    kp, .1 ;smoothing pan transition
anoise   rand      ampdb(kdb), 2, 1
kbw      =         kcf/kq ;bandwidth of resonant filter
abp      reson     anoise, kcf, kbw
aout     balance   abp, anoise
aL, aR   pan2      aout, kp
          outs      aL, aR
endin

</CsInstruments>
<CsScore>
i 1 0 5
i 2 5 5
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

WRITING:

```
kcf = 1122.469723, kq = 11.762839, kdb = -14.313445, kpan = 0.538142
kcf = 1122.469723, kq = 11.762839, kdb = -14.313445, kpan = 0.538142
kcf = 1148.638412, kq = 12.040490, kdb = -14.061868, kpan = 0.552205
kcf = 165.796855, kq = 18.523179, kdb = -15.816977, kpan = 0.901528
kcf = 147.729960, kq = 13.071911, kdb = -11.924531, kpan = 0.982518
kcf = 497.430113, kq = 13.605512, kdb = -21.586611, kpan = 0.179229
```

READING:

WARNING: Seeding from current time 3308160476

```
kcf = 1122.469700, kq = 11.762800, kdb = -14.313400, kpan = 0.538100
kcf = 1122.469700, kq = 11.762800, kdb = -14.313400, kpan = 0.538100
kcf = 1148.638400, kq = 12.040500, kdb = -14.061900, kpan = 0.552200
kcf = 165.796900, kq = 18.523200, kdb = -15.817000, kpan = 0.901500
kcf = 147.730000, kq = 13.071900, kdb = -11.924500, kpan = 0.982500
kcf = 497.430100, kq = 13.605500, kdb = -21.586600, kpan = 0.179200
```

## Voir aussi

*dumpk, dumpk2, dumpk3, readk, readk2, readk3, readk4*

## Crédits

Par : John ffitch et Barry L. Vercoe

1999 ou avant

# duserrnd

duserrnd — Générateur de nombres aléatoires de distribution discrète définie par l'utilisateur.

## Description

Générateur de nombres aléatoires de distribution discrète définie par l'utilisateur.

## Syntaxe

```
aout duserrnd ktableNum
```

```
iout duserrnd itableNum
```

```
kout duserrnd ktableNum
```

## Initialisation

*itableNum* -- numéro d'une table contenant la fonction de la distribution aléatoire. Cette table est générée par l'utilisateur. Voir GEN40, GEN41 et GEN42. La longueur de la table peut être différente d'une puissance de 2.

## Exécution

*ktableNum* -- numéro d'une table contenant la fonction de la distribution aléatoire. Cette table est générée par l'utilisateur. Voir GEN40, GEN41 et GEN42. La longueur de la table peut être différente d'une puissance de 2.

*duserrnd* (Discrete USER-defined-distribution RaNDom generator) génère des nombres aléatoires selon une distribution aléatoire discrète créée par l'utilisateur. L'utilisateur peut créer l'histogramme de la distribution discrète au moyen de GEN41. Afin de créer cette table, on doit définir une quantité arbitraire de couples de nombres, le premier nombre de chaque paire représentant une valeur et le second représentant sa probabilité (voir GEN41 pour plus de détails).

Lorsqu'on l'utilise comme une fonction, le taux de génération dépend du type du taux de la variable d'entrée *XtableNum*. Dans ce cas, on peut l'insérer dans n'importe quelle formule. Le numéro de table peut varier au taux-k, ce qui permet de changer l'histogramme de la distribution durant l'exécution d'une note. *duserrnd* est destiné à être utilisé pour la génération de musique algorithmique.

On peut aussi utiliser *duserrnd* pour générer des valeurs suivant un ensemble d'intervalles de probabilités au moyen de fonctions de distribution générées par GEN42 (voir GEN42 pour plus de détails). Dans ce cas, si l'on veut simuler des intervalles continus, la longueur de la table *XtableNum* doit être raisonnablement grande car *duserrnd* ne fait pas d'interpolation entre les éléments de la table.

Pour un tutoriel sur les histogrammes et les fonctions de distribution aléatoires consulter :

- D. Lorrain. "A panoply of stochastic cannons". In C. Roads, ed. 1989. Music machine. Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode *duserrnd*. Il utilise le fichier *duserrnd.csd* [examples/duserrnd.csd].

## Exemple 240. Exemple de l'opcode `duserrnd`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o duserrnd.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

k1    duserrnd 1
      printk 0, k1
asig  poscil .5, 220*k1, 2 ;multiply frequency with random value
      outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 -20 -41 2 .1 8 .9 ;choose 2 at 10% probability, and 8 at 90%

f2 0 8192 10 1

i1 0 2
e
</CsScore>
</CsoundSynthesizer>
```

La sortie contiendra des lignes comme celles-ci :

```
i 1 time 0.00067: 8.00000
i 1 time 0.00133: 8.00000
i 1 time 0.00200: 8.00000
i 1 time 0.00267: 8.00000
i 1 time 0.00333: 2.00000
i 1 time 0.00400: 8.00000
i 1 time 0.00533: 8.00000
i 1 time 0.00600: 8.00000
.....
```

## Voir aussi

*cuserrnd, urd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.16

# dust

dust — Impulsions aléatoires.

## Description

Génère des impulsions aléatoires entre 0 et 1.

## Syntaxe

```
ares dust kamp, kdensity
```

```
kres dust kamp, kdensity
```

## Exécution

*kamp* -- amplitude.

*kdensity* -- nombre moyen d'impulsions par seconde.

## Exemples

Voici un exemple de l'opcode dust. Il utilise le fichier *dust.csd* [examples/dust.csd].

### Exemple 241. Exemple de l'opcode dust.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o oscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kdens expon 2, p3, 20000
aout dust 0.5, kdens
outs aout, aout

endin
</CsInstruments>
<CsScore>
i1 0 10
e
</CsScore>
```

`</CsoundSynthesizer>`

## Voir aussi

*dust2 gausstrig mpulse*

## Crédits

Basé sur le générateur unitaire Dust de James McCartney (SuperCollider)

Auteur : Tito Latini

Janvier 2012

Nouveau dans la version 5.16 de Csound.

La version de taux-k a été corrigée pour être conforme à la documentation et à SuperCollider, dans la version 6.09



# dust2

dust2 — Impulsions aléatoires.

## Description

Génère des impulsions aléatoires entre 0 et 1.

## Syntaxe

```
ares dust2 kamp, kdensity
```

```
kres dust2 kamp, kdensity
```

## Exécution

*kamp* -- amplitude.

*kdensity* -- nombre moyen d'impulsions par seconde.

## Exemples

Voici un exemple de l'opcode dust2. Il utilise le fichier *dust2.csd* [examples/dust2.csd].

### Exemple 242. Exemple de l'opcode dust2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o oscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kdens expon 2, p3, 20000
aout dust2 0.5, kdens
outs aout, aout

endin
</CsInstruments>
<CsScore>
i1 0 10
e
</CsScore>
```

</CsoundSynthesizer>

## Voir aussi

*dust gausstrig mpulse*

## Crédits

Basé sur le générateur unitaire Dust2 de James McCartney (SuperCollider)

Auteur : Tito Latini

Janvier 2012

Nouveau dans la version 5.16 de Csound.

La version de taux-k a été corrigée pour être conforme à la documentation et à SuperCollider, dans la version 6.09

# else

else — Exécute un bloc de code lorsqu'une condition "if...then" est fausse.

## Description

Exécute un bloc de code lorsqu'une condition "if...then" est fausse.

## Syntaxe

`else`

## Exécution

*else* est utilisé dans un bloc de code entre les opcodes "*if...then*" et *endif*. Il définit les instructions à exécuter lorsqu'une condition "if...then" est fausse. Il ne peut y avoir qu'une seule instruction *else* et celle-ci doit être la dernière instruction conditionnelle avant l'opcode *endif*.

## Exemples

Voici un exemple de l'opcode *else*. Il utilise le fichier *else.csd* [examples/else.csd].

### Exemple 243. Exemple de l'opcode *else*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o else.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ipch = cpspch(p4)
ienv = p5

if (ienv == 0) then
  kenv adsr 0.01, 0.95, .7, .5
else
  kenv linseg 0, p3 * .5, 1, p3 * .5, 0
endif

aout vco2      .8, ipch, 10
aout moogvcf aout, ipch + (kenv * 6 * ipch) , .5
```

```
    aout = aout * kenv
    outs aout, aout

    endin
  </CsInstruments>
</CsScore>

i 1 0 2 8.00 0
i 1 3 2 8.00 1

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*elseif, endif, goto, if, igoto, kgoto, tigoto, timeout*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/2006spring/controlFlow.html>, écrit par Steven Yi.

## Crédits

Nouveau dans la version 4.21

# elseif

elseif — Définit une autre condition "if...then" lorsqu'une condition "if...then" est fausse.

## Description

Définit une autre condition "if...then" lorsqu'une condition "if...then" est fausse.

## Syntaxe

```
elseif xa R xb then
```

où *R* est un des opérateurs relationnels (<, =, <=, ==, !=) (et = par commodité, voir aussi *Valeurs Conditionnelles*).

## Exécution

*elseif* est utilisé dans un bloc de code entre les opcodes "*if...then*" et *endif*. Lorsqu'une condition "if...then" est fausse, cela définit une autre condition "if...then" à tester. On peut utiliser n'importe quel nombre d'instructions *elseif*.

## Exemples

Voici un exemple de l'opcode elseif. Il utilise le fichier *elseif.csd* [examples/elseif.csd].

### Exemple 244. Exemple de l'opcode elseif.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o elseif.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ipch = cpspch(p4)
ienv = p5

if (ienv == 0) then
;ADSR
kenv adsr 0.05, 0.05, .95, .05
elseif (ienv == 1) then
```

```

;Linear Triangular Envelope
kenv linseg 0, p3 * .5, 1, p3 * .5, 0
elseif (ienv == 2) then
;Ramp Up
kenv linseg 0, p3 - .01, 1, .01, 0
endif

aout vco2 .8, ipch, 10
aout moogvcf aout, ipch + (kenv * 5 * ipch) , .5

aout = aout * kenv

outs aout, aout
endin
</CsInstruments>
<CsScore>

i 1 0 2 8.00 0
i 1 3 2 8.00 1
i 1 6 2 8.00 2
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*else, endif, goto, if, igoto, kgoto, tigoto, timeout*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/2006spring/controlFlow.html>, écrit par Steven Yi.

## Crédits

Nouveau dans la version 4.21

# endif

endif — Termine un bloc de code qui commence par une instruction "if...then".

## Description

Termine un bloc de code qui commence par une instruction "if...then".

## Syntaxe

endif

## Exécution

Tout bloc de code commençant par une instruction "if...then" doit se terminer par une instruction *endif*.

## Exemples

Voici un exemple de l'opcode endif. Il utilise le fichier *endif.csd* [examples/endif.csd].

### Exemple 245. Exemple de l'opcode endif.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o endif.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
; Get the note value from the fourth p-field.
knote = p4

; Does the user want a low note?
if (knote == 0) then
  kcps = 220
; Does the user want a middle note?
elseif (knote == 1) then
  kcps = 440
; Does the user want a high note?
elseif (knote == 2) then
  kcps = 880
endif

; Create the note.
```

```

    kamp init .8
    ifn = 1
    a1 oscili kamp, kcps, ifn

    outs a1, a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; p4: 0=low note, 1=middle note, 2=high note.
; Play Instrument #1 for one second, low note.
i 1 0 1 0
; Play Instrument #1 for one second, middle note.
i 1 1 1 1
; Play Instrument #1 for one second, high note.
i 1 2 1 2
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*elseif, else, goto, if, igoto, kgoto, tgoto, timeout*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/2006spring/controlFlow.html>, écrit par Steven Yi.

## Crédits

Nouveau dans la version 4.21



# endin

endin — Termine un bloc d'instrument.

## Description

Termine le bloc d'instrument courant.

## Syntaxe

endin

## Initialisation

Termine le bloc d'instrument courant.

On peut définir les instruments dans n'importe quel ordre (mais ils seront toujours initialisés et exécutés par ordre de numéro d'instrument ascendant). Les blocs d'instruments ne peuvent pas être imbriqués (un bloc ne peut pas en contenir un autre).



### Note

Il peut y avoir n'importe quel nombre de blocs d'instrument dans un orchestre.

## Exemples

Voici un exemple de l'opcode *endin*. Il utilise le fichier *endin.csd* [examples/endin.csd].

### Exemple 246. Exemple de l'opcode *endin*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o endin.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  iamp = 10000
  icps = 440
  iphs = 0
```

```
    a1 oscils iamp, icps, iphs
    out a1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*instr*

## Crédits

Exemple écrit par Kevin Conder.

# endop

endop — Termine un bloc d'opcode défini par l'utilisateur.

## Description

Termine un bloc d'opcode défini par l'utilisateur.

## Syntaxe

endop

## Exécution

La syntaxe d'un bloc d'opcode défini par l'utilisateur est la suivante :

```
opcode nom, outtypes, intypes
xinarg1 [, xinarg2] [, xinarg3] ... [xinargN] xin
[setksmps iksmps]
... the rest of the instrument's code.
xout xoutarg1 [, xoutarg2] [, xoutarg3] ... [xoutargN]
endop
```

Le nouvel opcode peut ensuite être utilisé avec la syntaxe usuelle :

```
[xinarg1] [, xinarg2] ... [xinargN] nom [xoutarg1] [, xoutarg2] ... [xoutargN] [, iksmps]
```

## Exemples

Voici un exemple de l'opcode endop. Il utilise le fichier *endop.csd* [examples/endop.csd].

### Exemple 247. Exemple de l'opcode endop.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o endop.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

/* example opcode 1: simple oscillator */

opcode Oscillator, a, kk
```

```

kamp, kcps      xin           ; read input parameters
a1      vco2 kamp, kcps      ; sawtooth oscillator
      xout a1                ; write output

endop

/* example opcode 2: lowpass filter with local ksmps */

opcode Lowpass, a, akk

      setksmps 1              ; need sr=kr
ain, ka1, ka2    xin           ; read input parameters
aout      init 0              ; initialize output
aout      = ain*ka1 + aout*ka2 ; simple tone-like filter
      xout aout                ; write output

endop

/* example opcode 3: recursive call */

opcode RecursiveLowpass, a, akkpp

ain, ka1, ka2, idep, icnt      xin           ; read input parameters
      if (icnt >= idep) goto skip1 ; check if max depth reached
ain      RecursiveLowpass ain, ka1, ka2, idep, icnt + 1
skip1:
aout      Lowpass ain, ka1, ka2      ; call filter
      xout aout                ; write output

endop

/* example opcode 4: de-click envelope */

opcode DeClick, a, a

ain      xin
aenv      linseg 0, 0.02, 1, p3 - 0.05, 1, 0.02, 0, 0.01, 0
      xout ain * aenv           ; apply envelope and write output

endop

/* instr 1 uses the example opcodes */

instr 1

kamp      = .6                  ; amplitude
kcps      expon 50, p3, 500      ; pitch
a1      Oscillator kamp, kcps      ; call oscillator
kflt      linseg 0.4, 1.5, 0.4, 1, 0.8, 1.5, 0.8 ; filter envelope
a1      RecursiveLowpass a1, kflt, 1 - kflt, 10 ; 10th order lowpass
a1      DeClick a1
      outs a1, a1

endin
</CsInstruments>
<CsScore>

i 1 0 4
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*opcode, setksmps, xin, xout*

Plus d'information sur cet opcode : [http://www.csoundjournal.com/2006summer/controlFlow\\_part2.html](http://www.csoundjournal.com/2006summer/controlFlow_part2.html), écrit par Steven Yi.

La page des opcodes définis par l'utilisateur : <http://www.csounds.com/udo/>, maintenue par Steven Yi.

## Crédits

Auteur : Istvan Varga, 2002 ; basé sur du code de Matt J. Ingalls

Nouveau dans la version 4.22

# envlpx

envlpx — Applique une enveloppe constituée de 3 segments.

## Description

*envlpx* -- applique une enveloppe constituée de 3 segments :

1. une attaque dont la forme est donnée par une fonction
2. un pseudo entretien modifié exponentiellement
3. une chute exponentielle

## Syntaxe

ares **envlpx** xamp, irise, idur, idec, ifn, iatss, iatdec [, ixmod]

kres **envlpx** kamp, irise, idur, idec, ifn, iatss, iatdec [, ixmod]

## Initialisation

*irise* -- durée de l'attaque en secondes. Une valeur nulle ou négative signifie pas d'attaque.

*idur* -- durée globale en seconde. Avec une valeur nulle ou négative, l'initialisation sera ignorée.

*idec* -- durée de la chute en secondes. Zéro signifie pas de chute. Si *idec* > *idur* la chute sera tronquée.

*ifn* -- numéro de la table de fonction avec point de garde dans laquelle la forme de l'attaque est stockée.

*iatss* -- facteur d'atténuation par lequel la dernière valeur de l'attaque d'*envlpx* évolue pendant le pseudo entretien de la note. Un facteur supérieur à 1 provoque une montée exponentielle tandis qu'un facteur inférieur à 1 crée une descente exponentielle. Un facteur égal à 1 maintient un véritable entretien de la note sur la dernière valeur de l'attaque. Il faut noter que cette atténuation n'évolue pas à vitesse constante (comme dans le cas du piano), mais qu'elle dépend de la durée de la note. Cependant, si *iatss* est négatif (ou si l'entretien < 4 périodes-k) une vitesse d'atténuation de *abs(iatss)* par seconde sera utilisée. 0 est interdit.

*iatdec* -- facteur d'atténuation par lequel la dernière valeur de l'entretien diminue exponentiellement pendant la chute. Cette valeur doit être positive et elle est normalement de l'ordre de 0,01. Une valeur trop longue ou excessivement courte peut produire une coupure audible. Les valeurs nulles ou négatives sont interdites.

*ixmod* (facultatif, entre +- 0,9 environ) -- facteur de modification de courbe exponentielle, qui influe sur la raideur de la trajectoire exponentielle pendant l'entretien. Les valeurs négatives provoqueront une montée ou une descente accélérée (par exemple *subito piano*). Les valeurs positives provoqueront une montée ou une descente ralentie. La valeur par défaut est zéro (exponentielle non modifiée).

## Exécution

*kamp*, *xamp* -- amplitude du signal d'entrée.

Les modifications de l'attaque sont appliquées pendant les premières *irise* secondes, et celles de la chute à partir de *idur* - *idec*. Si ces périodes sont séparées dans le temps il y aura un entretien au cours duquel *amp*

sera modifié selon le schéma exponentiel décrit. Si la durée globale *idur* est dépassée pendant l'exécution, la chute continuera dans la même direction, en tendant asymptotiquement vers zéro.

## Exemples

Voici un exemple de l'opcode *envlpx*. Il utilise le fichier *envlpx.csd* [examples/envlpx.csd].

### Exemple 248. Exemple de l'opcode *envlpx*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o envlpx.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

irise = 0.2
idec = 0.5
idur = p3 - idec

ifn = 1
iatss = p5
iatdec = 0.01

kenv envlpx .6, irise, idur, idec, ifn, iatss, iatdec
kcps = cpspch(p4)
asig vco2 kenv, kcps
;apply envlpx to the filter cut-off frequency
asig moogvcf asig, kcps + (kenv * 8 * kcps) , .5 ;the higher the pitch, the higher the filter cut-off f
outs asig, asig

endin
</CsInstruments>
<CsScore>
; a linear rising envelope
f 1 0 129 -7 0 128 1

i 1 0 2 7.00 .1
i 1 + 2 7.02 1
i 1 + 2 7.03 2
i 1 + 2 7.05 3
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*envlpxr*, *linen*, *linenr*

## Crédits

Merci à Luis Jure pour avoir signalé une erreur avec *iatss*.



# envlpxr

envlpxr — L'opcode *envlpx* avec un segment final de relâchement.

## Description

*envlpxr* est le même que *envlpx* sauf que le segment final n'est exécuté qu'après un évènement MIDI de relâchement de note. La note est ensuite allongée de la durée de la chute.

## Syntaxe

```
ares envlpxr xamp, irise, idec, ifn, iatss, iatdec [, ixmod] [, irind]
```

```
kres envlpxr kamp, irise, idec, ifn, iatss, iatdec [, ixmod] [, irind]
```

## Initialisation

*irise* -- durée de l'attaque en secondes. Une valeur nulle ou négative signifie pas d'attaque.

*idec* -- durée de la chute en secondes. Zéro signifie pas de chute.

*ifn* -- numéro de la table de fonction avec point de garde dans laquelle la forme de l'attaque est stockée.

*iatss* -- facteur d'atténuation par lequel la dernière valeur de l'attaque d'*envlpxr* évolue pendant le pseudo entretien de la note. Un facteur supérieur à 1 provoque une montée exponentielle tandis qu'un facteur inférieur à 1 crée une descente exponentielle. Un facteur égal à 1 maintient un véritable entretien de la note sur la dernière valeur de l'attaque. Il faut noter que cette atténuation n'évolue pas à vitesse constante (comme dans le cas du piano), mais qu'elle dépend de la durée de la note. Cependant, si *iatss* est négatif (ou si l'entretien < 4 périodes-k) une vitesse d'atténuation de *abs(iatss)* par seconde sera utilisée. 0 est interdit.

*iatdec* -- facteur d'atténuation par lequel la dernière valeur de l'entretien diminue exponentiellement pendant la chute. Cette valeur doit être positive et elle est normalement de l'ordre de 0,01. Une valeur trop longue ou excessivement courte peut produire une coupure audible. Les valeurs nulles ou négatives sont interdites.

*ixmod* (facultatif, entre +- 0,9 environ) -- facteur de modification de courbe exponentielle, qui influe sur la raideur de la trajectoire exponentielle pendant l'entretien. Les valeurs négatives provoqueront une montée ou une descente accélérée (par exemple *subito piano*). Les valeurs positives provoqueront une montée ou une descente ralentie. La valeur par défaut est zéro (exponentielle non modifiée).

*irind* (facultatif) -- indicateur d'indépendance. S'il est nul, la durée de relâchement (*idec*) aura une influence sur l'allongement de la note après un note-off. S'il est non nul, la durée *idec* sera relativement indépendante de l'allongement de la note (voir ci-dessous). La valeur par défaut est 0.

## Exécution

*kamp*, *xamp* -- amplitude du signal d'entrée.

*envlpxr* fait partie des unités « r » de Csound qui contiennent un détecteur de fin de note et une extension de durée pour le relâchement. Quand la fin d'un évènement ou MIDI note-off est détectée, la durée d'exécution de l'instrument courant est immédiatement allongée de *idec* secondes à moins qu'il ne soit rendu

indépendant par *irind*. Dans ce cas, la chute démarrera de l'endroit, quelqu'il soit, où l'on se trouvait à ce moment précis.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note-off, comme *linsegr* et *expsegr*, ou bien l'on peut construire des enveloppes plus complexes au moyen de *xtratim* et de *release*. Noter qu'il n'est pas nécessaire d'utiliser *xtratim* avec *envlpxr*, car la durée est allongée automatiquement.

Ces unités « r » peuvent être modifiées également par des événements MIDI note-off provoqués par une vitesse nulle. Si l'indicateur *irind* est positionné (différent de zéro), la durée d'exécution totale n'est pas affectée par les données de note-off ou de vitesse nulle.

**Unités « r » multiples.** Quand plusieurs unités « r » sont présentes dans le même instrument, il est habituel qu'une seule d'entre elle influence la durée totale de la note. C'est normalement l'unité contrôlant l'amplitude principale de la note. D'autres unités contrôlant par exemple l'évolution d'un filtre, peuvent toujours être sensibles aux commandes note-off tout en n'affectant pas la durée grâce à leur indépendance (*irind* non nul). En fonction de leur propre valeur *idec* (durée de relâchement), les unités « r » indépendantes pourront ou ne pourront pas atteindre leur destination finale avant que la note ne se termine. Si elles y arrivent, elles tiendront simplement leur dernière valeur jusqu'à la fin. Si plusieurs unités « r » sont principales, l'extension de la note sera celle de la plus grande valeur *idec*.

## Exemples

Voici un exemple de l'opcode *envlpxr*. Il utilise le fichier *envlpxr.csd* [examples/envlpxr.csd].

### Exemple 249. Exemple de l'opcode *envlpxr*.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0   ;;realtime audio out and realtime midi in
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o envlpxr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

icps cpsmidi
iamp ampmidi .5

kenv envlpxr iamp, 0.2, 1, 1, 1, 0.01
asig pluck kenv, icps, 200, 2, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 129 -7 0 128 1
f 2 0 4096 10 1

f0 30 ;runs 30 seconds
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*envlpx, linen, linenr*

## Crédits

Merci à Luis Jure pour avoir signalé une erreur avec *iatss*.

# ephasor

ephasor — Produit deux sorties : un signal de phase périodique et un signal de décroissance exponentielle périodique.

## Description

Cet opcode produit deux sorties : un signal de phase périodique (comme l'opcode *phasor*), et un signal de décroissance exponentielle périodique. Le second est synchronisé au premier, commençant à 1 et décroissant pendant que le signal de phase croît de 0 à 1. Le taux de décroissance exponentielle peut être contrôlé par le second paramètre.

## Syntaxe

```
aexp,aphephasor kfreq, kR
```

## Exécution

*kfreq* - le taux de génération des signaux de phase et exponentiel.

*kR* - un paramètre contrôlant le taux de décroissance du signal exponentiel,  $0 < kR < 1$ . Les petites valeurs produisent une décroissance plus rapide.

## Exemples

Voici un exemple de l'opcode ephasor. Il utilise le fichier *ephasor.csd* [examples/ephasor.csd].

### Exemple 250. Exemple de l'opcode ephasor.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr=44100
ksmps=64
nchnls=2
0dbfs = 1

instr 1

iamp = p4
ifr = p5
kfr expon ifr,p3,2*ifr

kfc expon 2000,p3, 4000
kq = 10
kbw = kfc/kq
kR = 1 - $M_PI*(kbw/sr)

k1 = kfc/kfr
kn = int(k1)
```

```

k1 = k1 - kn

amod,aph ephasor kfr,kR
aosc1 table aph*kn,-1,1,0,1
aosc2 table aph*(kn+1),-1,1,0,1

asig = iamp*(aosc1*(1 - k1) + aosc2*k1)*amod
outs asig, asig

endin

</CsInstruments>
<CsScore>

i1 0 10 0.5 220

e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Victor Lazzarini  
2008

Nouveau dans la version 5.10

# eqfil

eqfil — Filtre égaliseur.

## Description

L'opcode *eqfil* est un filtre d'égalisation du second ordre accordable basé sur le modèle de Regalia et Mitra ("Tunable Digital Frequency Response Equalization Filters", IEEE Trans. on Ac., Sp. and Sig Proc., 35 (1), 1987). Il fournit un filtre à pics/creux pour construire des égaliseurs paramétriques/graphiques.

L'amplitude de la réponse du filtre sera plate (=1) pour *kgain*=1. Si *kgain* est supérieur à 1, il y aura un pic à la fréquence centrale dont la largeur est donnée par le paramètre *kbw*, et en-dehors de cette bande, la réponse tendra vers 1. Inversement, si *kgain* est inférieur à 1, il y aura un creux autour de la fréquence centrale.

## Syntaxe

```
asig eqfil ain, kcf, kbw, kgain[, istor]
```

## Initialisation

*istor* -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal de sortie filtré.

*ain* -- signal d'entrée.

*kcf* -- fréquence centrale du filtre.

*kbw* -- largeur de bande du pic/creux (Hz).

*kgain* -- gain du pic/creux.

## Exemples

Voici un exemple de l'opcode *eqfil*. Il utilise le fichier *eqfil.csd* [examples/eqfil.csd].

### Exemple 251. Exemple de l'opcode *eqfil*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;;realtime audio out
```

```
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o eqfil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kcf = p4
kfe expseg 10, p3*0.9, 1800, p3*0.1, 175
kenv linen .03, 0.05, p3, 0.05 ;low amplitude is needed to avoid clipping
asig buzz kenv, kfe, sr/(2*kfe), 1
afil eqfil asig, kcf, 200, 10
outs afil*20, afil*20

endin
</CsInstruments>
<CsScore>
; a sine wave.
f 1 0 16384 10 1

i 1 0 10 200 ;filter centre freq=200
i 1 + 10 1500 ;filter centre freq=1500
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
Avril 2007

Nouveau dans la version 5.06

# evalstr

evalstr — evalstrs évalue une chaîne de caractères contenant du code Csound et retourne une valeur.

## Description

*evalstr* compile et exécute du code Csound et retourne une valeur de l'espace global (instr 0). On peut utiliser cet opcode pour compiler de nouveaux instruments (comme *compilestr*).

## Syntaxe

```
ires evalstr Scode  
kres evalstr Scode, ktrig
```

## Initialisation

« *Scode* » -- une chaîne de caractères à compiler et à évaluer.

## Exécution

« *ktrig* » -- déclenche la compilation/évaluation s'il est différent de zéro.

## Exemples

Voici un exemple de l'opcode evalstr en conjonction avec return :

### Exemple 252.

```
ival evalstr "return 2 + 2"  
print ival
```

## Crédits

Auteur : Victor Lazzarini, 2013



# event

event — Génère un évènement de partition à partir d'un instrument.

## Description

Génère un évènement de partition à partir d'un instrument.

## Syntaxe

```
event "scorechar", kinsnum, kdelay, kdur, [, kp4] [, kp5] [, ...]
```

```
event "scorechar", "insname", kdelay, kdur, [, kp4] [, kp5] [, ...]
```

## Initialisation

« *scorechar* » -- Une chaîne de caractères (entre guillemets) représentant le p-champ initial dans une instruction de partition. C'est habituellement « *e* », « *f* », ou « *i* ».

« *insname* » -- Une chaîne de caractères (entre guillemets) représentant un instrument nommé.

## Exécution

*kinsnum* -- L'instrument à utiliser pour l'évènement. Cela correspond au premier p-champ, p1, dans une instruction de partition.

*kdelay* -- Quand (en secondes) l'évènement aura lieu à partir de l'instant courant de l'exécution. Cela correspond au second p-champ, p2, dans une instruction de partition.

*kdur* -- Durée (en secondes) de l'évènement. Cela correspond au troisième p-champ, p3, dans une instruction de partition.

*kp4*, *kp5*, ... (facultatif) -- Paramètres représentant des p-champs supplémentaires dans une instruction de partition. Ils commencent par le quatrième p-champ, p4.



### Note

Noter que l'opcode *event* ne peut pas accepter de p-champs chaîne de caractère. Si vous devez passer des chaînes de caractère à l'instanciation d'un instrument, utilisez l'opcode *scoreline* ou *scoreline\_i*.

## Exemples

Voici un exemple de l'opcode event. Il utilise le fichier *event.csd* [examples/event.csd].

### Exemple 253. Exemple de l'opcode event.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```

; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o event.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - an oscillator with a high note.
instr 1
; Create a trigger and set its initial value to 1.
ktrigger init 1

; If the trigger is equal to 0, continue playing.
; If not, schedule another event.
if (ktrigger == 0) goto contin
; kscoreop="i", an i-statement.
; kinsnum=2, play Instrument #2.
; kwhen=1, start at 1 second.
; kdur=0.5, play for a half-second.
event "i", 2, 1, 0.5

; Make sure the event isn't triggered again.
ktrigger = 0

contin:
a1 oscils 10000, 440, 1
out a1
endin

; Instrument #2 - an oscillator with a low note.
instr 2
a1 oscils 10000, 220, 1
out a1
endin

</CsInstruments>
<CsScore>

; Make sure the score plays for two seconds.
f 0 2

; Play Instrument #1 for a half-second.
i 1 0 0.5
e

</CsScore>
</CsoundSynthesizer>

```

Voici un exemple de l'opcode event utilisant un instrument nommé. Il utilise le fichier *event\_named.csd* [examples/event\_named.csd].

### Exemple 254. Exemple de l'opcode event utilisant un instrument nommé.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages

```

```

-odac          -iadc      -d      ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o event_named.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - an oscillator with a high note.
instr 1
; Create a trigger and set its initial value to 1.
ktrigger init 1

; If the trigger is equal to 0, continue playing.
; If not, schedule another event.
if (ktrigger == 0) goto contin
; kscoreop="i", an i-statement.
; kinsnum="low_note", instrument named "low_note".
; kwhen=1, start at 1 second.
; kdur=0.5, play for a half-second.
event "i", "low_note", 1, 0.5

; Make sure the event isn't triggered again.
ktrigger = 0

contin:
a1 oscils 10000, 440, 1
out a1
endin

; Instrument "low_note" - an oscillator with a low note.
instr low_note
a1 oscils 10000, 220, 1
out a1
endin

</CsInstruments>
<CsScore>

; Make sure the score plays for two seconds.
f 0 2

; Play Instrument #1 for a half-second.
i 1 0 0.5
e

</CsScore>
</CsSoundSynthesizer>

```

## Voir aussi

*event\_i, schedule, schedwhen, schedkwhen, schedkwhennamed, scoreline, scoreline\_i*

## Crédits

Exemples écrits par Kevin Conder.

Nouveau dans la version 4.17

Merci à Matt Ingalls pour son aide à corriger l'exemple.

Merci à Matt Ingalls pour son aide à clarifier le paramètre `kwhen/kdelay`.

# event\_i

event\_i — Génère un évènement de partition à partir d'un instrument.

## Description

Génère un évènement de partition à partir d'un instrument.

## Syntaxe

```
event_i "scorechar", iinsnum, idelay, idur, [, ip4] [, ip5] [, ...]
```

```
event_i "scorechar", "insname", idelay, idur, [, ip4] [, ip5] [, ...]
```

## Initialisation

« *scorechar* » -- Une chaîne de caractères (entre guillemets) représentant le p-champ initial d'une instruction de partition. C'est habituellement « *e* », « *f* », ou « *i* ».

« *insname* » -- Une chaîne de caractères (entre guillemets) représentant un instrument nommé.

*iinsnum* -- L'instrument à utiliser pour cet évènement. Cela correspond au premier p-champ, p1, dans une instruction de partition.

*idelay* -- Quand (en secondes) l'évènement aura lieu à partir de l'instant courant de l'exécution. Cela correspond au second p-champ, p2, dans une instruction de partition.

*idur* -- Durée (en secondes) de l'évènement. Cela correspond au troisième p-champ, p3, dans une instruction de partition.

*ip4*, *ip5*, ... (facultatif) -- Paramètres représentant des p-champs supplémentaires dans une instruction de partition. Ils commencent par le quatrième p-champ, p4.

## Exécution

L'évènement est ajouté à la file d'attente pendant la phase d'initialisation.



### Note

Noter que l'opcode *event\_i* ne peut pas accepter de p-champs chaîne de caractère. Si vous devez passer des chaînes de caractère à l'instanciation d'un instrument, utilisez l'opcode *scoreline* ou *scoreline\_i*.

## Exemples

Voici un exemple de l'opcode *event\_i*. Il utilise le fichier *event\_i.csd* [examples/event\_i.csd].

### Exemple 255. Exemple de l'opcode event\_i.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime output leave only the line below:
; -o event_i.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

seed 0
gisine ftgen 0, 0, 2^10, 10, 1

instr 1 ;master instrument

ininstr = 10 ;number of called instances
indx = 0
loop:
ipan random 0, 1
ifreq random 100, 1000
iamp = 1/ininstr
event_i "i", 10, 0, p3, iamp, ifreq, ipan
loop_lt indx, 1, ininstr, loop

endin

instr 10

    print p4, p5, p6
ipeak random 0, 1 ;where is the envelope peak
asig poscil3 p4, p5, gisine
aenv transeg 0, p3*ipeak, 6, 1, p3-p3*ipeak, -6, 0
aL,aR pan2 asig*aenv, p6
outs aL, aR

endin

</CsInstruments>
<CsScore>
i1 0 10
i1 8 10
i1 16 15
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*event, schedule, schedwhen, schedkwhen, schedkwhennamed, scoreline, scoreline\_i*

## Crédits

Ecrit par Istvan Varga.

Nouveau dans Csound5

# exciter

exciter — Un système de filtre non-linéaire pour exciter le signal.

## Description

Opcode du greffon exciter.

Distorsion filtrée pour ajouter de la brillance à un signal.

## Syntaxe

```
ares exciter asig, kfreq, kceil, kharmonics, kblend
```

## Exécution

*asig* -- signal d'entrée

*kfreq* -- la limite inférieure des harmoniques créés.

*kceil* -- la limite supérieure des harmoniques créés.

*kharmonics* -- quantité d'harmoniques, entre 0.1 et 10.

*kblend* -- mélange entre harmoniques du second et du troisième ordre, compris entre -10 et +10.

*exciter* est une réécriture du greffon *calf exciter*.

## Exemples

Voici un exemple de l'opcode exciter. Il utilise le fichier *exciter.csd* [examples/exciter.csd].

### Exemple 256. Exemple de l'opcode exciter.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;real-time audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o exciter.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=32
nchnls=2
0dbfs =1

instr 1 ; excited sound + original sound
```

```

a1 diskin2 "fox.wav", 1
a2 exciter a1, 3000, 20000, 10, 10 ;generate uneven harmonics at maximum setting
    outs a2+a1, a2+a1
endin

instr 2 ; original sound for comparison

a1 diskin2 "fox.wav", 1
    outs a1, a1
endin

instr 3 ; the effect of the excited sound only

a1 diskin2 "fox.wav", 1
a2 exciter a1, 3000, 20000, 10, 10 ;generate uneven harmonics at maximum setting
    outs a2, a2
endin

</CsInstruments>
<CsScore>
i1 0 3
i2 3 3
i3 6 3
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch d'après Markus Schmidt  
Août 2014

Nouveau dans la version 6.04 de Csound.



# exitnow

exitnow — Quitte Csound aussi vite que possible, sans nettoyage.

## Description

Dans Csound4 cela appelle une fonction de sortie pour quitter Csound aussi vite que possible. Dans Csound5, on revient au code appelant.

## Syntaxe

**exitnow** [*ivalue*]

## Initialisation

Arrête Csound lors du cycle d'initialisation et retourne *ivalue* qui vaut zéro par défaut. A noter qu'il est habituel de trouver cet opcode seul dans un instrument.

## Exemples

Voici un exemple de l'opcode exitnow. Il utilise le fichier *exitnow.csd* [examples/exitnow.csd].

### Exemple 257. Exemple de l'opcode exitnow.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n          ;;no sound
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o exitnow.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

;after an example by Iain McCurdy

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

FLcolor 200, 200, 200, 0, 0, 0
; LABEL      | WIDTH | HEIGHT | X | Y
FLpanel "rtclock", 500, 130, 0, 0
;
; ON,OFF,TYPE,WIDTH, HEIGHT, X, Y, OPCODE, INS,START,IDUR
gkOnOff,ihOnOff FLbutton "On/Off", 1, 0, 22, 150, 25, 5, 5, 0, 1, 0, 3600
gkExit,ihExit FLbutton "exitnow",1, 0, 21, 150, 25, 345, 5, 0, 999, 0, 0.001
FLsetColor2 255, 0, 50, ihOnOff ;reddish color

;VALUE DISPLAY BOXES WIDTH,HEIGHT,X, Y
gidclock FLvalue "clock", 100, 25, 200, 60
FLsetVal_i 1, ihOnOff
FLpanel_end
```

**FLrun**

**instr** 1

**if** gkOnOff !=0 **kgoto** CONTINUE ;sense if FLTK on/off switch is not off (in which case skip the next line)  
**turnoff** ;turn this instr. off now

**CONTINUE:**

**ktime** **rtclock** ;clock continues to run even

**FLprintk2** ktime, gldclock ;after the on/off button was used to stop

**endin**

**instr** 999

**exitnow** 2 ;exit Csound as fast as possible

**endin**

</CsInstruments>

<CsScore>

**f** 0 60 ;runs 60 seconds

**e**

</CsScore>

</CsoundSynthesizer>

# exp

exp — Retourne  $e$  élevé à la puissance  $x$ .

## Description

Retourne  $e$  élevé à la puissance  $x$ .

## Syntaxe

**exp**( $x$ ) (pas de restriction de taux)

**exp**( $k/i[]$ ) ( $k$ - ou  $i$ -tableau)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode exp. Il utilise le fichier *exp.csd* [examples/exp.csd].

### Exemple 258. Exemple de l'opcode exp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o exp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisine      ftgen      0, 0, 2^10, 10, 1 ;table for a sine wave

instr 1 ;master instrument
koct      linseg      6, p3, 12 ; octave register straight rising from 6 to 12
kexp      linseg      0, p3/3, 5, p3/3, 5, p3/3, 0 ;exponent goes from 0 to 5 and back
kdens      =          exp(kexp) ;density is e to the power of kexp
printks    "Generated events per second: %d\n", 1, kdens
ktrig      metro      kdens ;trigger single notes in kdens frequency
if ktrig == 1 then
;call instr 10 for 1/kdens duration, .5 amplitude and koct register
event      "i", 10, 0, 1/kdens, .5, koct
endif
endin

instr 10 ;performs one tone
```

```
ioct      rnd31      1, 0 ;random deviation maximum one octave plus/minus
aenv      transeg    p4, p3, -6, 0 ;fast decaying envelope for p4 amplitude
asin      poscil      aenv, cpsoct(p5+ioct), gisine ;sine for p5 octave register plus random deviation
outs      asin, asin

endin

</CsInstruments>
<CsScore>
i 1 0 30
e
</CsScore>
</CsoundSynthesizer>
```

La sortie contiendra des lignes comme celles-ci :

```
Generated events per second: 1
rtevent:  T 0.000 TT 0.000 M: 0.00000 0.00000
new alloc for instr 10:
rtevent:  T 0.811 TT 0.811 M: 0.48906 0.48906
new alloc for instr 10:
Generated events per second: 2
rtevent:  T 1.387 TT 1.387 M: 0.48611 0.48611
rtevent:  T 1.833 TT 1.833 M: 0.48421 0.48421
Generated events per second: 3
rtevent:  T 2.198 TT 2.198 M: 0.47536 0.47536
rtevent:  T 2.506 TT 2.506 M: 0.46530 0.46530
rtevent:  T 2.773 TT 2.773 M: 0.44986 0.44986
Generated events per second: 4
rtevent:  T 3.009 TT 3.009 M: 0.48096 0.48096
.....
```

## Voir aussi

*abs, frac, int, log, log10, i, sqrt*

Nouveau dans la version 4.21

# expcurve

expcurve — Cet opcode implémente une formule qui génère une courbe exponentielle normalisée dans l'intervalle 0 - 1. Il est basé sur le travail dans Max / MSP de Eric Singer (c) 1994.

## Description

Génère une courbe exponentielle dans l'intervalle de 0 à 1 avec une raideur de pente arbitraire. Une raideur de pente inférieure ou égale à 1,0 lévera des erreurs NaN (Not-a-Number) et provoquera un comportement instable.

La formule utilisée pour le calcul de la courbe est :

$$(\exp(x * \log(y))-1) / (y-1)$$

où x est égal à *kindex* et y est égal à *ksteepness*.

## Syntaxe

kout **expcurve** kindex, ksteepness

## Exécution

*kindex* -- Valeur d'indice. Attendue dans l'intervalle de 0 à 1.

*ksteepness* -- Raideur de la courbe générée. Avec des valeurs proches de 1,0 on obtient une courbe plus rectiligne alors qu'avec des valeurs plus grandes la courbe est plus raide.

*kout* -- Sortie pondérée.

## Exemples

Voici un exemple de l'opcode expcurve. Il utilise le fichier *expcurve.csd* [examples/expcurve.csd].

### Exemple 259. Exemple de l'opcode expcurve.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  Silent
-odac          -iadc     -d      ;;realtime output
</CsOptions>
<CsInstruments>

sr = 48000
ksmps = 1000
nchnls = 2

instr 1 ; logcurve test

kmod phasor 1/p3
```

```
kout expcurve kmod, p4

printks "mod = %f out= %f\\n", 0.5, kmod, kout

    endin

/*--- */
</CsInstruments>
<CsScore>

i1 0 5 2
i1 5 5 5
i1 10 5 30
i1 15 5 0.5

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*scale, gainslider, logcurve*

## Crédits

Auteur : David Akbari  
Octobre  
2006

# expon

expon — Trace une courbe exponentielle entre les points spécifiés.

## Description

Trace une courbe exponentielle entre les points spécifiés.

## Syntaxe

```
ares expon ia, idur, ib
```

```
kres expon ia, idur, ib
```

## Initialisation

*ia* -- valeur initiale. Zéro est interdit pour les exponentielles.

*ib* -- valeur après *idur* secondes. Pour les exponentielles, doit être non nulle et du même signe que *ia*.

*idur* -- durée en secondes du segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

## Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par deux points spécifiés. La valeur de *idur* peut égaliser ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, le segment continuera dans la même direction.

## Exemples

Voici un exemple de l'opcode *expon*. Il utilise le fichier *expon.csd* [examples/expon.csd].

### Exemple 260. Exemple de l'opcode *expon*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o expon.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
```

```

kpitch = p6
;choose between expon or line
if (kpitch == 0) then
    kpitch expon p4, p3, p5
elseif (kpitch == 1) then
    kpitch line p4, p3, p5
endif

asig    vco2 .6, kpitch
        outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 2 300 600    0 ;if p6=0 then expon is used
i 1 3 2 300 600    1 ;if p6=1 then line is used
i 1 6 2 600 1200   0
i 1 9 2 600 1200   1
i 1 12 2 1200 2400  0
i 1 15 2 1200 2400  1
i 1 18 2 2400 30    0
i 1 21 2 2400 30    1
e
</CsScore>
</CsSoundSynthesizer>

```

## Voir aussi

*expseg, expsegr, line, linseg, linsegr*



# exprand

exprand — Générateur de nombres aléatoires de distribution exponentielle (valeurs positives seulement).

## Description

Générateur de nombres aléatoires de distribution exponentielle (valeurs positives seulement). C'est un générateur de bruit de classe x.

## Syntaxe

```
ares exprand klambda
```

```
ires exprand klambda
```

```
kres exprand klambda
```

## Exécution

*klambda* -- paramètre lambda pour la distribution exponentielle.

La fonction de densité de probabilité d'une distribution exponentielle est une courbe exponentielle, dont la moyenne est  $0,69515/\lambda$ . Pour des explications plus détaillées de ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode exprand. Il utilise le fichier *exprand.csd* [examples/exprand.csd].

### Exemple 261. Exemple de l'opcode exprand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o exprand.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
odbfs = 1

instr 1    ; every run time same values
```

```

klamda exprand 20
  printk .2, klamda ; look
aout oscili 0.8, 440+klamda, 1 ; & listen
  outs aout, aout
endin

instr 2 ; every run time different values

  seed 0
klamda exprand 20
  printk .2, klamda ; look
aout oscili 0.8, 440+klamda, 1 ; & listen
  outs aout, aout
endin
</CsInstruments>
<CsScore>
; sine wave
f 1 0 16384 10 1

i 1 0 2
i 2 3 2
e

</CsScore>
</CsoundSynthesizer>

```

La sortie contiendra des lignes comme celles-ci :

```

i 1 time 0.00033: 4.09813
i 1 time 0.20033: 56.39567
i 1 time 0.40033: 3.23362
i 1 time 0.60033: 0.24277
i 1 time 0.80033: 13.71228
i 1 time 1.00000: 12.71885
i 1 time 1.20033: 32.36737
i 1 time 1.40033: 0.29747
i 1 time 1.60033: 4.04450
i 1 time 1.80000: 35.75676
i 1 time 2.00000: 3.69845

Seeding from current time 3034472128

i 2 time 3.00033: 6.67934
i 2 time 3.20033: 2.72431
i 2 time 3.40033: 14.51822
i 2 time 3.60000: 12.10120
i 2 time 3.80033: 1.12266
i 2 time 4.00000: 26.90772
i 2 time 4.20000: 0.43554
i 2 time 4.40033: 28.59836
i 2 time 4.60033: 27.01831
i 2 time 4.80033: 18.19911
i 2 time 5.00000: 4.45125

```

## Voir aussi

*seed, betarand, bexprnd, cauchy, gauss, linrand, pcauchy, poisson, trirand, unirand, weibull*

## Crédits

Auteur: Paris Smaragdis  
MIT, Cambridge  
1995

# exprandi

exprandi — Générateur de nombres aléatoires de distribution exponentielle avec interpolation (valeurs positives seulement).

## Description

Générateur de nombres aléatoires de distribution exponentielle avec interpolation contrôlée entre les valeurs (valeurs positives seulement). C'est un générateur de bruit de classe x.

## Syntaxe

```
ares exprandi klambda, xamp, xcps
ires exprandi klambda, xamp, xcps
kres exprandi klambda, xamp, xcps
```

## Exécution

*klambda* -- paramètre lambda de la distribution exponentielle.

La densité de probabilité d'une distribution exponentielle est une courbe exponentielle dont la moyenne vaut  $0.69515/\lambda$ . Pour des explications plus détaillées, voir :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

*xamp* -- intervalle de distribution des nombres aléatoires.

*xcps* -- fréquence à laquelle de nouveaux nombres sont générés.

## Exemples

Voici un exemple de l'opcode exprandi. Il utilise le fichier *exprandi.csd* [examples/exprandi.csd].

### Exemple 262. Exemple de l'opcode exprandi.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime output leave only the line below:
; -o exprand.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
```

```
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
klambda exprandi 20, 1, 3
  printk2 klambda ; look
aout oscili 0.8, 440+klambda, 1 ; & listen
  outs aout, aout
endin

</CsInstruments>
<CsScore>
; sine wave
f 1 0 16384 10 1

i 1 0 4
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*exprand*

## Crédits

Auteur : John ffitch  
Bath  
Mai 2011  
Nouveau dans la version 5.14

# expseg

expseg — Trace une suite de segments d'exponentielle entre les points spécifiés.

## Description

Trace une suite de segments d'exponentielle entre les points spécifiés.

## Syntaxe

```
ares expseg ia, idur1, ib [, idur2] [, ic] [...]  
kres expseg ia, idur1, ib [, idur2] [, ic] [...]
```

## Initialisation

*ia* -- valeur initiale. Zéro est interdit pour les exponentielles.

*ib*, *ic*, etc. -- valeur après *dur1* secondes, etc. Pour les exponentielles, doivent être différentes de zéro et du même signe que *ia*.

*idur1* -- durée en secondes du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

*idur2*, *idur3*, etc. -- durée en secondes des segments suivants. Une valeur nulle ou négative terminera la phase d'initialisation avec le point précédent, permettant au dernier segment défini de continuer durant toute l'exécution. La valeur par défaut est zéro.

## Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par 2 ou plus points spécifiés. La somme des valeurs *dur* peut égaler ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, le dernier segment défini continuera dans la même direction.

Noter que l'opcode *expseg* n'opère pas correctement au taux audio lorsque les segments sont plus courts qu'une k-période. Dans ce cas, il vaut mieux utiliser l'opcode *expsega*.

## Exemples

Voici un exemple de l'opcode expseg. Il utilise le fichier *expseg.csd* [examples/expseg.csd].

### Exemple 263. Exemple de l'opcode expseg.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
; Audio out  Audio in  
-odac          -iadc      ;;RT audio I/O
```

```
; For Non-realtime ouput leave only the line below:
; -o expseg.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; p4 = frequency in pitch-class notation.
kcps = cpspch(p4)

; Create an amplitude envelope.
kenv expseg 0.01, p3*0.25, 1, p3*0.75, 0.01
kamp = kenv * 30000

a1 oscil kamp, kcps, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for a half-second, p4=8.00
i 1 0 0.5 8.00
; Play Instrument #1 for a half-second, p4=8.01
i 1 1 0.5 8.01
; Play Instrument #1 for a half-second, p4=8.02
i 1 2 0.5 8.02
; Play Instrument #1 for a half-second, p4=8.03
i 1 3 0.5 8.03
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*expon, expsega, expsegr, line, linseg, linsegr transeg*

## Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

Nouveau dans Csound 3.57

# expsega

expsega — Un générateur de segments exponentiels opérant au taux-a.

## Description

Un générateur de segments exponentiels opérant au taux-a. Cette unité est pratiquement identique à *expseg*, mais elle est plus précise lorsque l'on définit des segments de courte durée (c-à-d., dans une phase d'attaque percussive) au taux audio.

## Syntaxe

```
ares expsega ia, idur1, ib [, idur2] [, ic] [...]
```

## Initialiation

*ia* -- valeur initiale. Zéro est interdit.

*ib*, *ic*, etc. -- valeur après *idur1* secondes, etc. Doivent être non nulles et de même signe que *ia*.

*idur1* -- durée en secondes du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

*idur2*, *idur3*, etc. -- durée en secondes des segments suivants. Une valeur nulle ou négative terminera la phase d'initialisation avec le point précédent, permettant au dernier segment défini de continuer durant toute l'exécution. La valeur par défaut est zéro.

## Exécution

Cette unité génère des signaux audio dont les valeurs passent par 2 ou plus points spécifiés. La somme des valeurs *dur* peut égaier ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, le dernier segment défini continuera dans la même direction.

## Exemples

Voici une exemple de l'opcode expsega. Il utilise le fichier *expsega.csd* [examples/expsega.csd].

### Exemple 264. Exemple de l'opcode expsega.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o expsega.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
    ; Define a short percussive amplitude envelope that
    ; goes from 0.01 to 20,000 and back.
    aenv expsega 0.01, 0.1, 20000, 0.1, 0.01

    a1 oscil aenv, 440, 1
    out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #1 for one second.
i 1 1 1
; Play Instrument #1 for one second.
i 1 2 1
; Play Instrument #1 for one second.
i 1 3 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*expseg*, *expsegr*

## Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

Nouveau dans Csound 3.57



# expsegb

expsegb — Trace une suite de segments d'exponentielle entre les points absolus spécifiés.

## Description

Trace une suite de segments d'exponentielle entre les points absolus spécifiés.

## Syntaxe

```
ares expsegb ia, itim1, ib [, itim2] [, ic] [...]  
kres expsegb ia, itim1, ib [, itim2] [, ic] [...]
```

## Initialisation

*ia* -- valeur initiale. Zéro est interdit pour les exponentielles.

*ib*, *ic*, etc. -- valeurs à *tim1* secondes, etc. Pour les exponentielles doivent être différentes de zéro et de même signe que *ia*.

*itim1* -- date en secondes de la fin du premier segment.

*itim2*, *itim3*, etc. -- date en secondes de la fin des segments suivants.

## Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par 2 ou plus points spécifiés. La dernière valeur *tim* peut égal ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, la dernière valeur sera répétée jusqu'à la fin de la note.

Noter que l'opcode *expsegb* n'opère pas correctement au taux audio s'il y a des segments inférieurs à une période-k. Dans ce cas, essayez plutôt l'opcode *expsegba* opcode instead.

## Exemples

Voici un exemple de l'opcode expsegb. Il utilise le fichier *expsegb.csd* [examples/expsegb.csd].

### Exemple 265. Exemple de l'opcode expsegb.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
; Audio out  Audio in  
-odac          -iadc      ;;RT audio I/O  
; For Non-realtime ouput leave only the line below:  
; -o expseg.wav -W ;; for file output any platform  
</CsOptions>
```

```

<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; p4 = frequency in pitch-class notation.
kcps = cpspch(p4)

; Create an amplitude envelope.
kenv expsegb 0.01, p3*0.25, 1, p3, 0.01
kamp = kenv * 30000

a1 oscil kamp, kcps, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for a half-second, p4=8.00
i 1 0 0.5 8.00
; Play Instrument #1 for a half-second, p4=8.01
i 1 1 0.5 8.01
; Play Instrument #1 for a half-second, p4=8.02
i 1 2 0.5 8.02
; Play Instrument #1 for a half-second, p4=8.03
i 1 3 0.5 8.03
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*expon, expseg, expsega, expsegr, line, linseg, linsegr transeg*

## Crédits

Auteur : Victor Lazzarini  
Juin 2011

Nouveau dans la version 5.14

# expsegba

expsegba — Un générateur de segments d'exponentielle opérant au taux-a avec des temps absolus.

## Description

Un générateur de segments d'exponentielle opérant au taux-a. Cette unité est presque identique à *expsegb*, mais plus précises lorsque sont définis des segments de très courte durée (par exemple dans une phase d'attaque percussive) au taux audio.

## Syntaxe

```
ares expsegba ia, itim1, ib [, itim2] [, ic] [...]
```

## Initialisation

*ia* -- valeur initiale. Zéro est interdit.

*ib*, *ic*, etc. -- valeurs après *itim1* secondes, etc. Doivent être différentes de zéro et de même signe que *ia*.

*itim1* -- date en secondes de la fin du premier segment.

*itim2*, *itim3*, etc. -- date en secondes de la fin des segments suivants.

## Exécution

Cet unité génère des signaux audio dont les valeurs peuvent passer par deux ou plus points spécifiés. La dernière valeur *tim* peut égaliser ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, la dernière valeur sera répétée jusqu'à la fin de la note.

## Exemples

Voici un exemple de l'opcode expsegba. Il utilise le fichier *expsegba.csd* [examples/expsegba.csd].

### Exemple 266. Exemple de l'opcode expsegba.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o expsega.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
```

```
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Define a short percussive amplitude envelope that
; goes from 0.01 to 20,000 and back.
aenv expsegba 0.01, 0.1, 20000, 0.2, 0.01

a1 oscil aenv, 440, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #1 for one second.
i 1 1 1
; Play Instrument #1 for one second.
i 1 2 1
; Play Instrument #1 for one second.
i 1 3 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*expsegb, expsegr*

## Crédits

Auteur : John ffitich

Juin 2011

Nouveau dans Csound 5.14

# expsegr

**expsegr** — Trace une suite de segments d'exponentielle entre les points spécifiés avec un segment de relâchement.

## Description

Trace une suite de segments d'exponentielle entre les points spécifiés avec un segment de relâchement (fin de l'entretien de la note).

## Syntaxe

```
ares expsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz  
kres expsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
```

## Initialisation

*ia* -- valeur initiale. Zéro est interdit pour les exponentielles.

*ib*, *ic*, etc. -- valeur après *dur1* secondes, etc. Pour les exponentielles, doivent être différentes de zéro et du même signe que *ia*.

*idur1* -- durée en secondes du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

*idur2*, *idur3*, etc. -- durée en secondes des segments suivants. Une valeur nulle ou négative terminera la phase d'initialisation avec le point précédent, permettant au dernier segment défini de continuer durant toute l'exécution. La valeur par défaut est zéro.

*irel*, *iz* -- durée en secondes et valeur finale du segment de relâchement de la note.

## Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par 2 ou plus points spécifiés. La somme des valeurs *dur* peut égaler ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, le dernier segment défini continuera dans la même direction.

*expsegr* fait partie des unités « r » de Csound qui contiennent un détecteur de fin de note et une extension de durée pour le relâchement. Quand la fin d'un événement ou MIDI noteoff est détectée, la durée d'exécution de l'instrument courant est immédiatement allongée de *irel* secondes, de façon à ce que la valeur *iz* soit atteinte à la fin de cette période (quelque soit le segment dans lequel se trouvait l'unité). Les unités « r » peuvent aussi être modifiées par les vitesses nulles provoquant un message MIDI noteoff. S'il y a plusieurs extensions de durée dans un instrument, c'est la plus longue qui sera choisie.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note off, comme *linsegr* et *madsr*, ou bien l'on peut construire des enveloppes plus complexes au moyen de *xtratim* et de *release*. Noter que qu'il n'est pas nécessaire d'utiliser *xtratim* avec *expsegr*, car la durée est allongée automatiquement.

## Exemples

Voici un exemple de l'opcode `expsegr`. Il utilise le fichier `expsegr.csd` [examples/expsegr.csd].

### Exemple 267. Exemple de l'opcode `expsegr`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0   ;;realtime audio out and realtime midi in
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o expsegr.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

icps cpsmidi
iamp ampmidi .3

kenv expsegr 1, .05, 0.5, 1, .01
asig pluck kenv, icps, 200, 1, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 4096 10 1 ;sine wave

f0 30 ;runs 30 seconds
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*linsegr, expsegr, envlpxr, mxadsr, madsr expon, expseg, expsega, xtratim*

## Crédits

Auteur : Barry L. Vercoe

Nouveau dans Csound 3.47

# faustaudio

faustaudio — Instancie et exécute un programme Faust compilé.

## Description

Opcodes du greffon faustcsound.

*faustaudio* instancie et exécute un programme Faust compilé avec *faustcompile*.

## Syntaxe

```
ihandle, a1 [,a2, ...] faustaudio ifac [,ain1, ...]
```

## Initialisation

« *ifac* » -- une référence à un programme Faust compilé, produit par *faustcompile*.

« *ihandle* » -- une référence à l'instance DSP de Faust que l'on peut utiliser pour accéder à ses contrôles avec *faustctl*.

## Exécution

« *ain1,...* » -- signaux d'entrée

« *a1,...* » -- signaux de sortie

## Exemples

Voici un exemple de l'opcode faustaudio. Il utilise le fichier *faustaudio.csd* [examples/faustaudio.csd].

### Exemple 268. Exemple de l'opcode faustaudio.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>
instr 1
  a1 rand 0dbfs/4
  a2 oscili 0dbfs/4, 440
  ihandle faustcompile "process==;", "-vec -lv 1"
  idsp,asig faustaudio ihandle,a1,a2
    out asig
endin
</CsInstruments>
<CsScore>
i1 0 10
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini, 2013



# faustcompile

**faustcompile** — Invoque le compilateur à la volée pour produire un processus DSP instanciable depuis un programme Faust.

## Description

Opcodes du greffon `faustcsound`.

*faustcompile* compile un programme Faust contenu dans une chaîne de caractères, contrôlé par divers arguments. On peut utiliser des chaînes sur plusieurs lignes, délimitées par `{ { }`.

## Syntaxe

```
ihandle faustcompile Scode, Sargs[, iasync, istacksize]
```

## Initialisation

« *Scode* » -- une chaîne de caractères (entre guillemets ou délimitée par `{ { }`) contenant un programme Faust.

« *Sargs* » -- une chaîne de caractères (entre guillemets ou délimitée par `{ { }`) contenant les arguments du compilateur Faust.

« *iasync* » -- exécute le code de manière asynchrone, non bloquante, (*iasync*=1), ou en mode bloquant (*iasync*=0). Vaut 1 par défaut.

« *istacksize* » -- taille de la pile du fil d'exécution du compilateur en MO (vaut 1 par défaut).

## Exemples

Voici un exemple de l'opcode `faustcompile`. Il utilise le fichier *faustcompile.csd* [exemples/faustcompile.csd].

### Exemple 269. Exemple de l'opcode `faustcompile`.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>
instr 1
a1 rand 0dbfs/4
a2 oscili 0dbfs/4, 440
ihandle faustcompile "process==;", "-vec -lv 1"
idsp,asig faustaudio ihandle,a1,a2
out asig
endin
</CsInstruments>
<CsScore>
i1 0 10
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini, 2013

# faustctl

faustctl — Ajuste un contrôle donné dans une instance DSP de Faust.

## Description

Opcode du greffon faustcsound.

*faustctl* fixe un contrôle donnée dans un programme Faust actif.

## Syntaxe

```
faustctl idsp, Scontrol, kval
```

## Initialisation

« *Scontrol* » -- une chaîne de caractères contenant le nom du contrôle.

« *idsp* » -- une référence à une instance DSP de Faust existante.

## Exécution

« *kval* » -- valeur à laquelle le contrôle est fixé.

## Exemples

Voici un exemple de l'opcode faustctl. Il utilise le fichier *faustctl.csd* [examples/faustctl.csd].

### Exemple 270. Example of the faustctl opcode.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>
instr 1
  ain1 oscili 0dbfs/2, 440
  idsp, a1 faustgen {{
    gain = hslider("vol", 1, 0, 1, 0.01);
    process = (_ * gain);
  }}, ain1
  k1 line 0, p3, 1
  faustctl idsp, "vol", k1
  out a1
endin
</CsInstruments>
<CsScore>
i1 0 10
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini, 2013

# faustdsp

faustdsp — Instancie un programme Faust.

## Description

Opcodes du greffon faustcsound.

Faustdsp instancie un programme Faust compilé avec faustcompile.

## Syntaxe

```
ihandle faustdsp ifac
```

## Initialisation

« *ifac* » -- identifiant d'un programme Faust compilé, produit par faustcompile.

« *ihandle* » -- identifiant d'une instance Faust DSP qui peut être utilisé pour exécuter un programme avec faustplay et pour accéder à ses contrôles avec faustctl.

## Exemples

Voici un exemple de l'opcode faustdsp. Il utilise le fichier *faustdsp.csd* [examples/faustdsp.csd].

### Exemple 271. Exemple de l'opcode faustdsp.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-d -odac
</CsOptions>
<CsInstruments>
nchnls= 1

giph faustcompile {{
SR = 44100;
decimal(a) = a - floor(a);
freq = hslider("freq", 0, -20000, 20000, 1);
incr(fr) = fr / float(SR);
phasor(fr,ph) = incr(fr) : (+ : decimal) ~ _ :
+(ph) : decimal;
process = phasor(freq,0);
}}, "-vec -lv 1"

instr 1
kb1 = p5
ib faustdsp giph
faustctl ib,"freq",kb1
asig faustplay ib
out sin(2*$M_PI*asig)*p4*0dbfs
endin
```

```
</CsInstruments>  
<CsScore>  
i1 0 1 0.5 150  
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini, 2018

# faustgen

faustgen — Compile, instancie et exécute un programme Faust compilé.

## Description

Opcodes du greffon faustcsound.

*faustgen* invoque le compilateur à la volée, instancie et exécute un programme Faust.

## Syntaxe

```
ihandle, a1 [,a2, ...] faustgen SCode [,ain1, ...]
```

## Initialisation

« *Scode* » -- une chaîne de caractères contenant un programme Faust.

« *ihandle* » -- une référence à l'instance DSP de Faust que l'on peut utiliser pour accéder à ses contrôles avec *faustctl*.

## Exécution

« *ain1, ...* » -- signaux d'entrée

« *a1, ...* » -- signaux de sortie

## Exemples

Voici un exemple de l'opcode *faustgen*. Il utilise le fichier *faustgen.csd* [examples/faustgen.csd].

### Exemple 272. Exemple de l'opcode *faustgen*.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>
instr 1
  ain1 oscili 0dbfs/2, 440
  idsp,a1 faustgen {{
    gain = hslider("vol",1,0,1,0.01);
    process = (_ * gain);
  }}, ain1
  k1 line 0, p3, 1
  faustctl idsp, "vol", k1
  out a1
endin
</CsInstruments>
<CsScore>
i1 0 10
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini, 2013



# faustplay

faustplay — Exécute un programme Faust instancié.

## Description

Opcodes du greffon faustcsound.

Faustplay exécute un programme Faust instancié avec faustdsp.

## Syntaxe

```
a1[, a2,...] faustplay ihandle[, ain1,...]
```

## Initialisation

« *ihandle* » -- identifiant de l'instance Faust DSP provenant de faustdsp.

## Exécution

« *ain1,...* » -- signaux en entrée.

« *a1,...* » -- signaux de sortie.

## Exemples

Voici un exemple de l'opcode faustplay. Il utilise le fichier *faustplay.csd* [examples/faustplay.csd].

### Exemple 273. Exemple de l'opcode faustplay.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-d -odac
</CsOptions>
<CsInstruments>
nchnls= 1

giph faustcompile {{
SR = 44100;
decimal(a) = a - floor(a);
freq = hslider("freq", 0, -20000, 20000, 1);
incr(fr) = fr / float(SR);
phasor(fr,ph) = incr(fr) : (+ : decimal) ~ _ :
+(ph) : decimal;
process = phasor(freq,0);
}}, "-vec -lv 1"

instr 1
kb1 = p5
ib faustdsp giph
faustctl ib,"freq",kb1
```

```
asig faustplay ib
out sin(2*$M_PI*asig)*p4*0dbfs
endin
```

```
</CsInstruments>
<CsScore>
i1 0 1 0.5 150
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini, 2018

# fareylen

fareylen — retourne la longueur d'une suite de Farey.

## Description

On peut utiliser cet opcode de concert avec *GENfarey*. Il calcule la longueur de la suite de Farey  $F_n$ . Cette longueur est donnée par :  $|F_n| = 1 + \text{SOMME sur } n \text{ phi}(m)$  où  $\text{phi}(m)$  est l'indicatrice d'Euler, qui donne le nombre d'entiers  $\leq m$  premiers avec  $m$ .

Quelques valeurs de la longueur de  $F_n$  en fonction de  $n$  :

<b>n</b>	<b><math>F_n</math></b>
1	2
2	3
3	5
4	7
5	11
6	13
7	19
8	23
9	29
10	33
11	43
12	47
13	59
14	65
15	73
16	81
17	97
18	103
19	121
20	129

## Syntaxe

kfl **fareylen** kfn

## Exécution

La longueur de la suite de Farey identifiée est retournée.

*kfn* -- Entier identifiant la suite.

## Crédits

Auteur : Georg Boenn  
Université de Glamorgan, UK

Nouveau dans la version 5.13 de Csound.

# fareyleni

fareyleni — retourne la longueur d'une suite de Farey.

## Description

On peut utiliser cet opcode de concert avec *GENfarey*. Il calcule la longueur de la suite de Farey  $F_n$ . Cette longueur est donnée par :  $|F_n| = 1 + \text{SOMME sur } n \text{ phi}(m)$  où  $\text{phi}(m)$  est l'indicatrice d'Euler, qui donne le nombre d'entiers  $\leq m$  premiers avec  $m$ .

Quelques valeurs de la longueur de  $F_n$  en fonction de  $n$  :

<b>n</b>	<b>F<sub>n</sub></b>
1	2
2	3
3	5
4	7
5	11
6	13
7	19
8	23
9	29
10	33
11	43
12	47
13	59
14	65
15	73
16	81
17	97
18	103
19	121
20	129

## Syntaxe

```
ifl fareyleni ifn
```

## Initialisation

La longueur de la suite de Farey identifiée est retournée.

*ifn* -- Entier identifiant la suite.

## Exemples

Voici un exemple de l'opcode fareyleni. Il utilise le fichier *fareyleni.csd* [examples/fareyleni.csd].

### Exemple 274. Exemple de l'opcode fareyleni.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc for RT audio input as well
; For Non-realtime ouput leave only the line below:
; -o fareyleni.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

; initialise integer n for Farey Sequence F_8
gifarn init 8

; calculate length of F_8, it should return 23 for |F_8|
gires fareyleni gifarn

; convert to negative number for the GEN routine because
; the table length won't be a power of 2
; (The length of a Farey Sequence of n > 1 is always an odd number)
gilen init gires * -1

; create F_8 with GENfarey, the negative table number prevents
; unnecessary normalisation (F_8 IS already normalised)
;          n      mode:
gifarey ftgen 100, 0, gilen, "farey", gifarn, 4
; if mode=4 then 1 is added to each element of F_n.
; Useful for creating just tuning tables that can be read by the cps2pch opcode.

instr 1
; the very last element of F_n is not needed in the case of tuning tables
ires = gires - 1
; read out and print to file
kndx init 0
if (kndx < ires) then
kelem tab kndx, gifarey
fprintks "farey8_tuning.txt", "%2.6f\\n", kelem
kndx = kndx+1
endif

endin

instr 2

ip cps2pch p4, -100
asig poscil .5, ip, 1
aenv linseg 0, 0.1, 1, p3-0.2, 1, 0.1, 0
outs asig * aenv, asig * aenv

endin
</CsInstruments>
```

```
<CsScore>
f1 0 8192 10 1 ;sine wave

i1 0      .1

i2 1 .5 8.00
i2 + . >
i2 + . >
i2 + . >
i2 + . >
i2 + . >
i2 + . >
i2 + . >
i2 + . >
i2 + . >
i2 + . >
i2 + . >
i2 + . >
i2 + . >
i2 + . >
i2 + . >
i2 + . >
i2 + . >
i2 + . >
i2 + . >
i2 + 1 8.22

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Georg Boenn  
Université de Glamorgan, UK

Nouveau dans la version 5.13 de Csound.

# ficlose

ficlose — Ferme un fichier ouvert précédemment.

## Description

*ficlose* peut être utilisé pour fermer un fichier qui avait été ouvert avec *fiopen*.

## Syntaxe

```
ficlose ihandle  
ficlose Sfilename
```

## Initialisation

*ihandle* -- un nombre qui identifie le fichier (généré par un *fiopen* précédent).

*Sfilename* -- une chaîne de caractères entre guillemets ou une variable chaîne de caractères contenant le nom du fichier. Il faut donner le chemin complet si le répertoire du fichier n'est pas dans le PATH (chemin) du système et s'il n'est pas dans le répertoire courant.

## Exécution

*ficlose* ferme un fichier ouvert précédemment avec *fiopen*. *ficlose* n'est nécessaire que si l'on désire lire un fichier écrit durant la même exécution de Csound, car Csound ne sauve les données dans tous les fichiers ouverts et ne ferme ceux-ci que lorsqu'il termine une exécution. L'opcode *ficlose* est utile par exemple si l'on veut sauvegarder des presets dans des fichiers auxquels on veut pouvoir accéder sans terminer Csound.



### Note

Si l'on a pas besoin de cette fonctionnalité, il est plus sûr de ne pas appeler *ficlose*, et de laisser Csound fermer les fichiers lorsqu'il se termine.

Si un fichier fermé par *ficlose* est accédé par un autre opcode (comme *fout* ou *foutk*), il sera fermé plus tard lorsqu'il ne sera plus utilisé.



### Avertissement

Il faut utiliser cet opcode avec précaution, car l'identificateur de fichier n'est plus valide, et il y aura une erreur d'initialisation si un opcode essaie d'accéder au fichier fermé.

## Exemples

Voici un exemple de l'opcode *ficlose*. Il utilise le fichier *ficlose.csd* [exemples/ficlose.csd].

### Exemple 275. Exemple de l'opcode *ficlose*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.



```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ficlose.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gihand fiopen "test1.txt", 0

instr 1

ires random 0, 100
fouti gihand, 0, 1, ires
ficlose gihand

endin
</CsInstruments>
<CsScore>

i 1 0 1

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*fiopen, fout, fouti, foutir, foutk*

## Crédits

Auteur : Gabriel Maldonado  
 Italie  
 1999

Nouveau dans la version 5.02 de Csound

# filebit

filebit — Retourne le nombre de bit de chaque échantillon d'un fichier son.

## Description

Retourne le nombre de bit de chaque échantillon d'un fichier son.

## Syntaxe

```
ir filebit ifilcod [, iallowraw]
```

## Initialisation

*ifilcod* -- fichier son à interroger.

*iallowraw* -- (Facultatif) autorise les fichiers son bruts (vaut 1 par défaut)

## Exécution

*filebit* retourne le nombre de bit de chaque échantillon du fichier son *ifilcod*. Dans le cas d'échantillons en virgule flottante la valeur -1 est retournée pour des flottants et -2 pour des doubles. Pour les formats non-PCM, la valeur est négative et basée sur le format d'encodage de libsndfile.

## Exemples

Voici un exemple de l'opcode filebit. Il utilise les fichiers *filebit.csd* [examples/filebit.csd], et *mary.wav* [examples/mary.wav].

### Exemple 276. Exemple de l'opcode filebit.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o filebit.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Print out the number of channels in the
; audio file "mary.wav".
```

```

    ibits filebit "mary.wav"
    print ibits
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 1 second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Le fichier audio « mary.wav » est au format CD mono, ce qui fait que la sortie de *filebit* comprendra une ligne comme celle-ci :

```
instr 1:  ibits = 16.000
```

## Voir aussi

*filelen*, *filenchnls*, *filepeak*, *filesr*

## Crédits

Auteur : Victor Lazzarini  
Juillet 1999

Exemple écrit par John ffitich.

Nouveau dans la version 5.11 de Csound

# filelen

filelen — Retourne la longueur d'un fichier son.

## Description

Retourne la longueur d'un fichier son.

## Syntaxe

```
ir filelen ifilcod, [iallowraw]
```

## Initialisation

*ifilcod* -- fichier son à interroger

*iallowraw* -- (facultatif) permet des fichiers son bruts (vaut 1 par défaut)

## Exécution

*filelen* retourne la longueur du fichier son *ifilcod* en secondes. *filelen* peut retourner la longueur des fichiers de type convolve et PVOC si le paramètre *iallowraw* est différent de zéro (il est non nul par défaut).

## Exemples

Voici un exemple de l'opcode *filelen*. Il utilise les fichiers *filelen.csd* [examples/filelen.csd], *fox.wav* [examples/fox.wav] et *kickroll.wav* [examples/kickroll.wav].

### Exemple 277. Exemple de l'opcode *filelen*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
;-odac      ;;realtime audio out
-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o filelen.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; choose between mono or stereo file

ilen  filelen p4 ;calculate length of soundfile
print ilen
ichn filelenchnls p4 ;check number of channels
;print ichn
```

```
if (ichn == 1) then
;mono signal
asig diskinn2 p4, 1
    outs      asig, asig

else
;stereo signal
aL, aR diskinn2 p4, .5, 0, 1
    outs      aL, aR

endif

endin
</CsInstruments>
<CsScore>

i 1 0 3 "fox.wav" ;mono signal
i 1 5 2 "kickroll.wav" ;stereo signal

e
</CsScore>
</CsoundSynthesizer>
```

Le fichier audio mono « fox.wav » dure 2.8 secondes et le fichier stéréo « kickroll.wav » dure 0.9 secondes. Ainsi la sortie de *filelen* contiendra une ligne pour le fichier mono et pour le fichier stéréo comme ceci :

```
instr 1:  ilen = 2.757
instr 1:  ilen = 0.857
```

## Voir aussi

*filebit, filenchnls, filepeak, filesr*

## Crédits

Auteur : Matt Ingalls  
Juillet 1999

Nouveau dans la version 3.57 de Csound

# filenchnls

filenchnls — Retourne le nombre de canaux d'un fichier son.

## Description

Retourne le nombre de canaux d'un fichier son.

## Syntaxe

```
ir filenchnls ifilcod [, iallowraw]
```

## Initialisation

*ifilcod* -- fichier son à interroger

*iallowraw* -- (facultatif) permet des fichiers son bruts (vaut 1 par défaut)

## Exécution

*filenchnls* retourne le nombre de canaux du fichier son *ifilcod*. *filenchnls* peut retourner le nombre de canaux des fichiers de type convolve et PVOC si le paramètre *iallowraw* est différent de zéro (il est non nul par défaut).

## Exemples

Voici un exemple de l'opcode *filenchnls*. Il utilise les fichiers *filenchnls.csd*, [examples/filenchnls.csd]*fox.wav* [examples/fox.wav] et *kickroll.wav* [examples/kickroll.wav].

### Exemple 278. Exemple de l'opcode *filenchnls*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
;-odac      ;;realtime audio out
-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o filechnls.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; choose between mono or stereo file

ilen  filelen p4 ;calculate length of soundfile
;print ilen
ichn  filenchnls p4 ;check number of channels
```

```

print   ichn

if (ichn == 1) then
  ;mono signal
  asig diskin2 p4, 1
      outs      asig, asig

  else
    ;stereo signal
    aL, aR diskin2 p4, .5, 0, 1
        outs      aL, aR

  endif

endin
</CsInstruments>
<CsScore>

i 1 0 3 "fox.wav" ;mono signal
i 1 5 2 "kickroll.wav" ;stereo signal

e
</CsScore>
</CsoundSynthesizer>

```

Le fichier audio « fox.wav » est monophonique (1 canal), tandis que « kickroll.wav » est stéréophonique (2 canaux). Ainsi la sortie de *filenchnls* contiendra des lignes comme :

```

instr 1:   ichn = 1.000
instr 1:   ichn = 2.000

```

## Voir aussi

*filebit, filelen, filepeak, filesr*

## Crédits

Auteur : Matt Ingalls  
Juillet 1999

Nouveau dans la version 3.57 de Csound

# filepeak

filepeak — Retourne la valeur absolue de la crête d'un fichier son.

## Description

Retourne la valeur absolue de la crête d'un fichier son.

## Syntaxe

```
ir filepeak ifilcod [, ichnl]
```

## Initialisation

*ifilcod* -- fichier son à interroger

*ichnl* (facultatif, 0 par défaut) -- canal sur lequel la valeur de crête est calculée. La valeur par défaut est 0.

- *ichnl* = 0 retourne la valeur de crête de tous les canaux
- *ichnl* > 0 retourne la valeur de crête de *ichnl*

## Exécution

*filepeak* retourne la valeur absolue de la crête du fichier son *ifilcod*.

## Exemples

Voici un exemple de l'opcode filepeak. Il utilise les fichiers *filepeak.csd* [examples/filepeak.csd] et *Church.wav* [examples/Church.wav].

### Exemple 279. Exemple de l'opcode filepeak.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o filepeak.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

iscaldb = p4 ;set peak amplitude in dB
```



```
ipeak    filepeak "Church.wav"
iscal    = ampdb(iscaldB)/ipeak ;calculate amp multiplier
printf_i "\nPeak value in file '%s' is %f (%.3f dB).\n\n", 1, "Church.wav", ipeak, dbamp(ipeak)

asnd soundin "Church.wav"
outs asnd, asnd
; scale & write file to disk
asig = asnd*iscal ;scale to p4
fout "Church_norm.wav", 14, asig

endin

instr 2 ; play scaled file

aout soundin "Church_norm.wav"
ipknew filepeak "Church_norm.wav"
printf_i "\nPeak value in file '%s' is %f (%.3f dB).\n\n", 1, "Church_norm.wav", ipknew, dbamp(ipknew)
outs aout, aout

endin
</CsInstruments>
<CsScore>

i 1 0 2 -6 ; normalize audio to -6 dB
i 2 2 2
e
</CsScore>
</CsoundSynthesizer>
```

La sortie de *filepeak* contiendra des lignes comme celles-ci :

```
Peak value in file 'Church.wav' is 0.909363 (-0.825 dB).
Peak value in file 'Church_norm.wav' is 0.501190 (-6.000 dB).
```

## Voir aussi

*filelen, filenchnls, filesr*

## Crédits

Auteur : Matt Ingalls  
Juillet 1999

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.57 de Csound

# filesal

*filesal* — Traitement vocoder à verrouillage de phase avec détection/traitement d'attaque, 'pondération du tempo'.

## Description

*filesal* implémente un traitement vocoder à verrouillage de phase à partir de fichiers sur disque, avec reéchantillonnage si nécessaire.

Cet opcode permet une pondération indépendante du temps et de la fréquence. Le temps progresse en interne, contrôlé par un paramètre facteur d'échelle du tempo ; quand une attaque est détectée, la pondération temporelle est momentanément interrompue pour éviter une dégradation des attaques. La qualité de l'effet est généralement améliorée lorsque le verrouillage de phase est actif.

*filesal* pondère également la hauteur, indépendamment de la fréquence, avec un facteur de transposition de taux-k.

## Syntaxe

```
asig[, asig2] filesal ktimescal, kamp, kpitch, Sfile, klock [,ifftsize, idecim, ithresh]
```

## Initialisation

*Sfile* -- fichier son source ; des fichiers mono ou stéréo sont acceptés, mais ils doivent correspondre au nombre de sorties.

*ifftsize* -- taille de TFR (puissance de deux), 2048 par défaut.

*idecim* -- décimation, 4 par défaut (ce qui signifie hopsize = fftsize/4).

*idbthresh* -- seuil basé sur le rapport en dB des spectres de puissance entre deux fenêtres successives. Un rapport détecté supérieur à celui-ci annule momentanément la pondération temporelle, pour éviter une dégradation (vaut 1 par défaut).

## Exécution

*ktimescal* -- rapport de pondération temporelle, < 1 étirement, > 1 contraction. Nombres non-négatifs seulement.

*kamp* -- pondération de l'amplitude.

*kpitch* -- pondération de la hauteur des grains (1 = hauteur normale, < 1 inférieure, > 1 supérieure ; négative, lecture inversée).

*klock* -- active (valeur différente de zéro) ou désactive (zéro) le verrouillage de phase.

## Exemples

Voici un exemple de l'opcode *filesal*. Il utilise le fichier *filesal.csd* [examples/filesal.csd].

**Exemple 280. Exemple de l'opcode filescal.**

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

instr 1
p3 = filelen("fox.wav")/p4
k1 linen 1,0.01,p3,0.1
a1 filescal p4,0.5,1,"fox.wav",1
  out a1*k1
endin

</CsInstruments>
<CsScore>
i1 0 1 1.25
i1 2.5 1 .75
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
Avril 2016

Nouveau dans la version 6.07

Avril 2016.

# filesr

filesr — Retourne le taux d'échantillonnage d'un fichier son.

## Description

Retourne le taux d'échantillonnage d'un fichier son.

## Syntaxe

```
ir filesr ifilcod [, iallowraw]
```

## Initialisation

*ifilcod* -- fichier son à interroger

*iallowraw* -- (facultatif) permet des fichiers son bruts (vaut 1 par défaut)

## Exécution

*filesr* retourne le taux d'échantillonnage du fichier son *ifilcod*. *filesr* peut retourner le taux d'échantillonnage des fichiers de type convolve et PVOC si le paramètre *iallowraw* est différent de zéro (il est non nul par défaut).

## Exemples

Voici un exemple de l'opcode *filesr*. Il utilise les fichiers *filesr.csd* [examples/filesr.csd] et *beats.wav* [examples/beats.wav].

### Exemple 281. Exemple de l'opcode *filenchnls*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o filesr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

;after an example from Jonathan Murphy

instr 1
;load sound into an ftable
Sfile      strcpy      "beats.wav"
```

```
ilen      filelen  Sfile
isr       filesr   Sfile
isamps    = ilen * isr
;adjust the length of the table to be a power of two closest
;to the actual size of the sound
isize     init     1
loop:
  isize    = isize * 2
  if (isize < isamps) igoto loop
  itab     ftgen    0, 0, isize, 1, Sfile, 0, 0, 0
prints    "sample rate = %f, size = %f\n", isr, isize ;prints them

endin
</CsInstruments>
<CsScore>
i1 0 2
e
</CsScore>
</CsoundSynthesizer>
```

Le fichier audio « beats.wav » a été échantillonné à 44.1 KHz. Ainsi la sortie de *filesr* contiendra une ligne comme :

```
sample rate = 44100.000000, size = 131072.000000
```

## Voir aussi

*filebit, filelen, filenchnls, filepeak*

## Crédits

Auteur : Matt Ingalls  
Juillet 1999

Nouveau dans la version 3.57 de Csound

# filevalid

filevalid — Teste si un fichier peut être lu.

## Description

Retourne 1 si le fichier son existe et s'il est lisible, 0 sinon.

## Syntaxe

```
ir filevalid ifilcod
```

## Initialisation

*ifilcod* -- fichier son à tester

## Exécution

*filevalid* retourne 1 si le fichier son *ifilcod* peut être lu.

## Exemples

Voici un exemple de l'opcode filevalid. Il utilise le fichier *filevalid.csd* [examples/filevalid.csd].

### Exemple 282. Exemple de l'opcode filevalid.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o filevalid.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
Sfile      strget    p4
ivld       filevalid Sfile

if ivld>0 then
asig       diskinn2  Sfile, 1
           outs      asig, asig
else
           printf_i   "Audiofile '%s' does not exist!\n", 1, Sfile
endif
endin
```

```
</CsInstruments>
<CsScore>

i 1 0 3 "frox.wav";file does not exist!!!
i 1 + 3 "fox.wav" ;but this one certainly does...

e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
Audiofile 'frox.wav' does not exist!
```

## Voir aussi

*filebit, filelen, filenchnls, filepeak, filesr*

## Crédits

Auteur : Matt Ingalls  
Juillet 2010

Nouveau dans la version 5.13

# fillarray

fillarray — Crée un vecteur avec initialisation.

## Description

Crée un vecteur (tableau unidimensionnel de taux-k) à partir d'une suite de valeurs numériques ou alpha-numériques.

## Syntaxe

```
karray[] fillarray ival1, ival2,....ivaln  
karray fillarray ival1, ival2,....ivaln  
karray fillarray kval1, kval2,....kvaln
```

## Initialisation

*ival1,...ivaln* -- valeurs à placer dans le vecteur.

Sous la secondes forme, le tableau retourné doit être prédéclaré et ça peut être un tableau multidimensionnel qui sera rempli ligne par ligne.

## Exécution

Dans la troisième forme le vecteur est régénéré au taux-k avec les valeurs *kval1,..., kvaln*.

## Exemples

Voici un exemple de l'opcode fillarray. Il utilise le fichier *fillarray.csd* [examples/fillarray.csd].

### Exemple 283. Exemple de l'opcode fillarray.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-n  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 32  
nchnls = 2  
0dbfs = 1  
  
instr 1  
  
    kS[] fillarray 1,7,5  
    printk 0, kS[0]  
    printk 0, kS[1]
```



```
        printk 0, kS[2]
    turnoff
endin

</CsInstruments>
<CsScore>
i 1 0 1
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

Opcodes vectoriels Opcodes de tableaux.

## Crédits

Auteur : John ffitch  
Codemist Ltd  
2013

Nouveau dans la version 6.00

Troisième forme de taux-k dans la version 6.12

# fft

fft — Transformée de Fourier rapide dans le domaine complexe.

## Description

Applique une transformée de Fourier rapide directe à un tableau unidimensionnel de valeurs complexes produisant une sortie complexe. Cette sortie est un autre tableau contenant les valeurs complexes du signal, et les deux tableaux sont au format réel-imaginaire entrelacé. Le tableau de sortie a la même taille que le tableau d'entrée et la taille de la transformée est équivalente à la moitié de la longueur du tableau. Les transformées non puissance de deux sont restreintes à des tailles paires avec un nombre pas trop élevé de facteurs.

## Syntaxe

```
kout[] fft kin[]
```

## Exécution

*kout[]* -- tableau contenant les valeurs complexes de sortie. Créé s'il n'existe pas.

*kin[]* -- tableau contenant les valeurs complexes d'entrée.

## Exemples

Voici un exemple de l'opcode fft. Il utilise le fichier *fft.csd* [examples/fft.csd].

### Exemple 284. Exemple de l'opcode fft.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-d -o dac
</CsOptions>
<CsInstruments>
ksmps = 64

instr 1
  ifftsize = 1024
  kcnt init 0
  kIn[] init ifftsize
  kOut[] init ifftsize

  a1 oscili 0dbfs/2, 440

  if kcnt >= ifftsize then
    kCmplx[] r2c kIn
    kSpec[] fft kCmplx
    kCmplx fftinv kSpec
    kOut c2r kCmplx
    kcnt = 0
  endif
```

```
kIn[] shiftin a1
a2 shiftout kOut
kcnt += ksmps
    out a2
endin
</CsInstruments>
<CsScore>
i1 0 10
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux.

## Crédits

Auteur : Victor Lazzarini  
NUI Maynooth  
2014

Nouveau dans la version 6.04

## filter2

`filter2` — Réalise un filtrage au moyen d'un bloc de filtre numérique de forme transposée II sans contrôle variable.

## Description

Filtre configurable à usage général sans contrôle variable des pôles. Les coefficients du filtre implémentent l'équation aux différences suivante :

$$(1)*y(n) = b0*x[n] + b1*x[n-1] + \dots + bM*x[n-M] - a1*y[n-1] - \dots - aN*y[n-N]$$

dont la fonction de transfert est représentée par :

$$H(Z) = \frac{B(Z)}{A(Z)} = \frac{b0 + b1*Z^{-1} + \dots + bM*Z^{-M}}{1 + a1*Z^{-1} + \dots + aN*Z^{-N}}$$

## Syntaxe

```
ares filter2 asig, iM, iN, ib0, ib1, ..., ibM, ia1, ia2, ..., iaN
```

```
kres filter2 ksig, iM, iN, ib0, ib1, ..., ibM, ia1, ia2, ..., iaN
```

## Initialisation

A l'initialisation, les nombres de zéros et de pôles du filtres sont spécifiés ainsi que leurs valeurs. Les coefficients doivent être obtenus par une application externe de conception de filtre telle que Matlab et sont spécifiés directement ou bien chargés dans une table via *GEN01*.

## Exécution

L'opcode *filter2* réalise un filtrage au moyen d'un bloc de filtre numérique de forme transposée II sans contrôle variable.

Comme *filter2* implémente des filtres récursifs généralisés, on peut l'utiliser pour définir une grande variété d'algorithmes généraux de traitement numérique du signal. Par exemple, on peut implémenter un guide d'onde numérique pour modéliser un instrument de musique au moyen d'une paire d'opcodes *delayr* et *delayw* conjointement à l'opcode *filter2*.

## Exemples

Un filtre passe-bas RIF du premier ordre à phase linéaire opérant sur un signal de taux-*k* :

```
k1 filter2 ksig, 2, 0, 0.5, 0.5    ;; filtre RIF de taux-k
```

Voici un autre exemple de l'opcode *filter2*. Il utilise le fichier *filter2.csd* [examples/filter2.csd].

## Exemple 285. Exemple de l'opcode filter2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o filter2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 32
nchnls  = 2
0dbfs   = 1

instr 1  ; 2 saw waves of which one is slightly detuned

ib1      = p5
ivol     = p6      ;volume to compensate
kcps     = init cpspch(p4)
asig1    vco2 .05, kcps    ;saw 1
asaw1    filter2 asig1, 1, 1, 1, ib1 ;filter 1
asig2    vco2 .05, kcps+1  ;saw 2
asaw2    filter2 asig2, 1, 1, 1, ib1 ;filter 2
aout     = (asaw1+asaw2)*ivol ;mix
outs     aout, aout

endin
</CsInstruments>
<CsScore>

i 1 0 4 6.00 -.001 5 ;different filter values
i 1 + 4 6.00 -.6 2 ;and different volumes
i 1 + 4 6.00 -.95 .3 ;to compensate
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*zfilter2*

## Crédits

Auteur : Michael A. Casey  
M.I.T.  
Cambridge, Mass.  
1997

Nouveau dans la version 3.47

# fin

*fin* — Lit des signaux depuis un fichier au taux-a.

## Description

Lit des signaux depuis un fichier au taux-a.

## Syntaxe

```
fin ifilename, iskipframes, iformat, ain1 [, ain2] [, ain3] [...]  
fin ifilename, iskipframes, iformat, arr[]
```

## Initialisation

*ifilename* -- nom du fichier d'entrée (peut être une chaîne de caractères ou un identificateur numérique généré par *fiopen*).

*iskipframes* -- nombre de trames à ignorer au début (chaque trame contient un échantillon de chaque canal).

*iformat* -- un nombre spécifiant le format du fichier d'entrée pour les fichiers sans en-tête. Si un en-tête est trouvé, cet argument est ignoré.

- 0 - flottants sur 32 bit sans en-tête
- 1 - entiers sur 16 bit sans en-tête

## Exécution

*fin* (file input) est le complément de *fout* : il lit un fichier multicanaux pour générer des signaux de taux audio. Il faut s'assurer que le nombre de canaux du fichier d'entrée est le même que le nombre d'arguments *ainX*.



### Note

Prière de noter que comme cet opcode génère sa sortie en utilisant des paramètres d'entrée (placés à droite de l'opcode), ces variables doivent avoir été initialisées avant leur utilisation, sinon une erreur "utilisé avant d'être défini" se produira. On peut utiliser l'opcode *init* pour cela.

## Exemples

Voici un exemple de l'opcode *fin*. Il utilise les fichiers *fin.csd* [examples/fin.csd] et *fox.wav* [examples/fox.wav].

### Exemple 286. Exemple de l'opcode *fin*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fin.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

asnd init 0      ;input of fin must be initialized
    fin "fox.wav", 0, 0, asnd ;read audiofile
aenv follow asnd, 0.01 ;envelope follower
kenv downsamp aenv
asig rand kenv    ;gate the noise with audiofile
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 3
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*fini, fink*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1999  
Author: John ffitch  
NUIM, 2013

Nouveau dans la version 3.56 de Csound

Variante pour tableau ajoutée dans la version 6.01

# fini

**fini** — Lit des signaux depuis un fichier au taux-i.

## Description

Lit des signaux depuis un fichier au taux-i.

## Syntaxe

```
fini ifilename, iskipframes, iformat, in1 [, in2] [, in3] [, ...]
```

## Initialisation

*ifilename* -- nom du fichier d'entrée (peut être une chaîne de caractères ou un identificateur numérique généré par *fiopen*).

*iskipframes* -- nombre de trames à ignorer au début (chaque trame contient un échantillon de chaque canal).

*iformat* -- un nombre spécifiant le format du fichier d'entrée pour les fichiers sans en-tête. Si un en-tête est trouvé, cet argument est ignoré.

- 0 - flottants en format texte (avec boucle ; voir ci-dessous)
- 1 - flottants en format texte (sans boucle ; voir ci-dessous)
- 2 - flottants sur 32 bit en format binaire (sans boucle)

## Exécution

*fini* est le complément de *fouti* et de *foutir*. Il lit les valeurs chaque fois qu'une note de l'instrument correspondant est activée. Lorsque *iformat* vaut 0 et que la fin du fichier est atteinte, le pointeur de fichier est remis à zéro. Ceci redémarre la lecture depuis le début. Lorsque *iformat* vaut 1 ou 2, aucune boucle n'est active et à la fin du fichier, les variables correspondantes sont remplies avec des zéros.



### Note

Prière de noter que comme cet opcode génère sa sortie en utilisant des paramètres d'entrée (placés à droite de l'opcode), ces variables doivent avoir été initialisées avant leur utilisation, sinon une erreur "utilisé avant d'être défini" se produira. On peut utiliser l'opcode *init* pour cela.

## Exemples

Voici un exemple de l'opcode *fini*. Il utilise le fichier *fini.csd* [examples/fini.csd].

### Exemple 287. Exemple de l'opcode *fini*.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.



Noter que cet exemple nécessite la création du fichier test.txt, par exemple par *fouti*

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fini.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gihand fiopen "test.txt", 1

instr 1

  ihz  init 0
        fini gihand, 0, 1, ihz
  ar   oscil 0.5, ihz
        outs ar, ar
endin
</CsInstruments>
<CsScore>

i 1 0 1

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*fin, fink*

## Crédits

Auteur : Gabriel Maldonado  
 Italie  
 1999

Nouveau dans la version 3.56 de Csound

# fink

fink — Lit des signaux depuis un fichier au taux-k.

## Description

Lit des signaux depuis un fichier au taux-k.

## Syntaxe

```
fink ifilename, iskipframes, iformat, kin1 [, kin2] [, kin3] [...]
```

## Initialisation

*ifilename* -- nom du fichier d'entrée (peut être une chaîne de caractères ou un identificateur numérique généré par *fiopen*).

*iskipframes* -- nombre de trames à ignorer au début (chaque trame contient un échantillon de chaque canal).

*iformat* -- un nombre spécifiant le format du fichier d'entrée. Si un en-tête est trouvé, cet argument est ignoré.

- 0 - flottants sur 32 bit sans en-tête
- 1 - entiers sur 16 bit sans en-tête

## Exécution

*fink* est semblable à *fin* mais il opère au taux-k.



### Note

Prière de noter que comme cet opcode génère sa sortie en utilisant des paramètres d'entrée (placés à droite de l'opcode), ces variables doivent avoir été initialisées avant leur utilisation, sinon une erreur "utilisé avant d'être défini" se produira. On peut utiliser l'opcode *init* pour cela.

## Exemples

Voici un exemple de l'opcode *fink*. Il utilise le fichier *fink.csd* [examples/fink.csd].

### Exemple 288. Exemple de l'opcode *fink*.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

Noter que cet exemple nécessite la création du fichier test.txt, par exemple par *fouti*

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform
```

```

-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fink.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gihand fiopen "test.txt", 1

instr 1

  khz  init 0
       fink gihand, 0, 1, khz
  ar   oscil 0.5, khz
       outs  ar, ar
endin
</CsInstruments>
<CsScore>

i 1 0 1

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*fin, fini*

## Crédits

Auteur : Gabriel Maldonado  
 Italie  
 1999

Nouveau dans la version 3.56 de Csound

# fiopen

*fiopen* — Ouvre un fichier dans un mode spécifique.

## Description

On peut utiliser *fiopen* pour ouvrir un fichier dans un des modes spécifiés.

## Syntaxe

```
ihandle fiopen ifilename, imode
```

## Initialisation

*ihandle* -- un nombre qui spécifie ce fichier.

*ifilename* -- le nom du fichier concerné (entre guillemets).

*imode* -- choix du mode d'ouverture du fichier. *imode* peut prendre une des valeurs suivantes :

- 0 - ouvre un fichier texte en écriture
- 1 - ouvre un fichier texte en lecture
- 2 - ouvre un binaire texte en écriture
- 3 - ouvre un fichier binaire en lecture

## Exécution

*fiopen* ouvre un fichier à utiliser par la famille d'opcodes *fout*. Il est plus sûr de l'utiliser dans la section d'en-tête, en dehors de tout instrument. Il retourne un nombre, *ihandle*, qui fait référence de manière univoque au fichier ouvert.

Si *fiopen* est appelé sur un fichier déjà ouvert, il retourne simplement le même identificateur, et ne ferme pas le fichier.

Noter que *fout* et *foutk* peuvent utiliser soit un nom de chemin de fichier, soit un identificateur numérique généré par *fiopen*. Alors qu'avec *fouti* et *foutir*, le fichier cible ne peut être spécifié que par un identificateur numérique.

## Exemples

Voici un exemple de l'opcode *fiopen*. Il utilise le fichier *fiopen.csd* [exemples/*fiopen.csd*]. Noter que l'exemple doit être exécuté depuis un répertoire dans lequel l'écriture est autorisée.

### Exemple 289. Exemple de l'opcode *fiopen*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fiopen.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gihand fiopen "test1.txt", 0

instr 1

ires random 0, 100
fouti gihand, 0, 1, ires
ficlose gihand

endin
</CsInstruments>
<CsScore>

i 1 0 1

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*ficlose fout, fouti, foutir, foutk*

## Crédits

Auteur : Gabriel Maldonado  
 Italie  
 1999

Nouveau dans la version 3.56 de Csound

# flanger

flanger — Un flanger contrôlé par l'utilisateur.

## Description

Un flanger contrôlé par l'utilisateur.

## Syntaxe

```
ares flanger asig, adel, kfeedback [, imaxd]
```

## Initialisation

*imaxd*(facultatif) -- délai maximum en secondes (nécessaire pour l'allocation initiale de mémoire)

## Exécution

*asig* -- signal d'entrée

*adel* -- délai en secondes

*kfeedback* -- taux de rétroaction (en utilisation normale, ne doit pas dépasser 1, même si des valeurs supérieures sont permises)

Cette unité est utile pour générer des chorus et des flangers. Le retard doit être varié au taux-a, par exemple en connectant *adel* à la sortie d'un oscillateur. La rétroaction peut varier au taux-k. Cet opcode est implémenté de façon à ce que *kr* puisse être différent de *sr* (sinon le délai ne pourrait pas être inférieur à *ksmps*) ce qui facilite l'exécution en temps réel. Cette unité ressemble beaucoup à *wguide1*, la seule différence étant que *flanger* n'a pas de filtre passe-bas ou le fait qu'on ne peut varier le retard qu'au taux-a.

## Exemples

Voici un exemple de l'opcode flanger. Il utilise le fichier *flanger.csd* [examples/flanger.csd].

### Exemple 290. Exemple de l'opcode flanger.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o flanger.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
```

```

Odbfs = 1

instr 1

kfeedback = p4
asnd vco2 .2, 50
adel linseg 0, p3*.5, 0.02, p3*.5, 0 ;max delay time =20ms
aflg flanger asnd, adel, kfeedback
asig clip aflg, 1, 1
      outs asig+asnd, asig+asnd ;mix flanger with original

endin
</CsInstruments>
<CsScore>

i 1 0 10 .2
i 1 11 10 .8 ;lot of feedback
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

Plus d'information sur le flanger sur Wikipedia : <http://en.wikipedia.org/wiki/Flanger>

## Crédits

Auteur : Gabriel Maldonado  
Italie

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.49 de Csound

# flashtxt

flashtxt — Permet d'afficher du text depuis des instruments sous la forme de curseurs.

## Description

Opcode du greffon control.

Permet d'afficher du text depuis des instruments sous la forme de curseurs, etc. (Actuellement, ne fonctionne que sous Unix et Windows).

## Syntaxe

```
flashtxt iwhich, String
```

## Initialisation

*iwhich* -- le numéro de la fenêtre.

*String* -- la chaîne de caractères à afficher.

## Exécution

Noter que cet opcode n'est pas disponible sous Windows à cause de l'implémentation des tuyaux sur ce système.

Une fenêtre est créée, identifiée par l'argument *iwhich*, avec la chaîne de caractères affichée. Si le texte est remplacé par un nombre, la fenêtre est effacée. Noter que les fenêtres de texte sont numérotées globalement si bien que différents instruments peuvent changer le texte, et que la fenêtre survit à l'instance de l'instrument.

## Exemples

Voici un exemple de l'opcode flashtxt. Il utilise le fichier *flashtxt.csd* [examples/flashtxt.csd].

### Exemple 291. Exemple de l'opcode flashtxt.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o flashtxt.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
```



```
ksmps = 1
nchnls = 1

instr 1
  flashtxt 1, "Instr 1 live"
  ao oscil 4000, 440, 1
  out ao
endin

</CsInstruments>
<CsScore>

; Table 1: an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 4.11 de Csound

# FLbox

FLbox — Un widget FLTK qui affiche du texte dans une boîte.

## Description

Un widget FLTK qui affiche du texte dans une boîte.

## Syntaxe

```
ihandle FLbox "label", itype, ifont, isize, iwidth, iheight, ix, iy [, image]
```

```
ihandle FLbox istr, itype, ifont, isize, iwidth, iheight, ix, iy [, image]
```

## Initialisation

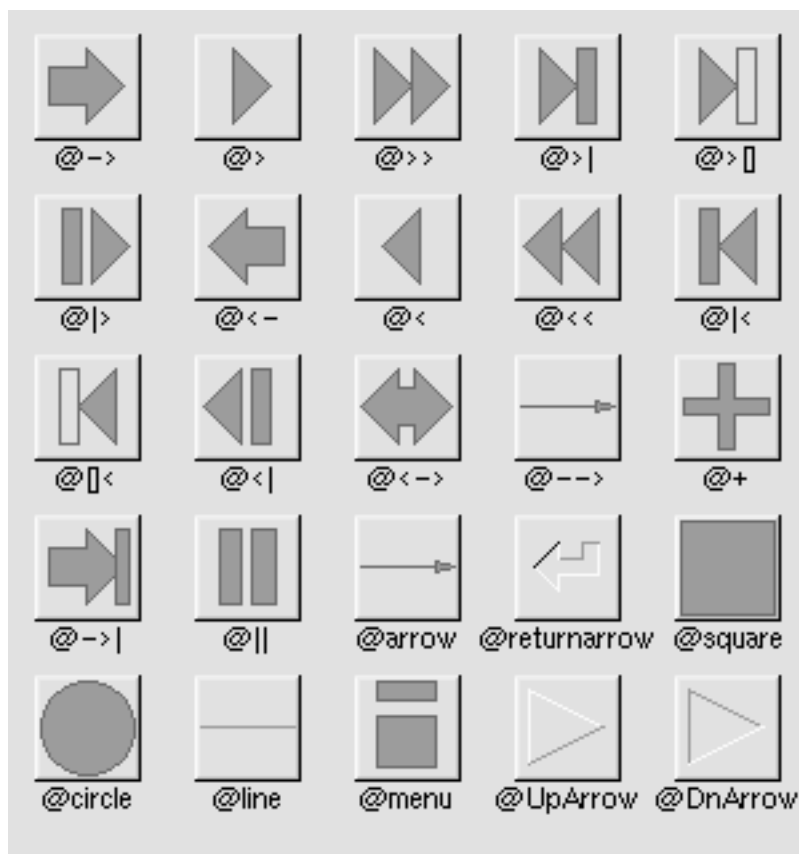
*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*). Il est automatiquement retourné par *FLbox* et ne doit pas être fixé par l'étiquette de l'utilisateur. (L'étiquette de l'utilisateur est une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.)

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*istr* -- une valeur de taux-i utilisée pour obtenir une chaîne de caractères via *strset*, placée près du widget correspondant.

Noter qu'avec *FLbox* il n'est pas nécessaire d'appeler l'opcode *FLsetTextType* pour utiliser un symbole. Dans ce cas, il suffit d'utiliser une étiquette commençant par « @ » suivi de la chaîne de formatage correcte.

Les symboles suivants sont supportés :



Symboles d'étiquette FLTK supportés.

Le signe @ peut être suivi par les caractères de « formatage » facultatifs suivants, dans cet ordre :

1. « # » force une image carrée sans distortion de la forme du widget.
2. +[1-9] ou -[1-9] grossit ou diminue l'image.
3. [1-9] effectue une rotation d'un multiple de 45 degrés. « 6 » ne fait rien, les autres valeurs pointent dans la direction de cette touche sur un pavé numérique.

*itype* -- un nombre entier dénotant l'apparence du widget. Les valeurs suivantes sont acceptées :

- 1 - boîte sans relief
- 2 - boîte saillante
- 3 - boîte en creux
- 4 - boîte légèrement saillante
- 5 - boîte légèrement en creux
- 6 - boîte gravée
- 7 - boîte en relief
- 8 - boîte avec cadre

- 9 - boîte ombrée
- 10 - boîte arrondie
- 11 - boîte arrondie ombrée
- 12 - boîte arrondie sans relief
- 13 - boîte arrondie saillante
- 14 - boîte arrondie creuse
- 15 - boîte en losange saillante
- 16 - boîte en losange en creux
- 17 - boîte ovale
- 18 - boîte ovale ombrée
- 19 - boîte ovale sans relief

*ifont* -- un nombre entier dénotant le type de la police de *FLbox*. Les valeurs suivantes sont acceptées :

- 1 - helvetica (comme "Arial" sous Windows)
- 2 - helvetica gras
- 3 - helvetica italique
- 4 - helvetica gras italique
- 5 - courrier
- 6 - courrier gras
- 7 - courrier italique
- 8 - courrier gras italique
- 9 - times
- 10 - times gras
- 11 - times italique
- 12 - times gras italique
- 13 - symbol
- 14 - screen
- 15 - screen gras
- 16 - dingbats

*isize* -- taille de la police.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*image* -- un identifiant faisant référence à une image éventuellement ouverte avec l'opcode *bmopen*. S'il est utilisé, cela permet un skin pour ce widget.



### Note sur l'opcode *bmopen*

Bien qu'il soit mentionné, l'opcode *bmopen* n'a pas été implémenté dans Csound 4.22.

## Exécution

*FLbox* est utile pour montrer du texte dans une fenêtre. Le texte est à l'intérieur d'une boîte dont l'aspect dépend de l'argument *itype*.

Noter que *FLbox* n'est pas un valuateur et que sa valeur est constante. Elle ne peut pas être modifiée.

## Exemples

Voici un exemple de l'opcode *FLbox*. Il utilise le fichier *FLbox.csd* [examples/FLbox.csd].

### Exemple 292. Exemple de l'opcode *FLbox*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLbox.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Text Box", 700, 400, 50, 50
; Box border type (7=embossed box)
itype = 7
; Font type (10='Times Bold')
ifont = 10
; Font size
isize = 20
; Width of the flbox
iwidth = 400
; Height of the flbox
iheight = 30
; Distance of the left edge of the flbox
; from the left edge of the panel
```

```

ix = 150
; Distance of the upper edge of the flbox
; from the upper edge of the panel
iy = 100

ih3 FLbox "Use Text Boxes For Labelling", itype, ifont, isize, iwidth, iheight, ix, iy
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
endin

</CsInstruments>
<CsScore>

; Real-time performance for 1 hour.
f 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*FLbutBank, FLbutton, FLprintk, FLprintk2, FLvalue*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLbutBank

FLbutBank — Un opcode de widget FLTK qui crée un banc de boutons.

## Description

Un opcode de widget FLTK qui crée un banc de boutons.

## Syntaxe

```
kout, ihandle FLbutBank itype, inumx, inumy, iwidth, iheight, ix, iy, \  
iopcode [, kp1] [, kp2] [, kp3] [, kp4] [, kp5] [....] [, kpN]
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*). Il est automatiquement retourné par *FLbutBank* et ne doit pas être fixé par l'étiquette de l'utilisateur. (L'étiquette de l'utilisateur est une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.)

*itype* -- un nombre entier dénotant l'apparence du widget. Les nombres acceptés sont :

- 1 - bouton normal
- 2 - bouton lumineux
- 3 - bouton à cocher
- 4 - bouton avec un cercle à cocher

On peut ajouter 20 à la valeur pour créer un bouton de type "plastique". Noter qu'il n'y a pas de bouton arrondi plastique (si vous fixez le type à 24, le bouton aura la même apparence qu'avec le type 23).

*inumx* -- nombre de boutons dans chaque rangée du banc.

*inumy* -- nombre de boutons dans chaque colonne du banc.

*ix* -- position horizontale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante, exprimée en pixels.

*iy* -- position verticale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante, exprimée en pixels.

*iopcode* -- type de l'instruction de partition. Il faut fournir le code ASCII de la lettre correspondant à l'instruction de partition. Actuellement seules les instructions de partition « i » (code ASCII 105) sont supportées. Une valeur de zéro fait référence à une valeur de « i » par défaut. Ainsi 0 et 105 activent l'instruction *i*. Une valeur de -1 désactive cette possibilité.

## Exécution

*kout* -- valeur de sortie.

*kp1*, *kp2*, ..., *kpN* -- arguments des instruments activés.

L'opcode *FLbutBank* crée un banc de boutons. Par exemple, la ligne suivante :

```
gkButton,ihb1 FLbutBank 22, 8, 8, 380, 180, 50, 350, 0, 7, 0, 0, 5000, 6000
```

créera ce banc :



*FLbutBank*.

Un clic sur un bouton coche celui-ci. Il peut aussi décocher un bouton préalablement coché appartenant au même banc. Ces boutons se comportent donc toujours comme des boutons radio. Noter que chaque bouton est étiqueté avec un nombre croissant. L'argument *kout* reçoit ce nombre lorsque le bouton correspondant est coché.

Non seulement *FLbutBank* retourne une valeur, mais il peut aussi activer (ou programmer) un instrument fourni par l'utilisateur chaque fois qu'un bouton est cliqué. Si l'argument *iopcode* est fixé à un nombre négatif, aucun instrument n'est activé ; ainsi cette possibilité est facultative. Pour activer un instrument, *iopcode* doit valoir 0 ou 105 (le code ASCII du caractère « i », faisant référence à l'instruction de partition *i*). Les p-champs de l'instrument activé sont *kp1* (numéro de l'instrument), *kp2* (date de l'action), *kp3* (durée), suivis des autres p-champs.

L'argument *itype* fixe le type des boutons de la même manière que pour l'opcode *FLbutton*. En ajoutant 10 à l'argument *itype* (c'est à dire en lui attribuant la valeur 11 pour le type 1, 12 pour le type 2, 13 pour le type 3 et 14 pour le type 4), il est possible d'ignorer la valeur courante de *FLbutBank* lorsque l'on sauve ou que l'on récupère des instantanés (voir *Opcodes Généraux relatifs aux Widgets FLTK*). On peut aussi ajouter 10 aux type de boutons "plastiques" (31 pour le type 1, 32 pour le type 2, etc).

*FLbutBank* est très utile pour retrouver des instantanés.

## Exemples

Voici un exemple de l'opcode *FLbutBank*. Il utilise le fichier *FLbutBank.csd* [exemples/*FLbutBank.csd*].

### Exemple 293. Exemple de l'opcode *FLbutBank*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d          ;;RT audio I/O
```



```

; For Non-realtime ouput leave only the line below:
; -o FLbutton.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
nchnls = 1

FLpanel "Button Bank", 520, 140, 100, 100
;itype = 2 ;Light Buttons
itype = 22 ;Plastic Light Buttons
inumx = 10
inumy = 4
iwidth = 500
iheight = 120
ix = 10
iy = 10
iopcode = 0
istarttim = 0
idur = 1

gkbutton, ihbb FLbutBank itype, inumx, inumy, iwidth, iheight, ix, iy, iopcode, 1, istarttim, idur

FLpanelEnd
FLrun

instr 1
ibutton = i(gkbutton)
prints "Button %i pushed!\\n", ibutton
endin

</CsInstruments>
<CsScore>

; Real-time performance for 1 hour.
f 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*FLbox, FLbutton, FLprintk, FLprintk2, FLvalue*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLbutton

FLbutton — Un opcode de widget FLTK qui crée un bouton.

## Description

Un opcode de widget FLTK qui crée un bouton.

## Syntaxe

```
kout, ihandle FLbutton "label", ion, ioff, itype, iwidth, iheight, ix, \
    iy, iopcode [, kp1] [, kp2] [, kp3] [, kp4] [, kp5] [....] [, kpN]
```

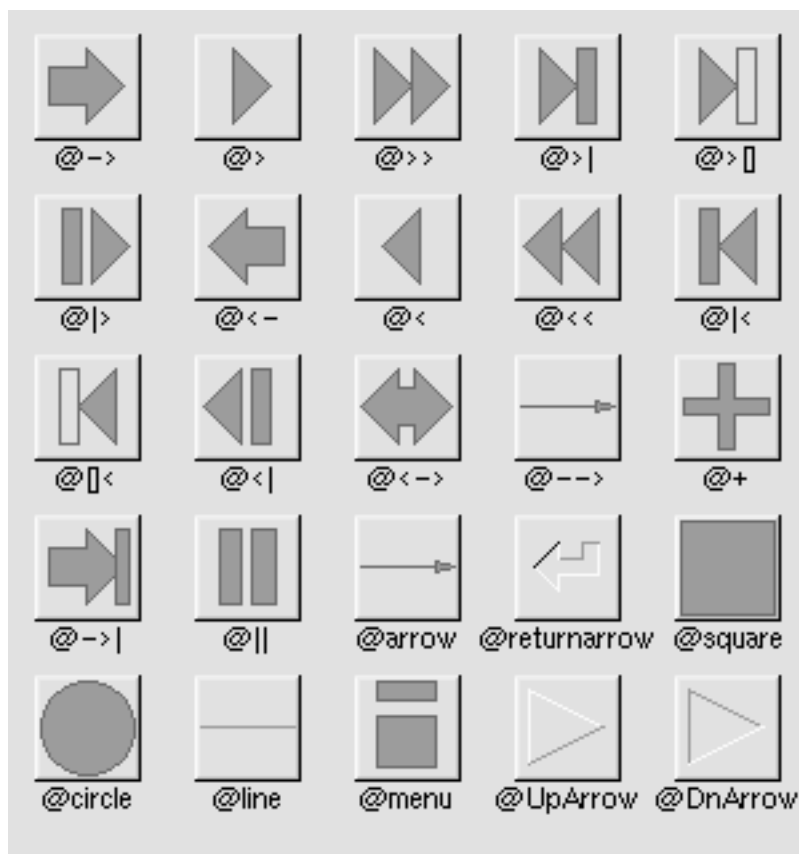
## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*). Il est automatiquement retourné par *FLbutton* et ne doit pas être fixé par l'étiquette de l'utilisateur. (L'étiquette de l'utilisateur est une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.)

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

Noter qu'avec *FLbutton* il n'est pas nécessaire d'appeler l'opcode *FLsetTextType* pour utiliser un symbole. Dans ce cas, il suffit d'utiliser une étiquette commençant par « @ » suivi de la chaîne de formatage correcte.

Les symboles suivants sont supportés :



Symboles d'étiquette FLTK supportés.

Le signe @ peut être suivi par les caractères de « formatage » facultatifs suivants, dans cet ordre :

1. « # » force une image carrée sans distortion de la forme du widget.
2. +[1-9] or -[1-9] grossit ou diminue l'image.
3. [1-9] effectue une rotation d'un multiple de 45 degrés. « 6 » ne fait rien, les autres valeurs pointent dans la direction de cette touche sur un pavé numérique.

*ion* -- valeur retournée quand le bouton est coché.

*ioff* -- valeur retournée quand le bouton est décoché.

*itype* -- un nombre entier dénotant l'apparence du widget.

Plusieurs types de bouton sont possibles selon la valeur de l'argument *itype* :

- 1 - bouton normal
- 2 - bouton lumineux
- 3 - bouton à cocher
- 4 - bouton avec un cercle à cocher

On peut ajouter 20 à la valeur pour créer un bouton de type "plastique". Noter qu'il n'y a pas de bouton arrondi plastique (si vous fixer le type à 24, le bouton aura la même apparence qu'avec le type 23).

Voici l'apparence des boutons :



FLbutton.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iopcode* -- type de l'instruction de partition. Il faut fournir le code ASCII de la lettre correspondant à l'instruction de partition. Actuellement seules les instructions de partition « i » (code ASCII 105) sont supportées. Une valeur de zéro fait référence à une valeur de « i » par défaut. Ainsi 0 et 105 activent l'instruction *i*. Une valeur de -1 désactive cette possibilité.

## Exécution

*kout* -- valeur de sortie.

*kp1*, *kp2*, ..., *kpN* -- arguments des instruments activés.

Les boutons de type 2, 3 et 4 retournent (argument *kout*) la valeur contenue dans l'argument *ion* lorsqu'ils sont cochés et celle contenue dans l'argument *ioff* lorsqu'ils sont décochés.

En ajoutant 10 à l'argument *itype* (c'est à dire en lui attribuant la valeur 11 pour le type 1, 12 pour le type 2, 13 pour le type 3 et 14 pour le type 4), il est possible d'ignorer la valeur du bouton lorsque l'on sauve ou que l'on récupère des instantanés (voir la section suivante). Non seulement *FLbutton* retourne une valeur, mais il peut aussi activer (ou programmer) un instrument fourni par l'utilisateur chaque fois le bouton est cliqué. On peut aussi ajouter 10 aux type de boutons "plastiques" (31 pour le type 1, 32 pour le type 2, etc).

Si l'argument *iopcode* est fixé à un nombre négatif, aucun instrument n'est activé. Ainsi cette possibilité est facultative. Pour activer un instrument, *iopcode* doit valoir 0 ou 105 (le code ASCII du caractère « i », faisant référence à l'instruction de partition *i*).

Les p-champs de l'instrument activé sont *kp1* (numéro de l'instrument), *kp2* (date de l'action), *kp3* (durée), suivis des p-champs de l'utilisateur. Noter que pour les boutons à deux états (bouton lumineux, bouton à cocher, bouton avec un cercle à cocher), l'instrument n'est activé que lorsque le l'état du bouton passe de décoché à coché (pas de coché à décoché).

## Exemples

Voici un exemple de l'opcode FLbutton. Il utilise les fichiers *FLbutton.csd* [examples/FLbutton.csd] et *beats.wav* [examples/beats.wav].

### Exemple 294. Exemple de l'opcode FLbutton.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLbutton.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Using FLbuttons to create on screen controls for play,
; stop, fast forward and fast rewind of a sound file
; This example also makes use of a preset graphic for buttons.

sr = 44100
kr = 44100
ksmps = 1
nchnls = 2

FLpanel "Buttons", 240, 400, 100, 100
  ion = 0
  ioff = 0
  itype = 1
  iwidth = 50
  iheight = 50
  ix = 10
  iy = 10
  iopcode = 0
  istarttim = 0
  idur = -1 ;Turn instruments on indefinitely

; Normal speed forwards
gkplay, ihb1 FLbutton "@>", ion, ioff, itype, iwidth, iheight, ix, iy, iopcode, 1, istarttim, idur,
; Stationary
gkstop, ihb2 FLbutton "@square", ion, ioff, itype, iwidth, iheight, ix+55, iy, iopcode, 2, istarttim, idur,
; Double speed backwards
gkrew, ihb3 FLbutton "@<<", ion, ioff, itype, iwidth, iheight, ix + 110, iy, iopcode, 1, istarttim, idur,
; Double speed forward
gkff, ihb4 FLbutton "@>>", ion, ioff, itype, iwidth, iheight, ix+165, iy, iopcode, 1, istarttim, idur,
; Type 1
gkt1, iht1 FLbutton "1-Normal Button", ion, ioff, 1, 200, 40, ix, iy + 65, -1
; Type 2
gkt2, iht2 FLbutton "2-Light Button", ion, ioff, 2, 200, 40, ix, iy + 110, -1
```

```

; Type 3
gkt3, iht3 FLbutton "3-Check Button", ion, ioff, 3, 200, 40, ix, iy + 155, -1
; Type 4
gkt4, iht4 FLbutton "4-Round Button", ion, ioff, 4, 200, 40, ix, iy + 200, -1
; Type 21
gkt5, iht5 FLbutton "21-Plastic Button", ion, ioff, 21, 200, 40, ix, iy + 245, -1
; Type 22
gkt6, iht6 FLbutton "22-Plastic Light Button", ion, ioff, 22, 200, 40, ix, iy + 290, -1
; Type 23
gkt7, iht7 FLbutton "23-Plastic Check Button", ion, ioff, 23, 200, 40, ix, iy + 335, -1
FLpanelEnd
FLrun

; Ensure that only 1 instance of instr 1
; plays even if the play button is clicked repeatedly
insnum = 1
icount = 1
maxalloc insnum, icount

instr 1
  asig disk2 "beats.wav", p4, 0, 1
  outs asig, asig
endin

instr 2
  turnoff2 1, 0, 0 ;Turn off instr 1
  turnoff ;Turn off this instrument
endin

</CsInstruments>
<CsScore>

; Real-time performance for 1 hour.
f 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*FLbox, FLbutBank, FLprintk, FLprintk2, FLvalue*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLcloseButton

FLcloseButton — Un opcode de widget FLTK qui crée un bouton qui fermera la fenêtre du panneau auquel il appartient.

## Description

Un opcode de widget FLTK qui crée un bouton qui fermera la fenêtre du panneau auquel il appartient.

## Syntaxe

```
ihandle FLcloseButton "label", iwidth, iheight, ix, iy
```

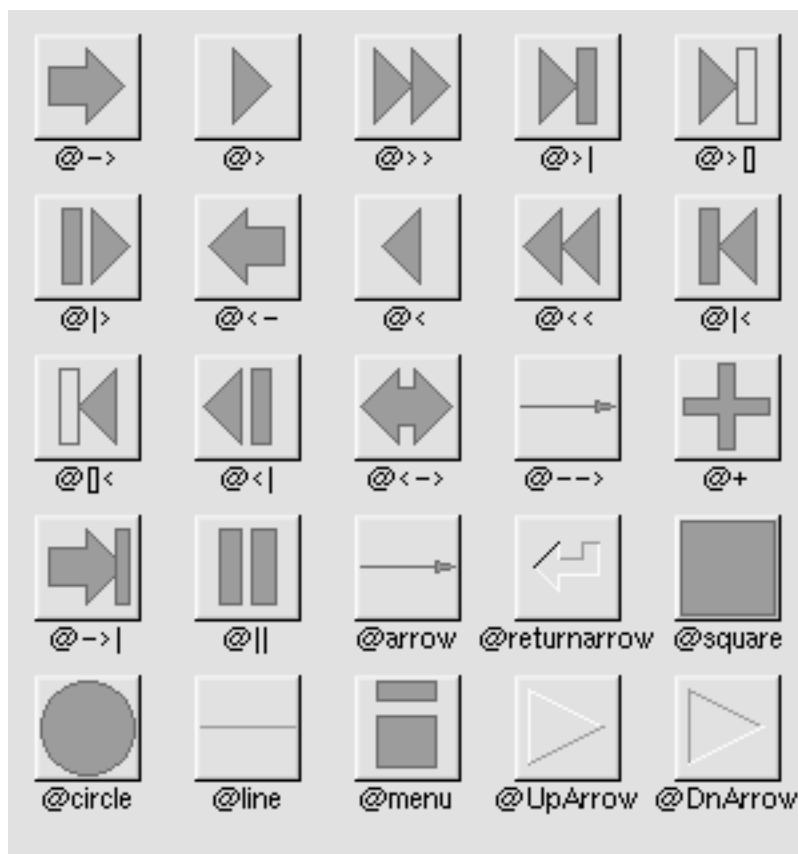
## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*). Il est automatiquement retourné par *FLcloseButton* et ne doit pas être fixé par l'étiquette de l'utilisateur. (L'étiquette de l'utilisateur est une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.)

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

Noter qu'avec *FLcloseButton* il n'est pas nécessaire d'appeler l'opcode *FLsetTextType* pour utiliser un symbole. Dans ce cas, il suffit d'utiliser une étiquette commençant par « @ » suivi de la chaîne de formatage correcte.

Les symboles suivants sont supportés :



Symboles d'étiquette FLTK supportés.

Le signe @ peut être suivi par les caractères de « formatage » facultatifs suivants, dans cet ordre :

1. « # » force une image carrée sans distortion de la forme du widget.
2. +[1-9] or -[1-9] grossit ou diminue l'image.
3. [1-9] effectue une rotation d'un multiple de 45 degrés. « 6 » ne fait rien, les autres valeurs pointent dans la direction de cette touche sur un pavé numérique.

*ewidth* -- largeur du widget.

*height* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

## Voir aussi

*FLbutton*, *FLbox*, *FLbutBank*, *FLprintk*, *FLprintk2*, *FLvalue*

## Crédits

Auteur : Steven Yi



Nouveau dans la version 5.05

# FLcolor

FLcolor — Un opcode FLTK qui fixe les couleurs principales.

## Description

Fixe les couleurs principales à des valeurs RVB données par l'utilisateur.

## Syntaxe

```
FLcolor ired, igreen, iblue [, ired2, igreen2, iblue2]
```

## Initialisation

*ired* -- La composante rouge du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*igreen* -- La composante verte du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*iblue* -- La composante bleue du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*ired2* -- La composante rouge de la couleur secondaire du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*igreen2* -- La composante verte de la couleur secondaire du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*iblue2* -- La composante bleue de la couleur secondaire du widget cible. Chaque composante RVB est comprise entre 0 et 255.

## Exécution

Ces opcodes modifient l'apparence d'autres widgets. Ils sont de deux types : ceux ne contenant pas d'argument *ihandle* qui affectent tous les widgets déclarés après eux, et ceux ayant un argument *ihandle* qui n'affectent qu'un widget cible déclaré avant eux.

*FLcolor* fixe les couleurs principales à des valeurs RVB données par l'utilisateur. Cet opcode affecte la couleur principale de (presque) tous les widgets définis après lui. On peut placer plusieurs instances de *FLcolor* devant chaque widget que l'on veut modifier. Cependant, pour modifier un seul widget, il sera plus judicieux d'utiliser un opcode appartenant à la seconde catégorie (c'est-à-dire ceux contenant l'argument *ihandle*).

*FLcolor* est conçu pour modifier les couleurs d'un groupe de widgets en relation, supposés être de la même couleur. L'influence de *FLcolor* sur les widgets suivants peut-être désactivée en utilisant -1 comme seul argument de l'opcode. De plus, en utilisant -2 (ou -3) comme seul argument de *FLcolor*, tous les widgets suivants auront une couleur choisie aléatoirement. -2 sélectionne une couleur aléatoire claire, tandis que -3 sélectionne une couleur aléatoire foncée.

L'utilisation de *ired2*, *igreen2*, *iblue2* est équivalente à l'utilisation d'un *FLcolor2* séparé.

## Voir aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLcolor2

FLcolor2 — Un opcode FLTK qui fixe la couleur secondaire (de sélection).

## Description

*FLcolor2* est semblable à *FLcolor* sauf qu'il affecte la couleur secondaire (de sélection).

## Syntaxe

```
FLcolor2 ired, igreen, iblue
```

## Initialisation

*ired* -- La composante rouge du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*igreen* -- La composante verte du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*iblue* -- La composante bleue du widget cible. Chaque composante RVB est comprise entre 0 et 255.

## Exécution

Ces opcodes modifient l'apparence d'autres widgets. Ils sont de deux types : ceux ne contenant pas d'argument *ihandle* qui affectent tous les widgets déclarés après eux, et ceux ayant un argument *ihandle* qui n'affectent qu'un widget cible déclaré avant eux.

*FLcolor2* est semblable à *FLcolor* sauf qu'il affecte la couleur secondaire (de sélection). L'influence de *FLcolor2* sur les widgets suivants peut-être désactivée en le fixant à -1. Avec un valeur de -2 (ou -3), tous les widgets suivants auront une couleur secondaire choisie aléatoirement. -2 sélectionne une couleur aléatoire claire, tandis que -3 sélectionne une couleur aléatoire foncée.

## Voir aussi

*FLcolor*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLcount

FLcount — Un opcode de widget FLTK qui crée un compteur.

## Description

Permet à l'utilisateur d'augmenter/diminuer une valeur avec des clics de souris sur un bouton fléché correspondant.

## Syntaxe

```
kout, ihandle FLcount "label", imin, imax, istep1, istep2, itype, \  
    iwidth, iheight, ix, iy, iopcode [, kp1] [, kp2] [, kp3] [...] [, kpN]
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du valuateur. Il est automatiquement fixé par le valuateur.

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*imin* -- valeur minimale de l'intervalle de sortie.

*imax* -- valeur maximale de l'intervalle de sortie.

*istep1* -- un nombre en virgule flottante indiquant le pas d'incréméntation du valuateur à chaque clic de souris. *istep1* sert aux réglages fins.

*istep2* -- un nombre en virgule flottante indiquant le pas d'incréméntation du valuateur à chaque clic de souris. *istep2* sert aux réglages grossiers.

*itype* -- un nombre entier dénotant l'apparence du valuateur.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

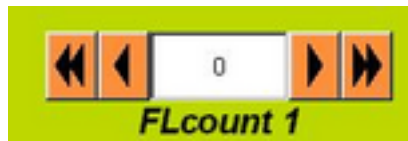
*iopcode* -- type de l'instruction de partition. Il faut fournir le code ASCII de la lettre correspondant à l'instruction de partition. Actuellement seules les instructions de partition « i » (code ASCII 105) sont supportées. Une valeur de zéro fait référence à une valeur de « i » par défaut. Ainsi 0 et 105 activent l'instruction *i*. Une valeur de -1 désactive cette possibilité.

## Exécution

*kout* -- valeur de sortie.

*kp1*, *kp2*, ..., *kpN* -- arguments des instruments activés.

*FLcount* permet à l'utilisateur d'augmenter/diminuer une valeur avec des clics de souris sur les boutons fléchés correspondants :



*FLcount*.

Il y a deux sortes de boutons fléchés, pour des pas plus grands ou plus petits. Noter que non seulement *FLcount* retourne une valeur et un identifiant, mais il peut aussi activer (programmer) un instrument fourni par l'utilisateur chaque fois qu'un bouton est cliqué. Les p-champs de l'instrument activé sont *kp1* (numéro de l'instrument), *kp2* (date de l'action), *kp3* (durée) suivis des p-champs de l'utilisateur. Si l'argument *iopcode* est fixé à un nombre négatif, aucun instrument n'est activé. Ainsi cette possibilité est facultative.

## Exemples

Voici un exemple de l'opcode *FLcount*. Il utilise le fichier *FLcount.csd* [exemples/*FLcount.csd*].

### Exemple 295. Exemple de l'opcode *FLcount*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLcount.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Demonstration of the flcount opcode
; clicking on the single arrow buttons
; increments the oscillator in semitone steps
; clicking on the double arrow buttons
; increments the oscillator in octave steps
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Counter", 900, 400, 50, 50
; Minimum value output by counter
imin = 6
; Maximum value output by counter
imax = 12
; Single arrow step size (semitones)
istep1 = 1/12
; Double arrow step size (octave)
istep2 = 1
; Counter type (1=double arrow counter)
itype = 1
; Width of the counter in pixels
iwidth = 200
; Height of the counter in pixels
```

```

    iheight = 30
    ; Distance of the left edge of the counter
    ; from the left edge of the panel
    ix = 50
    ; Distance of the top edge of the counter
    ; from the top edge of the panel
    iy = 50
    ; Score event type (-1=ignored)
    iopcode = -1

    gkoct, ihandle FLcount "pitch in oct format", imin, imax, istep1, istep2, itype, iwidth, iheight, i
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
    iamp = 15000
    ifn = 1
    asig oscili iamp, cpsoct(gkoct), ifn
    out asig
endin

</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*FLjoy, FLknob, FLroller, FLslider, FLtext*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLexecButton

FLexecButton — Un opcode de widget FLTK qui crée un bouton qui exécute une commande.

## Description

Un opcode de widget FLTK qui crée un bouton qui exécute une commande. Utile pour ouvrir une documentation HTML comme un texte Au sujet de ou pour démarrer un programme séparé depuis une interface de widgets FLTK.



### Avertissement

Comme n'importe quelle commande peut être exécutée, il faut être très prudent en utilisant cet opcode et en exécutant des orchestres d'autres personnes utilisant cet opcode.

## Syntaxe

```
ihandle FLexecButton "command", iwidth, iheight, ix, iy
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*). Il est automatiquement retourné par *FLexecButton*.

« *command* » -- une chaîne de caractères entre guillemets contenant la commande à exécuter.

Noter qu'avec *FLexecButton*, le texte par défaut du bouton est "About" et qu'il est nécessaire d'appeler l'opcode *FLsetText* pour changer le texte du bouton.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

## Exemples

Voici un exemple de l'opcode *FLexecButton*. Il utilise le fichier *FLexecButton.csd* [exemples/FLexecButton.csd].

### Exemple 296. Exemple de l'opcode FLexecButton.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.



```

<CsoundSynthesizer>

<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No display
-odac      -iadc      -d      ;;;RT audio I/O
</CsOptions>

<CsInstruments>

    sr      = 44100
    ksmps   = 10
    nchnls  = 1

; Example by Jonathan Murphy 2007

;;; reset amplitude range
    odbfs   = 1

;;; set the base colour for the panel
    FLcolor 100, 0, 200
;;; define the panel
    FLpanel "FLexecButton", 250, 100, 0, 0
;;; sliders to control time stretch and pitch
    gkstr, gistretch FLslider "Time", 0.5, 1.5, 0, 6, -1, 10, 60, 150, 20
    gkpch, gipitch   FLslider "Pitch", 0.5, 1.5, 0, 6, -1, 10, 60, 200, 20
;;; set FLexecButton colour
    FLcolor 255, 255, 0
;;; when this button is pressed, fourier analysis is performed on the file
;;; "beats.wav", producing the analysis file "beats.pvx"
    gipvoc FLexecButton "csound -U pvanal beats.wav beats.pvx", 60, 20, 20, 20
;;; set FLexecButton text
    FLsetText "PVOC", gipvoc
;;; when this button is pressed, instr 10000 is called, exiting
;;; Csound immediately

;;; cancel previous colour
    FLcolor -1
;;; set colour for kill button
    FLcolor 255, 0, 0
    gkkill, gikill FLbutton "X", 1, 1, 1, 20, 20, 100, 20, 0, 10000, 0, 0.1
;;; cancel previous colour
    FLcolor -1
;;; set colour for play/stop and pause buttons
    FLcolor 0, 200, 0
;;; pause and play/stop buttons
    gkpause, gipause FLbutton "@| |", 1, 0, 2, 40, 20, 20, 60, -1
    gkplay, gipplay  FLbutton "@|>", 1, 0, 2, 40, 20, 80, 60, -1
;;; end the panel
    FLpanelEnd
;;; set initial values for time stretch and pitch
    FLsetVal_i 1, gistretch
    FLsetVal_i 1, gipitch
;;; run the panel
    FLrun

    instr 1 ; trigger play/stop
    ;;; is the play/stop button on or off?
    ;;; either way we need to trigger something,
    ;;; so we can't just use the value of gkplay
    kon trigger gkplay, 0, 0
    koff trigger gkplay, 1, 1
    ;;; if on, start instr 2
    schedkwhen kon, -1, -1, 2, 0, -1

```

```

    ;;; if off, stop instr 2
    schedkwhen koff, -1, -1, -2, 0, -1

    endin

    instr 2

    ;;; paused or playing?
    if (gkpause == 1) kgoto pause
    kgoto start
pause:
    ;;; if the pause button is on, skip sound production
    kgoto end
start:
    ;;; get the length of the analysis file in seconds
    ilen filelen "beats.pvx"
    ;;; determine base frequency of playback
    icps = 1/ilen
    ;;; create a table over the length of the file
    itpt ftgen 0, 0, 513, -7, 0, 512, ilen
    ;;; phasor for time control
    kphs phasor icps * gkstr
    ;;; use phasor as index into table
    kndx = kphs * 512
    ;;; read table
    ktpt tablei kndx, itpt
    ;;; use value from table as time pointer into file
    fsig1 pvsfread ktpt, "beats.pvx"
    ;;; change playback pitch
    fsig2 pvscale fsig1, gkpch
    ;;; resynthesize
    aout pvsynth fsig2
    ;;; envelope to avoid clicks and clipping
    aenv linsegr 0, 0.3, 0.75, 0.1, 0
    aout = aout * aenv
    out aout
end:

    endin

    instr 10000 ; kill

    exitnow

    endin

</CsInstruments>

<CsScore>
i1 0 10000
e
</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

*FLbutton, FLbox, FLbutBank, FLprintk, FLprintk2, FLvalue*

## Crédits

Auteur : Steven Yi

Exemple par Jonathan Murphy

Nouveau dans la version 5.05

# FLgetsnap

FLgetsnap — Retrouve un instantané FLTK antérieurement enregistré.

## Description

Retrouve un instantané FLTK antérieurement enregistré (en mémoire), c'est-à-dire fixe tous les valueurs aux valeurs correspondantes enregistrées dans l'instantané.

## Syntaxe

```
inumsnap FLgetsnap index [, igroup]
```

## Initialisation

*inumsnap* -- nombre courant d'instantanés.

*index* -- un nombre faisant référence de manière univoque à un instantané. Plusieurs instantanés peuvent être enregistrés dans la même banque.

*igroup* -- (facultatif) un nombre entier faisant référence à un groupe de widgets en relation avec un instantané. Cela permet de lire/écrire, ou charger/sauvegarder l'état d'un sous-ensemble de valueurs. La valeur par défaut est zéro qui fait référence au premier groupe. Le numéro de groupe est déterminé par l'opcode *FLsetSnapGroup*.



### Note

Le paramètre *igroup* n'a pas encore été complètement implémenté dans la version actuelle de Csound. Prière de ne pas s'y fier.

## Exécution

*FLgetsnap* retrouve un instantané FLTK antérieurement enregistré (en mémoire), c'est-à-dire fixe tous les valueurs aux valeurs correspondantes enregistrées dans l'instantané. L'argument *index* doit faire référence de manière univoque à un instantané existant. Si l'argument *index* fait référence à un instantané vide ou à un instantané qui n'existe pas, rien ne se produit. *FLsetsnap* retourne le nombre courant d'instantanés (argument *inumsnap*).

Pour économiser la mémoire, les widgets peuvent être groupés afin que les instantanés n'affectent qu'un groupe défini de widgets. L'opcode *FLsetSnapGroup* est utilisé pour spécifier le groupe de tous les widgets déclarés après lui jusqu'à la prochaine instruction *FLsetSnapGroup*.

## Voir aussi

*FLloadsnap*, *FLrun*, *FLsavesnap*, *FLsetsnap*, *FLsetSnapGroup*, *FLupdate*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLgroup

FLgroup — Un opcode de conteneur FLTK qui regroupe des widgets enfants.

## Description

Un opcode de conteneur FLTK qui regroupe des widgets enfants.

## Syntaxe

```
FLgroup "label", iwidth, iheight, ix, iy [, iborder] [, image]
```

## Initialisation

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iborder* (facultatif, 0 par défaut) -- type de la bordure du conteneur. Il est exprimé par un nombre entier choisi parmi les suivants :

- 0 - pas de bordure
- 1 - bordure de boîte en creux
- 2 - bordure de boîte saillante
- 3 - bordure gravée
- 4 - bordure en relief
- 5 - bordure ligne noire
- 6 - mince bordure en creux
- 7 - mince bordure saillante

Si le nombre entier ne correspond à aucune de ces valeurs, aucune bordure n'est fournie par défaut.

*image* (facultatif) -- un identifiant faisant référence à une image éventuellement ouverte avec l'opcode *bmopen*. S'il est utilisé, cela permet un skin pour ce widget.



### Note sur l'opcode *bmopen*

Bien qu'il soit mentionné, l'opcode *bmopen* n'a pas été implémenté dans Csound 4.22.

## Exécution

Les conteneurs sont utiles pour formater l'apparence graphiques des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valueurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de taux-k dans les conteneurs.

## Voir aussi

*FLgroupEnd, FLpack, FLpackEnd, FLpanel, FLpanelEnd, FLscroll, FLscrollEnd, FLtabs, FLtabsEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLgroupEnd

FLgroupEnd — Marque la fin d'un groupe de widgets FLTK enfants.

## Description

Marque la fin d'un groupe de widgets FLTK enfants.

## Syntaxe

`FLgroupEnd`

## Exécution

Les conteneurs sont utiles pour formater l'apparence graphiques des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valueurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de taux-k dans les conteneurs.

## Voir aussi

*FLgroup*, *FLpack*, *FLpackEnd*, *FLpanel*, *FLpanelEnd*, *FLscroll*, *FLscrollEnd*, *FLtabs*, *FLtabsEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLgroup\_end

FLgroup\_end — Marque la fin d'un groupe de widgets FLTK enfants.

## Description

Marque la fin d'un groupe de widgets FLTK enfants. C'est un autre nom pour **FLgroupEnd** fourni pour des raisons de compatibilité. Voir *FLgroupEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22



# FLhide

FLhide — Cache le widget FLTK cible.

## Description

Cache le widget FLTK cible, le rendant invisible.

## Syntaxe

```
FLhide ihandle
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*).

## Exécution

*FLhide* cache le widget FLTK cible, le rendant invisible.

## Voir aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLhvsBox

**FLhvsBox** — Affiche une boîte avec une grille utile pour visualiser la Synthèse Hyper Vectorielle à deux dimensions.

## Description

*FLhvsBox* affiche une boîte avec une grille utile pour visualiser la Synthèse Hyper Vectorielle à deux dimensions.

## Syntaxe

```
ihandle FLhvsBox inumlinesX, inumlinesY, iwidth, iheight, ix, iy
```

## Initialisation

*ihandle* – un identifiant (un nombre entier) défini univoquement pour identifier une boîte HVS spécifique (voir ci-dessous).

*inumlinesX*, *inumlinesY* - nombre de lignes verticales et horizontales délimitant les zones carrées HVS.

*iwidth*, *iheight* - largeur et hauteur de la boîte HVS.

*ix*, *iy* - position de la boîte HVS.

## Exécution

*FLhvsBox* est un widget capable de montrer la position courante du curseur HVS dans une boîte HVS (c'est-à-dire une zone carrée contenant une grille). Le nombre de lignes horizontales et verticales de la grille peut être défini avec les arguments *inumlinesX*, *inumlinesY*. Cet opcode doit être déclaré dans un bloc *FLpanel* - *FLpanelEnd*. Voir l'entrée de l'opcode *hvs2* pour un exemple d'utilisation de *FLhvsBox*.

*FLhvsBoxSetValue* est utilisé pour fixer la position du curseur d'un widget *FLhvsBox*.

## Voir aussi

*hvs2*, *FLhvsBoxSetValue*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# FLhvsBoxSetValue

FLhvsBoxSetValue — Fixe la position du curseur d'un widget *FLhvsBox* préalablement déclaré.

## Description

*FLhvsBoxSetValue* fixe la position du curseur d'un widget *FLhvsBox* préalablement déclaré.

## Syntaxe

**FLhvsBox** *kx*, *ky*, *ihandle*

## Initialisation

*ihandle* – un identifiant (un nombre entier) défini univoquement pour identifier une boîte HVS spécifique (voir ci-dessous).

## Exécution

*kx*, *ky* – les coordonnées de la position du curseur HVS à fixer.

*FLhvsBoxSetValue* fixe la position du curseur d'un widget *FLhvsBox* préalablement déclaré. Les arguments *kx* et *ky* dénotant la position du curseur doivent être exprimées en valeurs normalisées (comprises entre 0 et 1).

Voir l'entrée *hvs2* pour un exemple d'utilisation de *FLhvsBoxSetValue*.

## Voir aussi

*hvs2*, *FLhvsBox*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# FLjoy

FLjoy — Un opcode FLTK qui agit comme un joystick.

## Description

*FLjoy* est une zone carrée qui permet à l'utilisateur de modifier deux valeurs de sortie en même temps. Il agit comme un joystick.

## Syntaxe

```
koutx, kouty, ihandlex, ihandley FLjoy "label", iminx, imaxx, iminy, \
    imaxy, iexpx, iexpy, idispx, idispy, iwidth, iheight, ix, iy
```

## Initialisation

*ihandlex* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du valuateur. Il est automatiquement fixé par le valuateur.

*ihandley* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du valuateur. Il est automatiquement fixé par le valuateur.

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*iminx* -- valeur x minimale de l'intervalle de sortie.

*imaxx* -- valeur x maximale de l'intervalle de sortie.

*iminy* -- valeur y minimale de l'intervalle de sortie.

*imaxy* -- valeur y maximale de l'intervalle de sortie.

*idispx* -- un identifiant retourné par une instance précédente de l'opcode *FLvalue* pour afficher la valeur courante du valuateur dans le widget *FLvalue*. Si l'on ne veut pas utiliser cette possibilité d'affichage des valeurs courantes, il faut donner à cet identifiant un nombre négatif.

*idispy* -- un identifiant retourné par une instance précédente de l'opcode *FLvalue* pour afficher la valeur courante du valuateur dans le widget *FLvalue*. Si l'on ne veut pas utiliser cette possibilité d'affichage des valeurs courantes, il faut donner à cet identifiant un nombre négatif.

*iexpx* -- un nombre entier indiquant le comportement du valuateur :

- 0 = la sortie est linéaire
- -1 = la sortie est exponentielle

Tout autre nombre positif pour *iexpx* indique le numéro d'une table existante lue par indexation avec interpolation linéaire. Un numéro de table négatif supprime l'interpolation.

*iexpy* -- un nombre entier indiquant le comportement du valuateur :

- 0 = la sortie est linéaire

- -1 = la sortie est exponentielle

Tout autre nombre positif pour *iexpy* indique le numéro d'une table existante lue par indexation avec interpolation linéaire. Un numéro de table négatif supprime l'interpolation.



## IMPORTANT !

Noter que les tables utilisées par les valuateurs doivent être créées avec l'opcode *ftgen* et placées dans l'orchestre avant le valuateur correspondant. On ne peut pas les placer dans la partition. En fait, les tables placées dans la partition sont créées après l'initialisation des opcodes placés dans la section d'en-tête de l'orchestre.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

## Exécution

*koutx* -- valeur x en sortie.

*kouty* -- valeur y en sortie.

## Exemples

Voici un exemple de l'opcode FLjoy. Il utilise le fichier *FLjoy.csd* [examples/FLjoy.csd].

### Exemple 297. Exemple de l'opcode FLjoy.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc         -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLjoy.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Demonstration of the flpanel opcode
; Horizontal click-dragging controls the frequency of the oscillator
; Vertical click-dragging controls the amplitude of the oscillator
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "X Y Panel", 900, 400, 50, 50
; Minimum value output by x movement (frequency)
iminx = 200
```

```

; Maximum value output by x movement (frequency)
imaxx = 5000
; Minimum value output by y movement (amplitude)
iminy = 0
; Maximum value output by y movement (amplitude)
imaxy = 15000
; Logarithmic change in x direction
iexpx = -1
; Linear change in y direction
iexpy = 0
; Display handle x direction (-1=not used)
idispx = -1
; Display handle y direction (-1=not used)
idispy = -1
; Width of the x y panel in pixels
iwidth = 800
; Height of the x y panel in pixels
iheight = 300
; Distance of the left edge of the x y panel from
; the left edge of the panel
ix = 50
; Distance of the top edge of the x y
; panel from the top edge of the panel
iy = 50

gkfreqx, gkampy, ihandlex, ihandley FLjoy "X - Frequency Y - Amplitude", iminx, imaxx, iminy, imaxy
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
    ifn = 1
        asig oscili gkampy, gkfreqx, ifn
        out asig
    endin

</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*FLcount, FLknob, FLroller, FLslider, FLtext*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLkeyIn

**FLkeyIn** — Retourne les touches enfoncées (sur le clavier alphanumérique) quand un panneau FLTK est actif.

## Description

*FLkeyIn* nous informe sur l'état d'une touche enfoncée par l'utilisateur sur le clavier alphanumérique quand un panneau FLTK est actif.

## Syntaxe

```
kascii FLkeyIn [ifn]
```

## Initialisation

*ifn* – (facultatif, zéro par défaut) fixe le comportement de *FLkeyIn* (voir ci-dessous).

## Exécution

*kascii* - la valeur ASCII de la dernière touche enfoncée. Si la touche est enfoncée, la valeur est positive alors que si la touche est relâchée la valeur est négative.

*FLkeyIn* est utile pour savoir si une touche a été enfoncée sur le clavier de l'ordinateur. Le comportement de cet opcode dépend de l'argument facultatif *ifn*.

Si *ifn* = 0 (par défaut), *FLkeyIn* retourne le code ASCII de la dernière touche enfoncée. Si c'est une touche spéciale (ctrl, maj, alt, f1-f12, etc.), une valeur de 256 est ajoutée à la valeur retournée afin de la distinguer des touches normales. La sortie continuera à retourner la valeur de la dernière touche jusqu'à ce qu'une nouvelle touche soit enfoncée ou relâchée. Noter que la sortie sera négative lorsqu'une touche est relâchée.

Si *ifn* contient le numéro d'une table déjà allouée ayant au moins 512 éléments, l'élément de la table d'indice égal au code ASCII de la touche enfoncée est fixé à 1, tous les autres éléments de la table étant fixés à 0. Cela permet de tester l'état d'une touche particulière ou d'un ensemble de touches.

Noter qu'il faut que le paramètre *ikbdcapture* du *FLpanel* concerné doit être différent de 0 pour que *FLkeyIn* capture les événements de clavier provenant de ce panneau.



### Note

Comme *FLkeyIn* travaille en interne au taux-k, on ne peut pas l'utiliser dans l'en-tête comme les autres opcodes FLTK. On doit l'utiliser dans un instrument.

## Exemples

Voici un exemple de l'opcode *FLkeyIn*. Il utilise le fichier *FLkeyIn.csd* [exemples/FLkeyIn.csd].

### Exemple 298. Exemple de l'opcode *FLkeyIn*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

;Example by Andres Cabrera 2007

FLpanel "FLkeyIn", 400, 300, -1, -1, 5, 1, 1
FLpanelEnd

FLrun

Odbfs = 1

instr 1
kascii  FLkeyIn
ktrig  changed kascii
if (kascii > 0) then
    printf "Key Down: %i\n", ktrig, kascii
else
    printf "Key Up: %i\n", ktrig, -kascii
endif
endin

</CsInstruments>
<CsScore>
i 1 0 120
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06



# FLknob

FLknob — Un opcode de widget FLTK qui crée un bouton rotatif.

## Description

Un opcode de widget FLTK qui crée un bouton rotatif.

## Syntaxe

```
kout, ihandle FLknob "label", imin, imax, iexp, itype, idisp, iwidth, \  
ix, iy [, icursorsize]
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*). Il est automatiquement retourné par *FLknob* et ne doit pas être fixé par l'étiquette de l'utilisateur. (L'étiquette de l'utilisateur est une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.)

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*imin* -- valeur minimale de l'intervalle de sortie.

*imax* -- valeur maximale de l'intervalle de sortie.

*iexp* -- un nombre entier indiquant le comportement du valuateur :

- 0 = la sortie est linéaire
- -1 = la sortie est exponentielle

Tout autre nombre positif pour *iexp* indique le numéro d'une table existante lue par indexation avec interpolation linéaire. Un numéro de table négatif supprime l'interpolation.



### IMPORTANT !

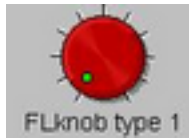
Noter que les tables utilisées par les valuateurs doivent être créées avec l'opcode *ftgen* et placées dans l'orchestre avant le valuateur correspondant. On ne peut pas les placer dans la partition. En fait, les tables placées dans la partition sont créées après l'initialisation des opcodes placés dans la section d'en-tête de l'orchestre.

*itype* -- un nombre entier indiquant l'apparence du valuateur.

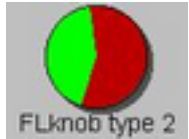
L'argument *itype* accepte les valeurs suivantes :

- 1 - un bouton rotatif 3D
- 2 - un bouton à secteur circulaire
- 3 - un bouton comme une horloge

- 4 - un bouton rotatif plat



Un bouton rotatif 3D.



Un bouton à secteur circulaire.



Un bouton comme une horloge.



Un bouton rotatif plat.

*idisp* -- un identifiant retourné par une instance précédente de l'opcode *FLvalue* pour afficher la valeur courante du valuateur dans le widget *FLvalue*. Si l'on ne veut pas utiliser cette possibilité d'affichage des valeurs courantes, il faut donner à cet identifiant un nombre négatif.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*icursorsize* (facultatif) -- Si *itype* de *FLknob* vaut 1 (bouton 3D), ce paramètre contrôle la taille du curseur du bouton.

## Exécution

*kout* -- valeur en sortie.

*FLknob* met un bouton rotatif dans le conteneur correspondant.

## Exemples

Voici un exemple de l'opcode *FLknob*. Il utilise le fichier *FLknob.csd* [examples/FLknob.csd].

## Exemple 299. Exemple de l'opcode FLknob.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLknob.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; A sine with oscillator with flknob controlled frequency
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Frequency Knob", 900, 400, 50, 50
; Minimum value output by the knob
imin = 200
; Maximum value output by the knob
imax = 5000
; Logarithmic type knob selected
iexp = -1
; Knob graphic type (1=3D knob)
itype = 1
; Display handle (-1=not used)
idisp = -1
; Width of the knob in pixels
iwidth = 70
; Distance of the left edge of the knob
; from the left edge of the panel
ix = 70
; Distance of the top edge of the knob
; from the top of the panel
iy = 125

gkfreq, ihandle FLknob "Frequency", imin, imax, iexp, itype, idisp, iwidth, ix, iy
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

; Set the widget's initial value
FLsetVal_i 300, ihandle

instr 1
iamp = 15000
ifn = 1
asig oscili iamp, gkfreq, ifn
out asig
endin

</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
```

```
i 1 0 3600
e
```

```
</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode FLknob, montrant les différents styles de boutons ainsi que l'usage de *FLvalue* pour afficher la valeur d'un bouton. Il utilise le fichier *FLknob-2.csd* [exemples/FLknob-2.csd].

### Exemple 300. Exemple plus complexe de l'opcode FLknob.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLknob.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 441
ksmps = 100
nchnls = 1

;By Andres Cabrera 2007
FLpanel "Knob Types", 330, 230, 50, 50
; Distance of the left edge of the knob
; from the left edge of the panel
ix = 20
; Distance of the top edge of the knob
; from the top of the panel
iy = 20

;Create boxes that display a widget's value
ihandleA FLvalue "A", 60, 20, ix + 130, iy + 110
ihandleB FLvalue "B", 60, 20, ix + 220, iy + 110
ihandleC FLvalue "C", 60, 20, ix + 130, iy + 160
ihandleD FLvalue "D", 60, 20, ix + 220, iy + 160

; The four types of FLknobs
gkdummy1, ihandle1 FLknob "Type 1", 200, 5000, -1, 1, ihandleA, 70, ix, iy, 90
gkdummy2, ihandle2 FLknob "Type 2", 200, 5000, -1, 2, ihandleB, 70, ix + 100, iy
gkdummy3, ihandle3 FLknob "Type 3", 200, 5000, -1, 3, ihandleC, 70, ix + 200, iy
gkdummy4, ihandle4 FLknob "Type 4", 200, 5000, -1, 4, ihandleD, 70, ix, iy + 100
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

; Set the color of widgets
FLsetColor 20, 23, 100, ihandle1
FLsetColor 0, 123, 100, ihandle2
FLsetColor 180, 23, 12, ihandle3
FLsetColor 10, 230, 0, ihandle4

FLsetColor2 200, 230, 0, ihandle1
FLsetColor2 200, 0, 123, ihandle2
FLsetColor2 180, 180, 100, ihandle3
FLsetColor2 180, 23, 12, ihandle4

; Set the initial value of the widget
```

```
FLsetVal_i 300, ihandle1
FLsetVal_i 1000, ihandle2

instr 1
; Nothing here for now
endin

</CsInstruments>
<CsScore>

f 0 3600 ;Dummy table to make csound wait for realtime events

e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*FLcount, FLjoy, FLroller, FLslider, FLtext*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLlabel

FLlabel — Un opcode FLTK qui modifie l'apparence d'une étiquette de texte.

## Description

Modifie un ensemble de paramètres en relation avec l'apparence de l'étiquette de texte d'un widget (c'est-à-dire la taille, la police, l'alignement et la couleur du texte correspondant).

## Syntaxe

```
FLlabel isize, ifont, ialign, ired, igreen, iblue
```

## Initialisation

*isize* -- taille de la police du widget cible. Les valeurs normales sont de l'ordre de 15. Des nombres plus grands augmentent la taille de la police, tandis que des nombres plus petits la réduisent.

*ifont* -- fixe le type de police de l'étiquette d'un widget.

Les valeurs admises pour l'argument *ifont* sont :

- 1 - Helvética (comme Arial sous Windows)
- 2 - Helvética Gras
- 3 - Helvética Italique
- 4 - Helvética Gras Italique
- 5 - Courier
- 6 - Courier Gras
- 7 - Courier Italique
- 8 - Courier Gras Italique
- 9 - Times
- 10 - Times Grasd
- 11 - Times Italique
- 12 - Times Gras Italique
- 13 - Symbole
- 14 - Ecran
- 15 - Ecran Gras
- 16 - Dingbats

*ialign* -- fixe l'alignement de l'étiquette de texte du widget.

Les valeurs admises pour l'argument *ialign* sont :

- 1 - centré
- 2 - en haut
- 3 - en bas
- 4 - à gauche
- 5 - à droite
- 6 - en haut à gauche
- 7 - en haut à droite
- 8 - en bas à gauche
- 9 - en bas à droite

*ired* -- la couleur rouge du widget cible. L'intervalle pour chaque composante RVB va de 0 à 255.

*igreen* -- la couleur verte du widget cible. L'intervalle pour chaque composante RVB va de 0 à 255.

*ibblue* -- la couleur bleue du widget cible. L'intervalle pour chaque composante RVB va de 0 à 255.

## Exécution

*FLlabel* modifie un ensemble de paramètres en relation avec l'apparence de l'étiquette de texte d'un widget, c'est-à-dire la taille, la police, l'alignement et la couleur du texte correspondant. Cet opcode affecte (presque) tous les widgets définis après lui. On peut placer plusieurs instances de *FLlabel* avant chaque widget que l'on veut modifier. Cependant, pour modifier un widget particulier, il vaut mieux utiliser l'opcode appartenant au second type (c'est-à-dire ceux ayant un argument *ihandle*).

l'influence de *FLlabel* sur le widget suivant peut être suspendue en lui donnant -1 comme seul argument. *FLlabel* est conçu pour modifier les attributs de texte d'un groupe de widgets relatifs.

## Voir aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLloadsnap

FLloadsnap — Charge tous les instantanés dans la banque de mémoire de l'orchestre courant.

## Description

*FLloadsnap* charge tous les instantanés contenus dans un fichier dans la banque de mémoire de l'orchestre courant.

## Syntaxe

```
FLloadsnap "filename" [, igroup]
```

## Initialisation

*"filename"* -- une chaîne de caractères entre guillemets correspondant au fichier à partir duquel charger une banque d'instantanés.

*igroup* -- (facultatif) un nombre entier faisant référence à un groupe de widgets en relation avec un instantané. Cela permet de lire/écrire, ou charger/sauvegarder l'état d'un sous-ensemble de valuateurs. La valeur par défaut est zéro qui fait référence au premier groupe. Le numéro de groupe est déterminé par l'opcode *FLsetSnapGroup*.



### Note

Le paramètre *igroup* n'a pas encore été complètement implémenté dans la version actuelle de Csound. Prière de ne pas s'y fier.

## Exécution

*FLloadsnap* charge tous les instantanés contenus dans *filename* dans la banque de mémoire de l'orchestre courant.

Pour économiser la mémoire, les widgets peuvent être groupés afin que les instantanés n'affectent qu'un groupe défini de widgets. L'opcode *FLsetSnapGroup* est utilisé pour spécifier le groupe de tous les widgets déclarés après lui jusqu'à la prochaine instruction *FLsetSnapGroup*.

## Voir aussi

*FLgetsnap*, *FLrun*, *FLsetSnapGroup*, *FLsavesnap*, *FLsetsnap*, *FLupdate*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22



# FLmouse

FLmouse — Retourne la position de la souris et l'état de ses trois boutons.

## Description

*FLmouse* retourne les coordonnées de la position de la souris dans un panneau FLTK et l'état de ses trois boutons.

## Syntaxe

`kx, ky, kb1, kb2, kb3 FLmouse [imode]`

## Initialisation

*imode* – (facultatif, 0 par défaut) Détermine la façon de rapporter la position de la souris.

- 0 - Position absolue normalisée dans l'intervalle 0-1
- 1 - Position absolue en pixel brut
- 2 - Position en pixel brut, relative au panneau FLTK

## Exécution

*kx, ky* – les coordonnées de la souris, dont l'intervalle dépend de l'argument *imode* (voir ci-dessus).

*kb1, kb2, kb3* – les états des boutons de la souris, 1 lorsque le bouton correspondant est enfoncé, 0 lorsqu'il est relâché.

*FLmouse* retourne les coordonnées de la position de la souris dans un panneau FLTK et l'état de ses trois boutons. Les coordonnées peuvent être récupérées selon trois modes dépendant de la valeur de l'argument *imode* (voir ci-dessus). Les modes 0 et 1 retournent la position de la souris par rapport à l'écran complet (mode absolu), tandis que le mode 2 retourne la position en pixels dans un panneau FLTK. Noter que *FLmouse* n'est actif que lorsque le curseur de la souris se trouve sur une zone *FLpanel*.

## Exemples

Voici un exemple de l'opcode *FLmouse*. Il utilise le fichier *FLmouse.csd* [exemples/FLmouse.csd].

### Exemple 301. Exemple de l'opcode FLmouse.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d          ;;RT audio I/O
</CsOptions>
<CsInstruments>
```

```

sr=44100
ksmps=128
nchnls=2

;Example by Andres Cabrera 2007
giwidth = 400
giheight = 300
FLpanel "FLmouse", giwidth, giheight, 10, 10
FLpanelEnd

FLrun

Odbfs = 1

instr 1
  kx, ky, kb1, kb2, kb3    FLmouse 2
  ktrig changed kx, ky  ;Print only if coordinates have changed
  printf "kx = %f   ky = %f \n", ktrig, kx, ky
  kfreq = ((giwidth - ky)*1000/giwidth) + 300

  ; y coordinate determines frequency, x coordinate determines amplitude
  ; Left mouse button (kb1) doubles the frequency
  ; Right mouse button (kb3) activates sound on channel 2
  aout oscil kx /giwidth , kfreq * (kb1 + 1), 1
  outs aout, aout * kb3
endin

</CsInstruments>
<CsScore>
f 1 0 1024 10 1

i 1 0 120
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# flooper

flooper — Lecture en boucle basée sur une table de fonction avec fondu enchainé.

## Description

Cet opcode lit les données audio d'une table de fonction et les restitue dans une boucle dont le début, la durée et l'étendue du fondu enchainé sont fixés par l'utilisateur. On peut également contrôler la hauteur de la boucle ainsi que sa lecture à l'envers. Il accepte des tables dont la longueur n'est pas une puissance de deux, telles que celles de GEN01 à allocation différée, avec un ou deux canaux.

## Syntaxe

```
asig1[, asig2] flooper kamp, kpitch, istart, idur, ifad, ifn
```

## Initialisation

*istart* -- début de la boucle en secondes

*idur* -- durée de la boucle en secondes

*ifad* -- étendue du fondu enchainé en secondes

*ifn* -- numéro de la table de fonction, généralement créée au moyen de GEN01

## Exécution

*asig1[, asig2]* -- signal de sortie (mono ou stéréo).

*kamp* -- contrôle de l'amplitude

*kpitch* -- contrôle de la hauteur (rapport de transposition) ; avec des valeurs négatives, la boucle est lue à l'envers.

## Exemples

### Exemple 302.

```
aout flooper 16000, 1, 1, 4, 0.05, 1 ; loop starts at 1 sec, for 4 secs, 0.05 crossfade  
out      aout
```

L'exemple ci-dessus montre l'opération de base de *flooper*. La hauteur peut être contrôlée au taux-k ainsi que l'amplitude. L'exemple suppose que la table 1 contient au moins 5.05 secondes de données audio (boucle durant 4 secondes, commençant 1 seconde après le début de la table, avec un fondu enchainé de 0.05 secondes après la fin de la boucle).

Voici un autre exemple de l'opcode *flooper*. Il utilise les fichiers *flooper.csd* [exemples/flooper.csd] et *fox.wav* [exemples/fox.wav].

### Exemple 303.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o flooper.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kpitch line 1, p3, .9 ;lower pitch a bit during the note
aout flooper .9, kpitch, 1, .53, 0.05, 1 ; loop starts at 1 sec, for .53 secs, 0.05 crossfade
outs aout, aout

endin
</CsInstruments>
<CsScore>
;table size is deferred,
; and format taken from the soundfile header
f 1 0 0 1 "fox.wav" 0 0 0

i 1 0 8.2
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Victor Lazzarini

Avril 2005

Nouveau greffon dans la version 5

Avril 2005.

# flooper2

flooper2 — Lecture en boucle basée sur une table de fonction avec fondu enchainé.

## Description

Cet opcode implémente une lecture de boucle avec fondu enchainé, avec des paramètres variables, trois modes de boucle, et une utilisation facultative d'une table pour la forme du fondu enchainé. Il accepte comme source audio des tables dont la longueur n'est pas une puissance de deux, telles que celles de GEN01 à allocation différée, avec un ou deux canaux.

## Syntaxe

```
asig1[,asig2] flooper2 kamp, kpitch, kloopstart, kloopend, kcrossfade, ifn \
[, istart, imode, ifenv, iskip]
```

## Initialisation

*ifn* -- numéro de la table de fonction de la source audio, généralement créée au moyen de GEN01

*istart* -- début de la lecture en secondes

*imode* -- modes de boucle : 0 à l'endroit, 1 à l'envers, 2 à l'endroit et à l'envers. (0 par défaut)

*ifenv* -- s'il est non nul, numéro de la table de la forme de l'enveloppe du fondu enchainé. La valeur par défaut, 0, donne un fondu enchainé linéaire.

*iskip* -- s'il vaut 1, l'initialisation de l'opcode est ignorée, pour les notes liées, l'exécution continuant depuis la position dans la boucle où la note précédente s'est terminée. Avec la valeur par défaut, 0, l'initialisation a lieu.

## Exécution

*asig1[,asig2]* -- signal de sortie (mono ou stéréo).

*kamp* -- contrôle de l'amplitude

*kpitch* -- contrôle de la hauteur (rapport de transposition) ; les valeurs négatives sont interdites.

*kloopstart* -- début de la boucle (en secondes). Noter que bien qu'étant de taux-k, les paramètres de boucle comme celui-ci ne sont mis à jour qu'une fois par itération de la boucle.

*kloopend* -- fin de la boucle (en secondes), mis à jour une seule fois par itération de la boucle.

*kcrossfade* -- longueur du fondu enchainé (en secondes), mis à jour une seule fois par itération de la boucle et limité par la longueur de la boucle.

Avec le mode 1 de *imode* la boucle ne se fait qu'à l'envers depuis la fin jusqu'au début.

## Exemples

**Exemple 304.**

```

aout flooper2 16000, 1, 1, 5, 0.05, 1 ; loop starts at 1 sec, for 4 secs, 0.05 crossfade
out      aout
```

L'exemple ci-dessus montre l'opération de base de *flooper2*. La hauteur peut être contrôlée au taux-k ainsi que l'amplitude et les paramètres de boucle. L'exemple suppose que la table 1 contient au moins 5.05 secondes de données audio (boucle durant 4 secondes, commençant 1 seconde après le début de la table, avec un fondu enchaîné de 0.05 secondes après la fin de la boucle). La boucle est en mode 0 (boucle normale à l'endroit).

Voici un autre exemple de l'opcode *flooper2*. Il utilise le fichier *flooper2.csd* [exemples/flooper2.csd] et *fox.wav* [exemples/fox.wav].

### Exemple 305.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o flooper2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
; looping back and forth, 0.05 crossfade
kst line .2, p3, 2 ;vary loopstartpoint
aout flooper2 .8, 1, 0, kst, 0.05, 1, 0, 2
outs      aout, aout

endin
</CsInstruments>
<CsScore>
; Its table size is deferred,
; and format taken from the soundfile header
f 1 0 0 1 "fox.wav" 0 0 0

i 1 0 12
e
</CsScore>
</CsSoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
Juillet 2006

Nouveau greffon dans la version 5

Juillet 2006.

# floor

floor — Retourne le plus grand entier inférieur ou égal à  $x$ .

## Description

Retourne le plus grand entier inférieur ou égal à  $x$ .

## Syntaxe

**floor**( $x$ ) (argument au taux d'initialisation, de contrôle ou audio)

**floor**( $k/i[]$ ) ( $k$ - ou  $i$ -tableau)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode floor. Il utilise le fichier *floor.csd* [examples/floor.csd].

### Exemple 306. Exemple de l'opcode floor.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o floor.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

idiv init 1

loop:
inumber = 9
il = inumber / idiv
ifl = floor(il)
print inumber, idiv, ifl ;print number / idiv = result using floor
idiv = idiv + 1
if (idiv <= 10) igoto loop

endin
</CsInstruments>
<CsScore>
```

```
i 1 0 0  
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
instr 1: inumber = 9.000 idiv = 1.000 ifl = 9.000  
instr 1: inumber = 9.000 idiv = 2.000 ifl = 4.000  
instr 1: inumber = 9.000 idiv = 3.000 ifl = 3.000  
instr 1: inumber = 9.000 idiv = 4.000 ifl = 2.000  
instr 1: inumber = 9.000 idiv = 5.000 ifl = 1.000  
instr 1: inumber = 9.000 idiv = 6.000 ifl = 1.000  
instr 1: inumber = 9.000 idiv = 7.000 ifl = 1.000  
instr 1: inumber = 9.000 idiv = 8.000 ifl = 1.000  
instr 1: inumber = 9.000 idiv = 9.000 ifl = 1.000  
instr 1: inumber = 9.000 idiv = 10.000 ifl = 0.000
```

## Voir aussi

*abs, exp, int, log, log10, i, sqrt*

## Crédits

Auteur : Istvan Varga  
Nouveau dans Csound 5  
2005



# FLpack

FLpack — Permet de concentrer et d'aligner des widgets FLTK.

## Description

*FLpack* permet de concentrer et d'aligner des widgets.

## Syntaxe

**FLpack** *iwidth, iheight, ix, iy, itype, ispace, iborder*

## Initialisation

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*itype* -- un nombre entier modifiant l'apparence du widget cible.

L'argument *itype* exprime le type de concentration :

- 0 - verticale
- 1 - horizontale

*ispace* -- fixe l'espace entre les widgets.

*iborder* -- type de la bordure du conteneur. Il est exprimé par un nombre entier choisi parmi les suivants :

- 0 - pas de bordure
- 1 - bordure de boîte en creux
- 2 - bordure de boîte saillante
- 3 - bordure gravée
- 4 - bordure en relief
- 5 - bordure ligne noire
- 6 - mince bordure en creux
- 7 - mince bordure saillante

## Exécution

*FLpack* permet de concentrer et d'aligner des widgets.

Les conteneurs sont utiles pour formater l'apparence graphique des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valueurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de taux-k dans les conteneurs.

## Exemples

L'exemple suivant :

```

FLpanel "Panel1", 450, 300, 100, 100
FLpack 400, 300, 10, 40, 0, 15, 3
gk1, ihs1      FLslider      "FLslider 1", 500, 1000, 2, 1, -1, 300, 15, 20, 50
gk2, ihs2      FLslider      "FLslider 2", 300, 5000, 2, 3, -1, 300, 15, 20, 100
gk3, ihs3      FLslider      "FLslider 3", 350, 1000, 2, 5, -1, 300, 15, 20, 150
gk4, ihs4      FLslider      "FLslider 4", 250, 5000, 1, 11, -1, 300, 30, 20, 200
gk5, ihs5      FLslider      "FLslider 5", 220, 8000, 2, 1, -1, 300, 15, 20, 250
gk6, ihs6      FLslider      "FLslider 6", 1, 5000, 1, 13, -1, 300, 15, 20, 300
gk7, ihs7      FLslider      "FLslider 7", 870, 5000, 1, 15, -1, 300, 30, 20, 350
FLpackEnd
FLpanelEnd

```

...produira ce résultat, lorsque l'on changera la taille de la fenêtre :



FLpack.

## Voir aussi

*FLgroup, FLgroupEnd, FLpackEnd, FLpanel, FLpanelEnd, FLscroll, FLscrollEnd, FLtabs, FLtabsEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLpackEnd

FLpackEnd — Marque la fin d'un groupe de widgets FLTK concentrés ou alignés.

## Description

Marque la fin d'un groupe de widgets FLTK concentrés ou alignés.

## Syntaxe

`FLpackEnd`

## Exécution

Les conteneurs sont utiles pour formater l'apparence graphiques des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valueurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de taux-k dans les conteneurs.

## Voir aussi

*FLgroup*, *FLgroupEnd*, *FLpack*, *FLpanel*, *FLpanelEnd*, *FLscroll*, *FLscrollEnd*, *FLtabs*, *FLtabsEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLpack\_end

FLpack\_End — Marque la fin d'un groupe de widgets FLTK concentrés ou alignés.

## Description

Marque la fin d'un groupe de widgets FLTK concentrés ou alignés. C'est un autre nom pour **FLpackEnd** fourni pour des raisons de compatibilité. Voir *FLpackEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLpanel

FLpanel — Crée une fenêtre contenant des widgets FLTK.

## Description

Crée une fenêtre contenant des widgets FLTK.

## Syntaxe

```
FLpanel "label", iwidth, iheight [, ix] [, iy] [, iborder] [, ikbdcapture] [, iclose]
```

## Initialisation

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* (facultatif) -- position horizontale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* (facultatif) -- position verticale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iborder* (facultatif) -- type de la bordure du conteneur. Il est exprimé par un nombre entier choisi parmi les suivants :

- 0 - pas de bordure
- 1 - bordure de boîte en creux
- 2 - bordure de boîte saillante
- 3 - bordure gravée
- 4 - bordure en relief
- 5 - bordure ligne noire
- 6 - mince bordure en creux
- 7 - mince bordure saillante

*ikbdcapture* (0 par défaut) -- Si cet indicateur est positionné à 1, les événements du clavier sont capturés par la fenêtre (pour une utilisation par *sensekey* et par *FLkeyIn*)

*iclose* (0 par défaut) -- Si cet indicateur contient une valeur différente de 0, le bouton de fermeture de la fenêtre est désactivé, et la fenêtre ne peut pas être fermée directement par l'utilisateur. Elle se fermera à la sortie de Csound.

## Exécution

Les conteneurs sont utiles pour formater l'apparence graphique des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valuateurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de taux-k dans les conteneurs.

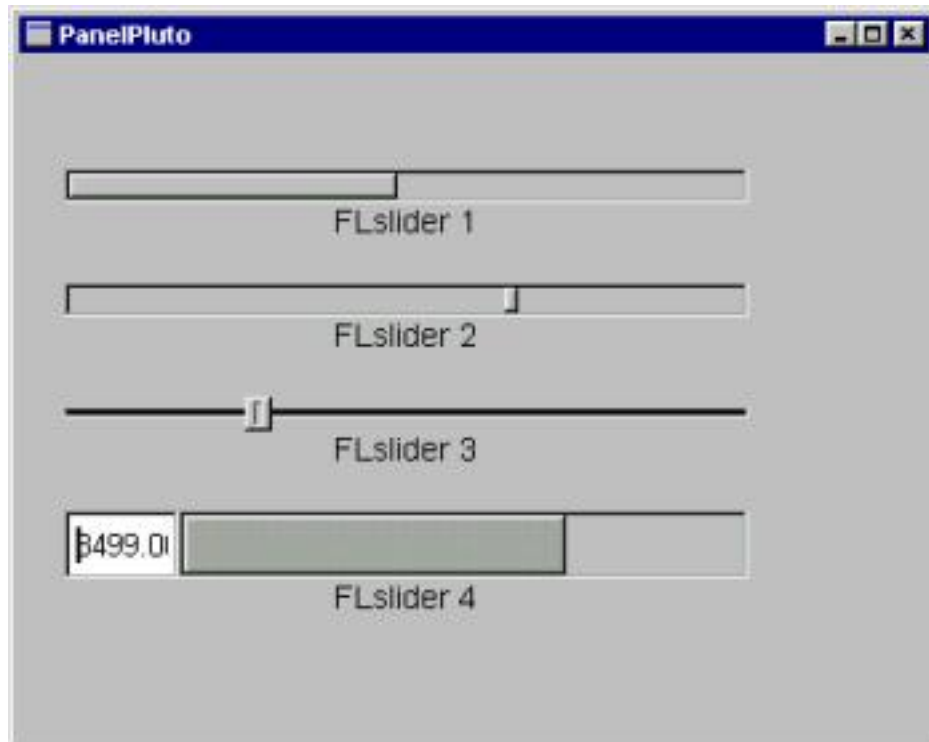
*FLpanel* crée une fenêtre. Il doit être suivi d'un opcode *FLpanelEnd* après que tous les widgets qu'il contient aient été déclarés. Par exemple :

```

      FLpanel      "PanelPluto",450,550,100,100 ;***** start of container
gk1, ih1 FLslider  "FLslider 1", 500, 1000, 2 ,1, -1, 300,15, 20,50
gk2, ih2 FLslider  "FLslider 2", 300, 5000, 2 ,3, -1, 300,15, 20,100
gk3, ih3 FLslider  "FLslider 3", 350, 1000, 2 ,5, -1, 300,15, 20,150
gk4, ih4 FLslider  "FLslider 4", 250, 5000, 1 ,11,-1, 300,30, 20,200
      FLpanelEnd ;***** end of container

```

produira le résultat suivant :



*FLpanel*.

Si l'indicateur *ikbdcapture* est positionné, la fenêtre capture les événements du clavier et les envoie à *sensekey*. Cet indicateur modifie le comportement de *sensekey*, et lui fait recevoir les événements depuis la fenêtre FLTK au lieu de stdin.

## Exemples

Voici un exemple de l'opcode *FLpanel*. Il utilise le fichier *FLpanel.csd* [examples/FLpanel.csd].

### Exemple 307. Exemple de l'opcode FLpanel.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLpanel.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Creates an empty window panel
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

; Panel height in pixels
ipanelheight = 900
; Panel width in pixels
ipanelwidth = 400
; Horizontal position of the panel on screen in pixels
ix = 50
; Vertical position of the panel on screen in pixels
iy = 50

FLpanel "A Window Panel", ipanelheight, ipanelwidth, ix, iy
; End of panel contents
FLpanelEnd

;Run the widget thread!
FLrun

instr 1
endin

</CsInstruments>
<CsScore>

; 'Dummy' score event of 1 hour.
f 0 3600
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*FLgroup, FLgroupEnd, FLpack, FLpackEnd, FLpanelEnd, FLscroll, FLscrollEnd, FLtabs, FLtabsEnd, sensekey*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22



Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLpanelEnd

FLpanelEnd — Marque la fin d'un groupe de widgets FLTK contenus dans une fenêtre (panel).

## Description

Marque la fin d'un groupe de widgets FLTK contenus dans une fenêtre (panel).

## Syntaxe

`FLpanelEnd`

## Exécution

Les conteneurs sont utiles pour formater l'apparence graphiques des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valueurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de taux-k dans les conteneurs.

## Voir aussi

*FLgroup*, *FLgroupEnd*, *FLpack*, *FLpackEnd*, *FLpanel*, *FLscroll*, *FLscrollEnd*, *FLtabs*, *FLtabsEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLpanel\_end

FLpanel\_end — Marque la fin d'un groupe de widgets FLTK contenus dans une fenêtre (panel).

## Description

Marque la fin d'un groupe de widgets FLTK contenus dans une fenêtre (panel). C'est un autre nom pour **FLpanelEnd** fourni pour des raisons de compatibilité. Voir *FLpanelEnd*.

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLprintk

FLprintk — Un opcode FLTK qui imprime une valeur de taux-k à intervalles donnés.

## Description

*FLprintk* est semblable à *printk* mais il montre les valeurs d'un signal de taux-k dans un champ texte plutôt que dans la console.

## Syntaxe

```
FLprintk itime, kval, idisp
```

## Initialisation

*itime* -- l'intervalle de temps en secondes entre deux mises à jour de l'affichage.

*idisp* -- la valeur d'un identifiant retourné par une instance précédente de l'opcode *FLvalue* pour afficher la valeur courante dans le widget *FLvalue*. Si l'on ne veut pas utiliser cette possibilité d'affichage des valeurs courantes, il faut fixer ce paramètre à une valeur négative.

## Exécution

*kval* -- signal de taux-k à afficher.

*FLprintk* est semblable à *printk*, mais il montre les valeurs d'un signal de taux-k dans un champ texte plutôt que dans la console. L'argument *idisp* doit contenir la valeur du *ihandle* retourné par un opcode *FLvalue* précédent. Alors que *FLvalue* doit être placé dans la section d'en-tête d'un orchestre dans un bloc *FLpanel/FLpanelEnd*, *FLprintk* doit être placé dans un instrument pour opérer correctement. Pour cette raison, il ralentit l'exécution et il ne faut l'utiliser que pour des raisons de débogages.

## Voir aussi

*FLbox*, *FLbutBank*, *FLbutton*, *FLprintk2*, *FLvalue*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLprintk2

FLprintk2 — Un opcode FLTK qui imprime une nouvelle valeur chaque fois qu'une variable au taux-k change.

## Description

*FLprintk2* est semblable à *FLprintk* mais il ne montre la valeur d'une variable de taux-k que lorsqu'elle change.

## Syntaxe

```
FLprintk2 kval, idisp
```

## Initialisation

*idisp* -- la valeur d'un identifiant retourné par une instance précédente de l'opcode *FLvalue* pour afficher la valeur courante dans le widget *FLvalue*. Si l'on ne veut pas utiliser cette possibilité d'affichage des valeurs courantes, il faut fixer ce paramètre à une valeur négative.

## Exécution

*kval* -- signal de taux-k à afficher.

*FLprintk2* est semblable à *FLprintk*, mais il ne montre la valeur d'une variable de taux-k que lorsqu'elle change. Utile pour surveiller les changements de contrôle MIDI lorsque l'on utilise des réglettes. Il ne faut l'utiliser que pour des raisons de débogages, car il ralentit l'exécution.

## Voir aussi

*FLbox*, *FLbutBank*, *FLbutton*, *FLprintk*, *FLvalue*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLroller

FLroller — Un widget FLTK qui crée une molette.

## Description

*FLroller* est une sorte de bouton rotatif, mais disposé transversalement (une molette).

## Syntaxe

```
kout, ihandle FLroller "label", imin, imax, istep, iexp, itype, idisp, \  
        iwidth, iheight, ix, iy
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*). Il est automatiquement retourné par *FLroller* et ne doit pas être fixé par l'étiquette de l'utilisateur. (L'étiquette de l'utilisateur est une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.)

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*imin* -- mvaleur minimale de l'intervalle de sortie.

*imax* -- valeur maximale de l'intervalle de sortie.

*istep* -- un nombre en virgule flottante indiquant le pas d'incréméntation du valuateur à chaque clic de souris. L'argument *istep* permet de ralentir le mouvement de la molette, ce qui autorise une précision arbitraire.

*iexp* -- un nombre entier indiquant le comportement du valuateur :

- 0 = la sortie est linéaire
- -1 = la sortie est exponentielle

Tout autre nombre positif pour *iexp* indique le numéro d'une table existante lue par indexation avec interpolation linéaire. Un numéro de table négatif supprime l'interpolation.



### IMPORTANT !

Noter que les tables utilisées par les valuateurs doivent être créées avec l'opcode *ftgen* et placées dans l'orchestre avant le valuateur correspondant. On ne peut pas les placer dans la partition. En fait, les tables placées dans la partition sont créées après l'initialisation des opcodes placés dans la section d'en-tête de l'orchestre.

*itype* -- un nombre entier indiquant l'apparence du valuateur.

L'argument *itype* accepte les valeurs suivantes :

- 1 - molette horizontale

- 2 - molette verticale

*idisp* -- un identifiant retourné par une instance précédente de l'opcode *FLvalue* pour afficher la valeur courante du valuateur dans le widget *FLvalue*. Si l'on ne veut pas utiliser cette possibilité d'affichage des valeurs courantes, il faut donner à cet identifiant un nombre négatif.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

## Exécution

*kout* -- valeur en sortie.

*FLroller* est une sorte de bouton rotatif, mais disposé transversalement (une molette) :



FLroller.

## Exemples

Voici un exemple de l'opcode *FLroller*. Il utilise le fichier *FLroller.csd* [exemples/FLroller.csd].

### Exemple 308. Exemple de l'opcode *FLroller*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLroller.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; A sine with oscillator with flroller controlled frequency
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Frequency Roller", 900, 400, 50, 50
; Minimum value output by the roller
imin = 200
; Maximum value output by the roller
imax = 5000
; Increment with each pixel
```

```

        istep = 1
        ; Logarithmic type roller selected
        iexp = -1
        ; Roller graphic type (1=horizontal)
        itype = 1
        ; Display handle (-1=not used)
        idisp = -1
        ; Width of the roller in pixels
        iwidth = 300
        ; Height of the roller in pixels
        iheight = 50
        ; Distance of the left edge of the knob
        ; from the left edge of the panel
        ix = 300
        ; Distance of the top edge of the knob
        ; from the top edge of the panel
        iy = 50

        gkfreq, ihandle FLroller "Frequency", imin, imax, istep, iexp, itype, idisp, iwidth, iheight, ix, iy
    ; End of panel contents
FLpanelEnd
    ; Run the widget thread!
FLrun

instr 1
    iamp = 15000
    ifn = 1
    asig oscili iamp, gkfreq, ifn
    out asig
endin

</CsInstruments>
<CsScore>

    ; Function table that defines a single cycle
    ; of a sine wave.
    f 1 0 1024 10 1

    ; Instrument 1 will play a note for 1 hour.
    i 1 0 3600
    e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*FLcount, FLjoy, FLknob, FLslider, FLtext*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.



# FLrun

FLrun — Démarre le processus léger des widgets FLTK.

## Description

Démarre le processus léger des widgets FLTK.

## Syntaxe

`FLrun`

## Exécution

Cet opcode doit être placé à la fin des déclaration de tous les widgets. Il n'a pas d'arguments et il sert à démarrer le processus léger relatif aux widgets. Les widgets ne seront pas opérationnels si *FLrun* est absent.

## Voir aussi

*FLgetsnap*, *FLloadsnap*, *FLsavesnap*, *FLsetsnap*, *FLupdate*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsavesnap

FLsavesnap — Sauvegarde dans un fichier tous les instantanés actuellement créés.

## Description

*FLsavesnap* sauvegarde dans un fichier tous les instantanés actuellement créés (c'est-à-dire la banque de mémoire en entier).

## Syntaxe

**FLsavesnap** "filename" [, igroup]

## Initialisation

« *filename* » -- une chaîne de caractères entre guillemets correspondant à un fichier dans lequel sauvegarder une banque d'instantanés.

*igroup* -- (facultatif) un nombre entier faisant référence à un groupe de widgets en relation avec un instantané. Cela permet de lire/écrire, ou charger/sauvegarder l'état d'un sous-ensemble de valueurs. La valeur par défaut est zéro qui fait référence au premier groupe. Le numéro de groupe est déterminé par l'opcode *FLsetSnapGroup*.



### Note

Le paramètre *igroup* n'a pas encore été complètement implémenté dans la version actuelle de Csound. Prière de ne pas s'y fier.

## Exécution

*FLsavesnap* sauvegarde tous les instantanés actuellement créés (c'est-à-dire la banque de mémoire en entier) dans un fichier dont le nom est *filename*. Comme le fichier est un fichier texte, les valeurs des instantanés peuvent être modifiées manuellement dans un éditeur de texte. Le format des données du fichier est le suivant (pour le moment, ceci pouvant changer dans une prochaine version de Csound) :

```
----- 0 -----
FLvalue 0 0 1 0 ""
FLvalue 0 0 1 0 ""
FLvalue 0 0 1 0 ""
FLslider 331.946 80 5000 -1 "frequency of the first oscillator"
FLslider 385.923 80 5000 -1 "frequency of the second oscillator"
FLslider 80 80 5000 -1 "frequency of the third oscillator"
FLcount 0 0 10 0 "this index must point to the location number where snapshot is stored"
FLbutton 0 0 1 0 "Store snapshot to current index"
FLbutton 0 0 1 0 "Save snapshot bank to disk"
FLbutton 0 0 1 0 "Load snapshot bank from disk"
FLbox 0 0 1 0 ""
----- 1 -----
FLvalue 0 0 1 0 ""
FLvalue 0 0 1 0 ""
FLvalue 0 0 1 0 ""
FLslider 819.72 80 5000 -1 "frequency of the first oscillator"
FLslider 385.923 80 5000 -1 "frequency of the second oscillator"
FLslider 80 80 5000 -1 "frequency of the third oscillator"
FLcount 1 0 10 0 "this index must point to the location number where snapshot is stored"
FLbutton 0 0 1 0 "Store snapshot to current index"
```

```
FLbutton 0 0 1 0 "Save snapshot bank to disk"
FLbutton 0 0 1 0 "Load snapshot bank from disk"
FLbox 0 0 1 0 ""
----- 2 -----
..... etc...
----- 3 -----
..... etc...
-----
```

Comme on peut le voir, chaque instantané contient plusieurs lignes. Chaque instantané est séparé du précédent et du suivant par une ligne de cette sorte :

"----- Num d'instantané -----"

Suivent plusieurs lignes contenant les données. Chacune de ces lignes correspond à un widget.

Le premier champ de chaque ligne est une chaîne de caractères sans guillemets contenant le nom de l'opcode correspondant au widget. Le second champ est un nombre exprimant la valeur courante d'un instantané. Dans la version actuelle, c'est le seul champ que l'on peut modifier manuellement. Les troisième et quatrième champs montrent les valeurs minimale et maximale pour ce valuateur. Le cinquième champ est un nombre spécial qui indique si le valuateur est linéaire (valeur 0), exponentiel (valeur -1), ou est indexé par une table avec interpolation des valeurs (numéro de table négatif) ou sans interpolation (numéro de table positif). Le dernier champ, une chaîne de caractères entre guillemets, contient l'étiquette du widget. La dernière ligne du fichier est toujours

"-----"

.

Noter que *FLvalue* et *FLbox* ne sont pas des valuateurs et que leurs valeurs sont constantes, ne pouvant pas être modifiées.

Pour économiser la mémoire, les widgets peuvent être groupés afin que les instantanés n'affectent qu'un groupe défini de widgets. L'opcode *FLsetSnapGroup* est utilisé pour spécifier le groupe de tous les widgets déclarés après lui jusqu'à la prochaine instruction *FLsetSnapGroup*.

## Exemples

Voici un exemple simple de sauvegarde d'un instantané FLTK. Il utilise le fichier *FLsavesnap\_simple.csd* [examples/FLsavesnap\_simple.csd].

### Exemple 309. Exemple de sauvegarde d'un instantané FLTK.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=48000
ksmps=128
```

```

nchnls=2

; Example by Hector Centeno and Andres Cabrera 2007

; giSWMtab4 ftgen 0, 0, 513, 21, 10, 1, .3
; giSWMtab4M ftgen 0, 0, 64, 7, 1, 50, 1

FLpanel "Snapshots", 530, 190, 40, 410, 3
  FLcolor 100, 118, 140
  ivalSM1 FLvalue "", 70, 20, 270, 20
  gksliderA, gislidSM1 FLslider "Slider", -4, 4, 0, 3, ivalSM1, 250, 20, 20, 20
  itext1 FLbox "store", 1, 1, 14, 50, 25, 355, 15
  itext2 FLbox "load", 1, 1, 14, 50, 25, 415, 15
  gksnap, ibuttn1 FLbutton "1", 1, 0, 11, 25, 25, 364, 45, 0, 3, 0, 3, 1
  gksnap, ibuttn2 FLbutton "2", 1, 0, 11, 25, 25, 364, 75, 0, 3, 0, 3, 2
  gksnap, ibuttn3 FLbutton "3", 1, 0, 11, 25, 25, 364, 105, 0, 3, 0, 3, 3
  gksnap, ibuttn4 FLbutton "4", 1, 0, 11, 25, 25, 364, 135, 0, 3, 0, 3, 4

  gkload, ibuttn1 FLbutton "1", 1, 0, 11, 25, 25, 424, 45, 0, 4, 0, 3, 1
  gkload, ibuttn2 FLbutton "2", 1, 0, 11, 25, 25, 424, 75, 0, 4, 0, 3, 2
  gkload, ibuttn3 FLbutton "3", 1, 0, 11, 25, 25, 424, 105, 0, 4, 0, 3, 3
  gkload, ibuttn4 FLbutton "4", 1, 0, 11, 25, 25, 424, 135, 0, 4, 0, 3, 4

  ivalSM2 FLvalue "", 70, 20, 270, 80
  gkknobA, gislidSM2 FLknob "Knob", -4, 4, 0, 3, ivalSM2, 60, 120, 60
FLpanelEnd
FLsetVal_i 1, gislidSM1
FLsetVal_i 1, gislidSM2
FLrun

  instr 1

  endin

instr 3 ; Save snapshot
index init 0
ipstno = p4
Sfile sprintf "snapshot_simple.%d.snap", ipstno

inumsnap, inumval FLsetsnap index ;, -1, igroup
FLsavesnap Sfile

  endin

instr 4 ;Load snapshot
index init 0
ipstno = p4
Sfile sprintf "snapshot_simple.%d.snap", ipstno

FLloadsnap Sfile
inumload FLgetsnap index ;, igroup

  endin

</CsInstruments>

<CsScore>
f 0 3600

e

</CsScore>

</CsoundSynthesizer>

```

Voici un autre exemple de sauvegarde d'instantané FLTK utilisant des groupes d'instantanés. Il utilise le fichier *FLsavesnap.csd* [exemples/FLsavesnap.csd].

### Exemple 310. Exemple de sauvegarde d'instantané FLTK utilisant des groupes d'instantanés.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=48000
ksmps=128
nchnls=2

; Example by Hector Centeno and Andres Cabrera 2007

; giSWMtab4 ftgen 0, 0, 513, 21, 10, 1, .3
; giSWMtab4M ftgen 0, 0, 64, 7, 1, 50, 1

FLpanel "Snapshots", 530, 350, 40, 410, 3
  FLcolor 100, 118, 140
  FLsetSnapGroup 0
    ivalSM1   FLvalue  "", 70, 20, 270, 20
    ivalSM2   FLvalue  "", 70, 20, 270, 60
    ivalSM3   FLvalue  "", 70, 20, 270, 100
    ivalSM4   FLvalue  "", 70, 20, 270, 140
    gksliderA, gislidSM1  FLslider "Slider A", -4, 4, 0, 3, ivalSM1, 250, 20, 20, 20
    gksliderB, gislidSM2  FLslider "Slider B", 1, 10, 0, 3, ivalSM2, 250, 20, 20, 60
    gksliderC, gislidSM3  FLslider "Slider C", 0, 1, 0, 3, ivalSM3, 250, 20, 20, 100
    gksliderD, gislidSM4  FLslider "Slider D", 0, 1, 0, 3, ivalSM4, 250, 20, 20, 140
    itext1    FLbox     "store", 1, 1, 14, 50, 25, 355, 15
    itext2    FLbox     "load", 1, 1, 14, 50, 25, 415, 15
    itext3    FLbox     "Group 1", 1, 1, 14, 30, 145, 485, 15
    gksnap, ibuttn1  FLbutton "1", 1, 0, 11, 25, 25, 364, 45, 0, 3, 0, 3, 1
    gksnap, ibuttn2  FLbutton "2", 1, 0, 11, 25, 25, 364, 75, 0, 3, 0, 3, 2
    gksnap, ibuttn3  FLbutton "3", 1, 0, 11, 25, 25, 364, 105, 0, 3, 0, 3, 3
    gksnap, ibuttn4  FLbutton "4", 1, 0, 11, 25, 25, 364, 135, 0, 3, 0, 3, 4
    gkload, ibuttn1  FLbutton "1", 1, 0, 11, 25, 25, 424, 45, 0, 4, 0, 3, 1
    gkload, ibuttn2  FLbutton "2", 1, 0, 11, 25, 25, 424, 75, 0, 4, 0, 3, 2
    gkload, ibuttn3  FLbutton "3", 1, 0, 11, 25, 25, 424, 105, 0, 4, 0, 3, 3
    gkload, ibuttn4  FLbutton "4", 1, 0, 11, 25, 25, 424, 135, 0, 4, 0, 3, 4

  FLcolor 100, 140, 118
  FLsetSnapGroup 1
    ivalSM5   FLvalue  "", 70, 20, 270, 190
    ivalSM6   FLvalue  "", 70, 20, 270, 230
    ivalSM7   FLvalue  "", 70, 20, 270, 270
    ivalSM8   FLvalue  "", 70, 20, 270, 310
    gkknobA, gislidSM5  FLknob "Knob A", -4, 4, 0, 3, ivalSM5, 45, 10, 230
    gkknobB, gislidSM6  FLknob "Knob B", 1, 10, 0, 3, ivalSM6, 45, 75, 230
    gkknobC, gislidSM7  FLknob "Knob C", 0, 1, 0, 3, ivalSM7, 45, 140, 230
    gkknobD, gislidSM8  FLknob "Knob D", 0, 1, 0, 3, ivalSM8, 45, 205, 230
    itext4    FLbox     "store", 1, 1, 14, 50, 25, 355, 185
    itext5    FLbox     "load", 1, 1, 14, 50, 25, 415, 185
    itext6    FLbox     "Group 2", 1, 1, 14, 30, 145, 485, 185
    gksnap, ibuttn1  FLbutton "5", 1, 0, 11, 25, 25, 364, 215, 0, 3, 0, 3, 5
    gksnap, ibuttn2  FLbutton "6", 1, 0, 11, 25, 25, 364, 245, 0, 3, 0, 3, 6
    gksnap, ibuttn3  FLbutton "7", 1, 0, 11, 25, 25, 364, 275, 0, 3, 0, 3, 7
```

```

gksnap, ibuttn4  FLbutton "8", 1, 0, 11, 25, 25, 364, 305, 0, 3, 0, 3, 8
gkload, ibuttn1  FLbutton "5", 1, 0, 11, 25, 25, 424, 215, 0, 4, 0, 3, 5
gkload, ibuttn2  FLbutton "6", 1, 0, 11, 25, 25, 424, 245, 0, 4, 0, 3, 6
gkload, ibuttn3  FLbutton "7", 1, 0, 11, 25, 25, 424, 275, 0, 4, 0, 3, 7
gkload, ibuttn4  FLbutton "8", 1, 0, 11, 25, 25, 424, 305, 0, 4, 0, 3, 8
FLpanelEnd
FLsetVal_i 1, gislidSM1
FLsetVal_i 1, gislidSM2
FLsetVal_i 0, gislidSM3
FLsetVal_i 0, gislidSM4
FLsetVal_i 1, gislidSM5
FLsetVal_i 1, gislidSM6
FLsetVal_i 0, gislidSM7
FLsetVal_i 0, gislidSM8
FLrun

instr 1

endin

instr 3 ; Save snapshot
index init 0
ipstno = p4
igroup = 0
Sfile sprintf "PVCsynth.%d.snap", ipstno
if ipstno > 4 then
    igroup = 1
endif

    inumsnap, inumval  FLsetsnap index , -1, igroup
    FLsavesnap Sfile

endin

instr 4 ;Load snapshot
index init 0
ipstno = p4
igroup = 0
Sfile sprintf "PVCsynth.%d.snap", ipstno
if ipstno > 4 then
    igroup = 1
endif

    FLloadsnap Sfile
    inumload  FLgetsnap index , igroup

endin

</CsInstruments>

<CsScore>
    ;Dummy table for FLgetsnap
    ; f 1 0 1024 10 1
    f 0 3600

e

</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

*FLgetsnap, FLloadsnap, FLsetSnapGroup, FLrun, FLsetsnap, FLupdate*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLscroll

FLscroll — Un opcode FLTK qui ajoute des barres d'ascenseur à une zone.

## Description

*FLscroll* ajoute des barres d'ascenseur à une zone.

## Syntaxe

```
FLscroll iwidth, iheight [, ix] [, iy]
```

## Initialisation

*iwidth* -- largeur widget.

*iheight* -- hauteur du widget.

*ix* (facultatif) -- position horizontale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* (facultatif) -- position verticale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

## Exécution

Les conteneurs sont utiles pour formater l'apparence graphiques des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valuateurs ou d'autres sortes de widgets.

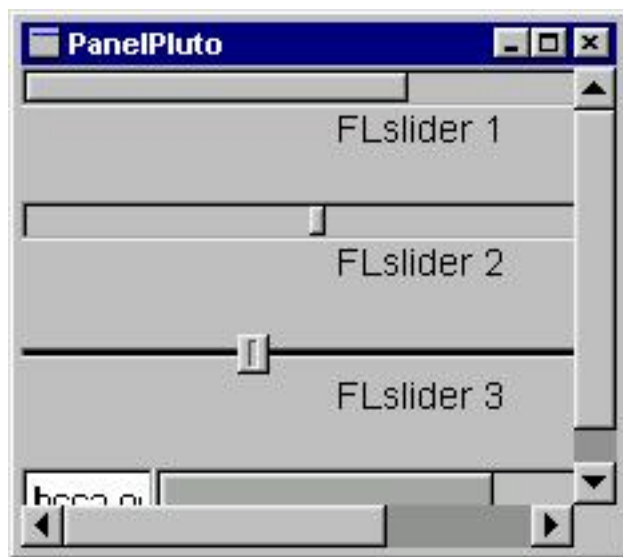
Il n'y a pas d'arguments de taux-k dans les conteneurs.

*FLscroll* ajoute des barres d'ascenseur à une zone. Normalement il faut fixer les arguments *iwidth* et *iheight* à la même valeur que ceux de la fenêtre parente ou d'un autre conteneur parent. *ix* et *iy* sont facultatifs car ils sont normalement fixés à zéro. Par exemple le code suivant :

```
FLpanel      "PanelPluto", 400, 300, 100, 100
FLscroll     400, 300
gk1, ih1 FLslider "FLslider 1", 500, 1000, 2 ,1, -1, 300,15, 20,50
gk2, ih2 FLslider "FLslider 2", 300, 5000, 2 ,3, -1, 300,15, 20,100
gk3, ih3 FLslider "FLslider 3", 350, 1000, 2 ,5, -1, 300,15, 20,150
gk4, ih4 FLslider "FLslider 4", 250, 5000, 1 ,11,-1, 300,30, 20,200
FLscrollEnd
FLpanelEnd
```

montrera des barres d'ascenseur quand la taille de la fenêtre principale sera diminuée.





FLscroll.

## Exemples

Voici un exemple de l'opcode FLscroll. Il utilise le fichier *FLscroll.csd* [examples/FLscroll.csd].

### Exemple 311. Exemple de l'opcode FLscroll.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc    -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLscroll.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Demonstration of the flscroll opcode which enables
; the use of widget sizes and placings beyond the
; dimensions of the containing panel
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Text Box", 420, 200, 50, 50
  iwidth = 420
  iheight = 200
  ix = 0
  iy = 0
  FLscroll iwidth, iheight, ix, iy
  ih3 FLbox "DRAG THE SCROLL BAR TO THE RIGHT IN ORDER TO READ THE REST OF THIS TEXT!", 1, 10, 20, 87
  FLscrollEnd
; End of panel contents
FLpanelEnd
; Run the widget thread!
```

```
FLrun

instr 1
endin

</CsInstruments>
<CsScore>

; 'Dummy' score event of 1 hour.
f 0 3600
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*FLgroup, FLgroupEnd, FLpack, FLpackEnd, FLpanel, FLpanelEnd, FLscrollEnd, FLtabs, FLtabsEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLscrollEnd

FLscrollEnd — Un opcode FLTK qui marque la fin d'une zone avec barres d'ascenseur.

## Description

Un opcode FLTK qui marque la fin d'une zone avec barres d'ascenseur.

## Syntaxe

`FLscrollEnd`

Exécution

## Exécution

Les conteneurs sont utiles pour formater l'apparence graphiques des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valueurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de taux-k dans les conteneurs.

## Voir aussi

*FLgroup*, *FLgroupEnd*, *FLpack*, *FLpackEnd*, *FLpanel*, *FLpanelEnd*, *FLscroll*, *FLtabs*, *FLtabsEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLscroll\_end

FLscroll\_end — Un opcode FLTK qui marque la fin d'une zone avec barres d'ascenseur.

## Description

Un opcode FLTK qui marque la fin d'une zone avec barres d'ascenseur. C'est un autre nom pour **FLscrollEnd** fourni pour des raisons de compatibilité. Voir *FLscrollEnd*.

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetAlign

FLsetAlign — Fixe l'alignement du texte de l'étiquette d'un widget FLTK.

## Description

*FLsetAlign* fixe l'alignement du texte de l'étiquette d'un widget FLTK.

## Syntaxe

```
FLsetAlign ialign, ihandle
```

## Initialisation

*ialign* -- fixe l'alignement du texte de l'étiquette d'un widget.

Les valeurs admises pour l'argument *ialign* sont :

- 1 - centré
- 2 - en haut
- 3 - en bas
- 4 - à gauche
- 5 - à droite
- 6 - en haut à gauche
- 7 - en haut à droite
- 8 - en bas à gauche
- 9 - en bas à droite

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Voir aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetBox

FLsetBox — Fixe l'apparence d'une boîte entourant un widget FLTK.

## Description

*FLsetBox* fixe l'apparence d'une boîte entourant le widget cible.

## Syntaxe

```
FLsetBox itype, ihandle
```

## Initialisation

*itype* -- un nombre entier qui modifie l'apparence du widget cible.

Les valeurs admises pour l'argument *itype* sont :

- 1 - boîte sans relief
- 2 - boîte saillante
- 3 - boîte en creux
- 4 - boîte légèrement saillante
- 5 - boîte légèrement en creux
- 6 - boîte gravée
- 7 - boîte en relief
- 8 - boîte avec cadre
- 9 - boîte ombrée
- 10 - boîte arrondie
- 11 - boîte arrondie ombrée
- 12 - boîte arrondie sans relief
- 13 - boîte arrondie saillante
- 14 - boîte arrondie creuse
- 15 - boîte en losange saillante
- 16 - boîte en losange en creux
- 17 - boîte ovale
- 18 - boîte ovale ombrée
- 19 - boîte ovale sans relief

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Voir aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetColor

FLsetColor — Fixe la couleur d'un widget FLTK.

## Description

*FLsetColor* fixe la couleur du widget cible.

## Syntaxe

```
FLsetColor ired, igreen, iblue, ihandle
```

## Initialisation

*ired* -- La composante rouge de la couleur du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*igreen* -- La composante verte de la couleur du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*iblue* -- La composante bleue de la couleur du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Exemples

Voici un exemple de l'opcode FLsetcolor. Il utilise le fichier *FLsetcolor.csd* [examples/FLsetcolor.csd].

### Exemple 312. Exemple de l'opcode FLsetcolor.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d           ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o FLsetcolor.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Using the opcode flsetcolor to change from the
; default colours for widgets
sr = 44100
kr = 441
ksmps = 100
nchnls = 1
```



```

FLpanel "Coloured Sliders", 900, 360, 50, 50
    gkfreq, ihandle FLslider "A Red Slider", 200, 5000, -1, 5, -1, 750, 30, 85, 50
    ired1 = 255
    igreen1 = 0
    iblue1 = 0
    FLsetColor ired1, igreen1, iblue1, ihandle

    gkfreq, ihandle FLslider "A Green Slider", 200, 5000, -1, 5, -1, 750, 30, 85, 150
    ired1 = 0
    igreen1 = 255
    iblue1 = 0
    FLsetColor ired1, igreen1, iblue1, ihandle

    gkfreq, ihandle FLslider "A Blue Slider", 200, 5000, -1, 5, -1, 750, 30, 85, 250
    ired1 = 0
    igreen1 = 0
    iblue1 = 255
    FLsetColor ired1, igreen1, iblue1, ihandle
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
endin

</CsInstruments>
<CsScore>

; 'Dummy' score event for 1 hour.
f 0 3600
e

</CsScore>
</CsSoundSynthesizer>

```

## Voir aussi

*FLcolor2, FLhide, FLlabel, FLsetAlign, FLsetBox, FLsetColor, FLsetColor2, FLsetFont, FLsetPosition, FLsetSize, FLsetText, FLsetTextColor, FLsetTextSize, FLsetTextType, FLsetVal\_i, FLsetVal, FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLsetColor2

FLsetColor2 — Fixe la couleur de sélection d'un widget FLTK.

## Description

*FLsetColor2* fixe la couleur de sélection du widget cible.

## Syntaxe

```
FLsetColor2 ired, igreen, iblue, ihandle
```

## Initialisation

*ired* -- La composante rouge de la couleur du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*igreen* -- La composante verte de la couleur du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*iblue* -- La composante bleue de la couleur du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Voir aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetFont

FLsetFont — Fixe le type de la police d'un widget FLTK.

## Description

*FLsetFont* fixe le type de la police du widget cible.

## Syntaxe

```
FLsetFont ifont, ihandle
```

## Initialisation

*ifont* -- fixe le type de la police de l'étiquette d'un widget.

Les valeurs admises pour l'argument *ifont* sont :

- 1 - helvetica (comme "Arial" sous Windows)
- 2 - helvetica gras
- 3 - helvetica italique
- 4 - helvetica gras italique
- 5 - courrier
- 6 - courrier gras
- 7 - courrier italique
- 8 - courrier gras italique
- 9 - times
- 10 - times gras
- 11 - times italique
- 12 - times gras italique
- 13 - symbol
- 14 - screen
- 15 - screen gras
- 16 - dingbats

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Voir aussi

*FLcolor2, FLhide, FLlabel, FLsetAlign, FLsetBox, FLsetColor, FLsetColor2, FLsetFont, FLsetPosition, FLsetSize, FLsetText, FLsetTextColor, FLsetTextSize, FLsetTextType, FLsetVal\_i, FLsetVal, FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetPosition

FLsetPosition — Fixe la position d'un widget FLTK.

## Description

*FLsetPosition* fixe la position du widget cible en fonction des arguments *ix* et *iy*.

## Syntaxe

```
FLsetPosition ix, iy, ihandle
```

## Initialisation

*ix* -- position horizontale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du widget, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Voir aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetSize

FLsetSize — Redimensionne un widget FLTK.

## Description

*FLsetSize* redimensionne le widget cible (pas la taille de son texte) en fonction des arguments *iwidth* et *iheight*.

## Syntaxe

```
FLsetSize iwidth, iheight, ihandle
```

## Initialisation

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Voir aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetsnap

FLsetsnap — Enregistre l'état courant de tous les valueurs FLTK dans un instantané.

## Description

*FLsetsnap* enregistre l'état courant de tous les valueurs présents dans l'orchestre dans un instantané (en mémoire).

## Syntaxe

```
inumsnap, inumval FLsetsnap index [, ifn, igroup]
```

## Initialisation

*inumsnap* -- nombre courant d'instantanés.

*inumval* -- nombre de valueurs (dont la valeur est enregistrée dans l'instantané) présents dans l'orchestre courant.

*index* -- un nombre faisant référence de manière univoque à un instantané. Plusieurs instantanés peuvent être enregistrés dans la même banque.

*ifn* (optional) -- optional argument referring to an already allocated table, to store values of a snapshot.

*igroup* -- (facultatif) un nombre entier faisant référence à un groupe de widgets en relation avec un instantané. Cela permet de lire/écrire, ou charger/sauvegarder l'état d'un sous-ensemble de valueurs. La valeur par défaut est zéro qui fait référence au premier groupe. Le numéro de groupe est déterminé par l'opcode *FLsetSnapGroup*.



### Note

Le paramètre *igroup* n'a pas encore été complètement implémenté dans la version actuelle de Csound. Prière de ne pas s'y fier.

## Exécution

L'opcode *FLsetsnap* enregistre l'état courant de tous les valueurs dans l'orchestre dans un instantané (en mémoire). On peut stocker n'importe quel nombre d'instantanés dans la banque courante. Les banques sont des structures qui n'existent qu'en mémoire, sans autre référence que le fait qu'on peut y accéder par les opcodes *FLsetsnap*, *FLsavesnap*, *FLloadsnap* et *FLgetsnap*. Il ne peut y avoir qu'une seule banque présente en mémoire.

Si l'argument facultatif *ifn* fait référence à une table valide déjà allouée, l'instantané sera enregistré dans la table plutôt que dans la banque. Ainsi cette table est accessible depuis d'autres opcodes de Csound.

L'argument *index* fait référence à un instantané déterminé de manière univoque. Si la valeur d'*index* fait référence à un instantané antérieurement sauvegardé, toutes ses anciennes valeurs seront remplacées par les valeurs courantes. Si *index* fait référence à un instantané qui n'existe pas, un nouvel instantané sera créé. Si la valeur d'*index* n'est pas adjacente à celle d'un instantané déjà créé, des instantanés vides seront créés. Par exemple, si la position d'*index* 0 contient le seul instantané présent dans une banque et que l'on sauvegarde

un nouvel instantané avec l'*index* 5, toutes les positions entre 1 et 4 contiendront automatiquement des instantanés vides. Les instantanés vides ne contiennent pas de données et sont neutres.

*FLsetsnap* retourne le nombre courant d'instantanés (l'argument *inumsnap*) et le nombre total de valeurs stockées dans chaque instantané (*inumval*). *inumval* est égal au nombre de valueurs présents dans l'orchestre.

Pour économiser la mémoire, les widgets peuvent être groupés afin que les instantanés n'affectent qu'un groupe défini de widgets. L'opcode *FLsetSnapGroup* est utilisé pour spécifier le groupe de tous les widgets déclarés après lui jusqu'à la prochaine instruction *FLsetSnapGroup*.

## Voir aussi

*FLgetsnap*, *FLloadsnap*, *FLsetSnapGroup*, *FLrun*, *FLsavesnap*, *FLupdate*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22



# FLsetSnapGroup

FLsetSnapGroup — Détermine le groupe d'instantané pour les valuateurs FLTK.

## Description

*FLsetSnapGroup* détermine le groupe d'instantané des valuateurs déclarés après lui.

## Syntaxe

**FLsetSnapGroup** *igroup*

## Initialisation

*igroup* -- (facultatif) un nombre entier faisant référence à un groupe de widgets en relation avec un instantané. Cela permet de lire/écrire, ou charger/sauvegarder l'état d'un sous-ensemble de valuateurs.



### Note

Le paramètre *igroup* n'a pas encore été complètement implémenté dans la version actuelle de Csound. Prière de ne pas s'y fier.

## Exécution

Pour économiser la mémoire, les widgets peuvent être groupés afin que les instantanés n'affectent qu'un groupe défini de widgets. L'opcode *FLsetSnapGroup* est utilisé pour spécifier le groupe de tous les widgets déclarés après lui jusqu'à la prochaine instruction *FLsetSnapGroup*.

*FLsetSnapGroup* détermine le groupe d'instantané d'un valuateur déclaré. Pour qu'un valuateur appartienne à un groupe fixé, il faut placer *FLsetSnapGroup* juste avant la déclaration du widget lui-même. Le groupe fixé par *FLsetSnapGroup* est valable pour tous les valuateurs déclarés après lui, jusqu'à ce qu'une nouvelle instruction *FLsetSnapGroup* avec un groupe différent soit rencontrée. Si aucune instruction *FLsetSnapGroup* n'est présente dans l'orchestre, le groupe par défaut pour tous les widgets sera le groupe zéro.

## Voir aussi

*FLgetsnap*, *FLsetsnap*, *FLloadsnap*, *FLsavesnap*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetText

FLsetText — Fixe l'étiquette d'un widget FLTK.

## Description

*FLsetText* met la chaîne de caractères entre guillemets de l'argument *itext* dans l'étiquette du widget cible.

## Syntaxe

```
FLsetText "itext", ihandle
```

```
FLsetText istr, ihandle
```

## Initialisation

« *itext* » -- une chaîne de caractères entre guillemets contenant le texte de l'étiquette du widget.

*istr* -- une valeur de taux-i qui dénote une chaîne de caractères via *strset* désignant le texte de l'étiquette du widget.

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Exemples

Voici un exemple de l'opcode FLsetText. Il utilise le fichier *FLsetText.csd* [exemples/FLsetText.csd].

### Exemple 313. Exemple de l'opcode FLsetText.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLsetText.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2

; Example by Giorgio Zucco and Andres Cabrera 2007

FLpanel "FLsetText",250,100,50,50

gk1,giha FLcount " ", 1, 20, 1, 20, 1, 200, 40, 20, 20, 0, 1, 0, 1
```

```
FLpanelEnd
FLrun

    instr 1
    ; This instrument is triggered by FLcount above each time
    ; its value changes
    iname = i(gk1)
    print iname
    ; Must use FLsetText on the init pass!
    if (iname == 1) igoto text1
    if (iname == 2) igoto text2
    if (iname == 3) igoto text3

    igoto end

text1:
FLsetText "FM",giha
    igoto end

text2:
FLsetText "GRANUL",giha
    igoto end

text3:
FLsetText "PLUCK",giha
    igoto end

end:
    endin

</CsInstruments>
<CsScore>

f 0 3600

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*FLcolor2, FLhide, FLlabel, FLsetAlign, FLsetBox, FLsetColor, FLsetColor2, FLsetFont, FLsetPosition, FLsetSize, FLsetText, FLsetTextColor, FLsetTextSize, FLsetTextType, FLsetVal\_i, FLsetVal, FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetTextColor

FLsetTextColor — Fixe la couleur du texte de l'étiquette d'un widget FLTK.

## Description

*FLsetTextColor* fixe la couleur du texte de l'étiquette du widget cible.

## Syntaxe

```
FLsetTextColor ired, iblue, igreen, ihandle
```

## Initialisation

*ired* -- La composante rouge de la couleur du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*igreen* -- La composante verte de la couleur du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*iblue* -- La composante bleue de la couleur du widget cible. Chaque composante RVB est comprise entre 0 et 255.

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Voir aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetTextSize

FLsetTextSize — Fixe la taille du texte de l'étiquette d'un widget FLTK.

## Description

*FLsetTextSize* fixe la taille du texte de l'étiquette du widget cible.

## Syntaxe

```
FLsetTextSize isize, ihandle
```

## Initialisation

*isize* -- taille de la police du widget cible. Les valeurs normales sont de l'ordre de 15. Des nombres plus grands augmentent la taille de la police, tandis que des nombres plus petits la réduisent.

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Voir aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetTextType

FLsetTextType — Fixe quelques attributs de la police du texte de l'étiquette d'un widget FLTK.

## Description

*FLsetTextType* fixe quelques attributs de la police du texte de l'étiquette du widget cible.

## Syntaxe

```
FLsetTextType itype, ihandle
```

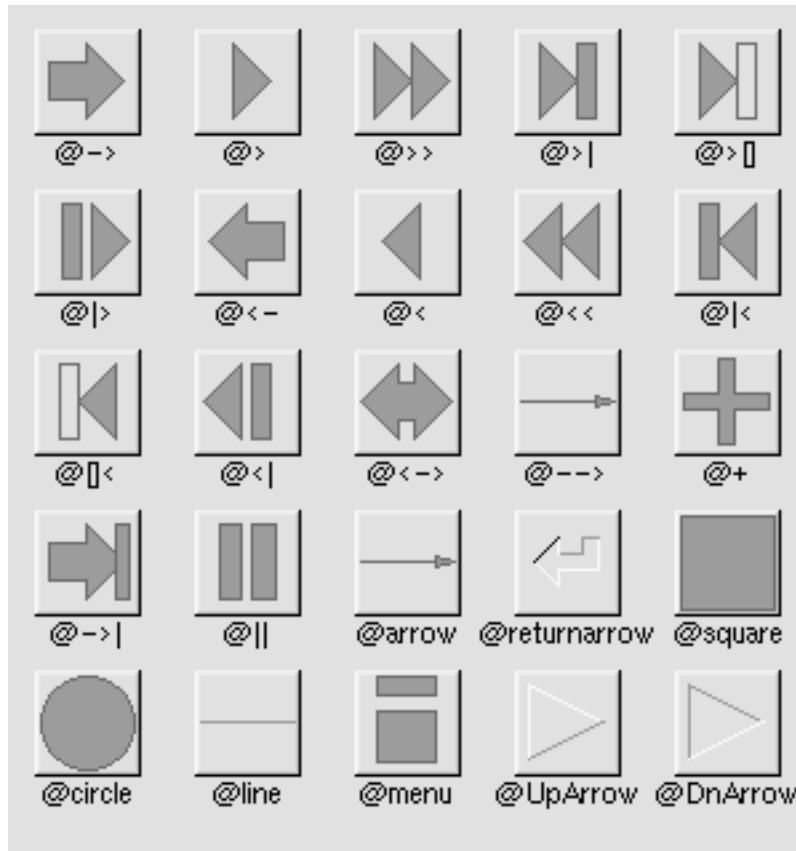
## Initialisation

*itype* -- un nombre entier qui modifie l'apparence du widget cible.

Les valeurs admises pour l'argument *itype* sont :

- 0 - étiquette normale
- 1 - pas d'étiquette (le texte est caché)
- 2 - étiquette pictogramme (voir ci-dessous)
- 3 - étiquette ombrée
- 4 - étiquette gravée
- 5 - étiquette en relief
- 6 - étiquette bitmap (pas encore implémenté)
- 7 - étiquette pixmap (pas encore implémenté)
- 8 - étiquette image (pas encore implémenté)
- 9 - étiquette multiple (pas encore implémenté)
- 10 - étiquette de type libre (pas encore implémenté)

Lorsque l'on utilise *itype*=3 (étiquette pictogramme), il est possible d'affecter un symbole graphique à la place du texte de l'étiquette du widget cible. Dans ce cas, la chaîne de caractères de l'étiquette cible doit toujours commencer par un « @ ». Si elle commence avec un autre caractère (ou que le symbole n'est pas trouvé), l'étiquette est dessinée normalement. Les symboles suivants sont supportés :



Symboles d'étiquette FLTK supportés.

Le signe @ peut être suivi par les caractères de « formatage » facultatifs suivants, dans cet ordre :

1. « # »force une image carrée sans distortion de la forme du widget.
2. +[1-9] or -[1-9] grossit ou diminue l'image.
3. [1-9] effectue une rotation d'un multiple de 45 degrés. « 6 » ne fait rien, les autres valeurs pointent dans la direction de cette touche sur un pavé numérique.

Noter qu'avec *FLbox* et *FLbutton* il n'est pas nécessaire d'appeler l'opcode *FLsetTextType* pour utiliser un symbole. Dans ce cas, il suffit d'utiliser une étiquette commençant par « @ » suivi de la chaîne de formatage correcte.

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Voir aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22



# FLsetVal\_i

FLsetVal\_i — Met un nombre fourni par l'utilisateur dans la valeur d'un valuateur FLTK.

## Description

*FLsetVal\_i* force la valeur d'un valuateur à un nombre fourni par l'utilisateur.

## Syntaxe

```
FLsetVal_i ivalue, ihandle
```

## Initialisation

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Exécution

*ivalue* -- Valeur à attribuer au widget.



### Note

*FLsetVal* n'est pas complètement implémenté, et il peut planter dans certains cas (par exemple en fixant la valeur d'un widget connecté au widget *FLvalue* - dans ce cas utiliser deux *FLsetVal\_i* séparés).

## Voir aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLsetVal

FLsetVal — Fixe la valeur d'un valuateur FLTK au taux de contrôle.

## Description

*FLsetVal* est presque identique à *FLsetVal\_i*. Sauf qu'il opère au taux-k et qu'il n'affecte le valuateur cible que lorsque *ktrig* est fixé à une valeur différente de zéro.

## Syntaxe

```
FLsetVal ktrig, kvalue, ihandle
```

## Initialisation

*ihandle* -- un nombre entier (utilisé comme identifiant unique) pris de la sortie d'un opcode de widget déjà en place (qui correspond au widget cible). Il est utilisé pour identifier de manière univoque le widget lors de la modification de son apparence par cette classe d'opcodes. Il ne faut pas fixer la valeur de *ihandle* directement sous peine de provoquer un plantage de Csound.

## Exécution

*ktrig* -- déclenche l'opcode lorsqu'il est différent de 0.

*kvalue* -- Valeur à attribuer au widget.



### Note

*FLsetVal* n'est pas complètement implémenté, et il peut planter dans certains cas (par exemple en fixant la valeur d'un widget connecté au widget *FLvalue* - dans ce cas utiliser deux *FLsetVal* séparés).

## Voir aussi

*FLcolor*, *FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLshow

FLshow — Rend visible un widget FLTK antérieurement caché.

## Description

*FLshow* rend visible un widget antérieurement caché.

## Syntaxe

```
FLshow ihandle
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*).

## Voir aussi

*FLcolor2*, *FLhide*, *FLlabel*, *FLsetAlign*, *FLsetBox*, *FLsetColor*, *FLsetColor2*, *FLsetFont*, *FLsetPosition*, *FLsetSize*, *FLsetText*, *FLsetTextColor*, *FLsetTextSize*, *FLsetTextType*, *FLsetVal\_i*, *FLsetVal*, *FLshow*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLslidBnk

FLslidBnk — Un widget FLTK contenant un banc de réglettes horizontales.

## Description

*FLslidBnk* est un widget FLTK contenant un banc de réglettes horizontales.

## Syntaxe

```
FLslidBnk "names", inumsliders [, ioutable] [, iwidth] [, iheight] [, ix] \
    [, iy] [, itypetable] [, iexptable] [, istart_index] [, iminmaxtable]
```

## Initialisation

« *names* » -- une chaîne de caractères entre guillemets contenant le nom de chaque réglette. Chaque réglette peut avoir un nom différent. Chaque nom est séparé par un caractère « @ », par exemple : « fréquence@amplitude@coupure ». Il est possible de ne fournir aucun nom en donnant un seul espace « ». Dans ce cas, l'opcode affectera automatiquement un numéro en progression ascendante comme étiquette pour chaque réglette.

*inumsliders* -- le nombre de réglettes.

*ioutable* (facultatif, 0 par défaut) -- numéro d'une table allouée préalablement dans laquelle seront stockées les valeurs de sortie de chaque réglette. Il faut s'assurer que la taille de la table est suffisante pour contenir toutes les cellules de sortie, sinon Csound plantera avec une erreur de segmentation. En affectant zéro à cet argument, la sortie sera dirigée vers l'espace zak dans la zone de taux-k. Dans ce cas, l'espace zak doit avoir été alloué au préalable avec l'opcode *zakinit* et il faut s'assurer que la taille d'allocation est suffisante pour couvrir toutes les réglettes. La valeur par défaut est zéro (c'est-à-dire stockage de la sortie dans l'espace zak).

*istart\_index* (facultatif, 0 par défaut) -- un nombre entier indiquant un décalage des positions des cellules de sortie. Il peut être positif pour permettre l'allocation en sortie de plusieurs bancs de réglettes dans la même table ou dans l'espace zak. La valeur par défaut est zéro (pas de décalage).

*iminmaxtable* (facultatif, 0 par défaut) -- numéro d'une table définie au préalable contenant une liste de couples min-max pour chaque réglette. Une valeur de zéro signifie l'intervalle allant de 0 à 1 pour toutes les réglettes, sans fournir de table. La valeur par défaut est zéro.

*iexptable* (facultatif, 0 par défaut) -- numéro d'une table définie au préalable contenant une liste d'identifiants (des nombres entiers) fournis pour modifier le comportement de chaque réglette de manière indépendante. Les identifiants peuvent avoir les valeurs suivantes :

- -1 -- courbe de réponse exponentielle
- 0 -- réponse linéaire
- nombre > 0 -- suit la courbe d'une table définie au préalable pour mettre en forme la réponse de la réglette correspondante. Dans ce cas, ce nombre correspond au numéro de la table.

On peut souhaiter que toutes les réglettes du banc aient la même courbe de réponse (exponentielle ou linéaire). Dans ce cas, on peut affecter -1 ou 0 à *iexptable* sans se préoccuper de définir auparavant une

table. La valeur par défaut est zéro (toutes les réglettes ont une réponse linéaire sans avoir à fournir de table).

*ityetable* (facultatif, 0 par défaut) -- numéro d'une table définie au préalable contenant une liste d'identifiants (des nombres entiers) fournis pour modifier l'aspect de chaque réglette de manière indépendante. Les identifiants peuvent avoir les valeurs suivantes :

- 0 = Réglette stylée
- 1 = Réglette pleine
- 3 = Réglette normale
- 5 = Réglette stylée
- 7 = Réglette stylée en creux

On peut souhaiter que toutes les réglettes du banc aient le même aspect. Dans ce cas, on peut affecter un nombre négatif à *ityetable* sans se préoccuper de définir auparavant une table. Les nombres négatifs ont la même signification que les identifiants positifs correspondants sauf que le même aspect est affecté à toutes les réglettes. On peut aussi donner un aspect aléatoire à chaque réglette en affectant à *ityetable* un nombre négatif inférieur à -7. La valeur par défaut est 0 (toutes les réglettes sont stylées, sans avoir à fournir de table).

On peut ajouter 20 à une valeur dans la table pour donner l'aspect "plastique" à la réglette, ou soustraire 20 si l'on veut affecter la valeur à tous les widgets sans définir une table (par exemple -21 pour donner à toutes les réglettes le type Plastique Plein).

*iwidth* (facultatif) -- largeur de la zone rectangulaire contenant toutes les réglettes du banc, à l'exclusion des étiquettes qui sont placées à la gauche de cette zone.

*iheight* (facultatif) -- hauteur de la zone rectangulaire contenant toutes les réglettes du banc, à l'exclusion des étiquettes qui sont placées à la gauche de cette zone.

*ix* (facultatif) -- position horizontale du coin supérieur gauche de la zone rectangulaire contenant toutes les réglettes appartenant au banc. Il faut laisser suffisamment d'espace à la gauche de ce rectangle afin que les étiquettes des réglettes soient visibles. En effet, les étiquettes elles-mêmes sont situées à l'extérieur de la zone rectangulaire.

*iy* (facultatif) -- position verticale du coin supérieur gauche de la zone rectangulaire contenant toutes les réglettes appartenant au banc. Il faut laisser suffisamment d'espace à la gauche de ce rectangle afin que les étiquettes des réglettes soient visibles. En effet, les étiquettes elles-mêmes sont situées à l'extérieur de la zone rectangulaire.

## Exécution

Il n'y a pas d'argument de taux-k, même si les cellules de la table en sortie (ou l'espace zak) sont mis à jour au taux-k.

*FLslidBnk* est un widget contenant un banc de réglettes horizontales. On peut y mettre n'importe quel nombre de réglettes (argument *inumsliders*). La sortie de toutes les réglettes est stockée dans une table allouée au préalable ou dans l'espace zak (argument *ioutable*). Il est possible de déterminer la première position de la table (ou de l'espace zak) dans lequel stocker la sortie de la première réglette au moyen de l'argument *istart\_index*.

Chaque réglette peut avoir une étiquette individuelle placée à sa gauche. Les étiquettes sont définies par l'argument « *names* ». L'intervalle de sortie de chaque réglette peut être fixé individuellement au moyen

d'une table externe (argument *iminmaxtable*). La courbe de réponse de chaque réglette peut être fixée individuellement, au moyen d'une liste d'identifiants placés dans une table (argument *iexptable*). Il est possible de définir l'aspect de chaque réglette indépendamment ou de donner le même aspect à toutes les réglettes (argument *ityetable*).

Les arguments *iwidth*, *iheight*, *ix* et *iy* déterminent la largeur, la hauteur, les positions horizontale et verticale de la zone rectangulaire contenant les réglettes. Noter que l'étiquette de chaque réglette est placée à sa gauche et n'est pas incluse dans la zone rectangulaire contenant les réglettes. Ainsi l'utilisateur doit laisser assez d'espace à la gauche du banc en affectant une valeur suffisante à *ix* afin que les étiquettes soient visibles.



## IMPORTANT !

Noter que les tables utilisées par *FLslidBnk* doivent être créées avec l'opcode *ftgen* et placées dans l'orchestre avant le valuateur correspondant. On ne peut pas les placer dans la partition. En effet, les tables placées dans la partition sont créées après l'initialisation des opcodes placés dans la section d'en-tête de l'orchestre.

## Exemples

Voici un exemple de l'opode *FLslidBnk*. Il utilise le fichier *FLslidBnk.csd* [exemples/FLslidBnk.csd].

### Exemple 314. Exemple de l'opode *FLslidBnk*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o FLslidBnk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 441
ksmps = 100
nchnls = 1

gityetable ftgen 0, 0, 8, -2, 1, 1, 3, 3, 5, 5, 7, 7
giouttable ftgen 0, 0, 8, -2, 0, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8, 1

FLpanel "Slider Bank", 400, 380, 50, 50
;Number of sliders
inum = 8
; Table to store output
iouttable = giouttable
; Width of the slider bank in pixels
iwidth = 350
; Height of the slider in pixels
iheight = 160
; Distance of the left edge of the slider
; from the left edge of the panel
ix = 30
; Distance of the top edge of the slider
; from the top edge of the panel
iy = 10
```

```

; Table containing fader types
itypetable = gitypetable
FLslidBnk "1@2@3@4@5@6@7@8", inum , iouttable , iwidth , iheight , ix \
, iy , itypetable
FLslidBnk "1@2@3@4@5@6@7@8", inum , iouttable , iwidth , iheight , ix \
, iy + 200 , -23
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
;Dummy instrument
endin

</CsInstruments>
<CsScore>

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*FLslider, FLslidBnk2, FLvslidBnk, FLvslidBnk2*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLslidBnk2

FLslidBnk2 — Un widget FLTK contenant un banc de réglottes horizontales.

## Description

*FLslidBnk2* est un widget FLTK contenant un banc de réglottes horizontales.

## Syntaxe

**FLslidBnk2** "names", inumsliders, ioutable, iconfigtable [,iwidth, iheight, ix, iy, istart\_index]

**FLslidBnk2** istring, inumsliders, ioutable, iconfigtable [,iwidth, iheight, ix, iy, istart\_index]

## Initialisation

« *names* » -- une chaîne de caractères entre guillemets contenant le nom de chaque réglotte. Chaque réglotte peut avoir un nom différent. Chaque nom est séparé par un caractère « @ », par exemple : « fréquence@amplitude@coupure ». Il est possible de ne fournir aucun nom en donnant un seul espace « ». Dans ce cas, l'opcode affectera automatiquement un numéro en progression ascendante comme étiquette pour chaque réglotte.

*istring* -- un entier associé à une chaîne de caractères par *strset*, qui est utilisée comme « *names* » ci-dessus.

*inumsliders* -- le nombre de réglottes.

*ioutable* (facultatif, 0 par défaut) -- numéro d'une table allouée préalablement dans laquelle seront stockées les valeurs de sortie de chaque réglotte. Il faut s'assurer que la taille de la table est suffisante pour contenir toutes les cellules de sortie, sinon Csound plantera avec une erreur de segmentation. En affectant zéro à cet argument, la sortie sera dirigée vers l'espace zak dans la zone de taux-k. Dans ce cas, l'espace zak doit avoir été alloué au préalable avec l'opcode *zakinit* et il faut s'assurer que la taille d'allocation est suffisante pour couvrir toutes les réglottes. La valeur par défaut est zéro (c'est-à-dire stockage de la sortie dans l'espace zak).

*iconfigtable* -- dans les opcodes *FLslidBnk2* et *FLvslidBnk2*, cette table remplace *iminmaxtable*, *iexptable* et *istyletable*, tous ces paramètres étant placés dans une seule table. Cette table doit être remplie avec un groupe de quatre paramètres pour chaque réglotte de la façon suivante :

min1, max1, exp1, style1, min2, max2, exp2, style2, min3, max3, exp3, style3 etc.

par exemple en utilisant GEN02 on peut taper :

*inum fngen* 1,0,256, -2, 0,1,0,1, 100, 5000, -1, 3, 50, 200, -1, 5,..... [etc]

Dans cet exemple la première réglotte reçoit les paramètres [0, 1, 0, 1] (valeurs comprises entre 0 et 1, réponse linéaire, aspect réglotte pleine), la seconde réglotte reçoit les paramètres [100, 5000, -1, 3] (valeurs comprises entre 100 et 5000, réponse exponentielle, aspect réglotte normale), la troisième réglotte reçoit les paramètres [50, 200, -1, 5] (valeurs comprises entre 50 et 200, réponse exponentielle, aspect réglotte stylée), et ainsi de suite.

*iwidth* (facultatif) -- largeur de la zone rectangulaire contenant toutes les réglottes du banc, à l'exclusion des étiquettes qui sont placées à la gauche de cette zone.

*iheight* (facultatif) -- hauteur de la zone rectangulaire contenant toutes les réglottes du banc, à l'exclusion des étiquettes qui sont placées à la gauche de cette zone.



*ix* (facultatif) -- position horizontale du coin supérieur gauche de la zone rectangulaire contenant toutes les réglettes appartenant au banc. Il faut laisser suffisamment d'espace à la gauche de ce rectangle afin que les étiquettes des réglettes soient visibles. En effet, les étiquettes elles-mêmes sont situées à l'extérieur de la zone rectangulaire.

*iy* (facultatif) -- position verticale du coin supérieur gauche de la zone rectangulaire contenant toutes les réglettes appartenant au banc. Il faut laisser suffisamment d'espace à la gauche de ce rectangle afin que les étiquettes des réglettes soient visibles. En effet, les étiquettes elles-mêmes sont situées à l'extérieur de la zone rectangulaire.

*istart\_index* (facultatif, 0 par défaut) -- un nombre entier indiquant un décalage des positions des cellules de sortie. Il peut être positif pour permettre l'allocation en sortie de plusieurs bancs de réglettes dans la même table ou dans l'espace zak. La valeur par défaut est zéro (pas de décalage).

## Exécution

Il n'y a pas d'argument de taux-k, même si les cellules de la table en sortie (ou l'espace zak) sont mis à jour au taux-k.

*FLslidBnk2* est un widget contenant un banc de réglettes horizontales. On peut y mettre n'importe quel nombre de réglettes (argument *inumsliders*). La sortie de toutes les réglettes est stockée dans une table allouée au préalable ou dans l'espace zak (argument *ioutable*). Il est possible de déterminer la première position de la table (ou de l'espace zak) dans laquelle stocker la sortie de la première réglette au moyen de l'argument *istart\_index*.

Chaque réglette peut avoir une étiquette individuelle placée à sa gauche. Les étiquettes sont définies par l'argument « *names* ». L'intervalle de sortie de chaque réglette peut être fixé individuellement au moyen des valeurs *min* et *max* dans la table *iconfigtable*. La courbe de réponse de chaque réglette peut être fixée individuellement, au moyen d'une liste d'identifiants placés dans la table *iconfigtable* (argument *exp*). Il est possible de définir l'aspect de chaque réglette indépendamment ou de donner le même aspect à toutes les réglettes (argument *style* dans la table *iconfigtable*).

Les arguments *iwidth*, *iheight*, *ix* et *iy* déterminent la largeur, la hauteur, les positions horizontale et verticale de la zone rectangulaire contenant les réglettes. Noter que l'étiquette de chaque réglette est placée à sa gauche et n'est pas incluse dans la zone rectangulaire contenant les réglettes. Ainsi l'utilisateur doit laisser assez d'espace à la gauche du banc en affectant une valeur suffisante à *ix* afin que les étiquettes soient visibles.



### IMPORTANT !

Noter que les tables utilisées par *FLslidBnk2* doivent être créées avec l'opcode *ftgen* et placées dans l'orchestre avant le valuateur correspondant. On ne peut pas les placer dans la partition. En effet, les tables placées dans la partition sont créées après l'initialisation des opcodes placés dans la section d'en-tête de l'orchestre.

## Exemples

Voici un exemple de l'opcode *FLslidBnk2*. Il utilise le fichier *FLslidBnk2.csd* [examples/FLslidBnk2.csd].

### Exemple 315. Exemple de l'opcode *FLslidBnk2*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>
```

```

; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc          -M0 ;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 100
nchnls = 2

;Example by Gabriel Maldonado

giElem init 8
giOutTab ftgen 1,0,128, 2, 0

;min1, max1, exp1, type1, min2, max2, exp2, type2, min3, max3, exp3, type3 etc.
giConfigTab ftgen 2,0,128,-2, .1, 1000, -1, 3, .1, 1000, -1, 3, .1, 1000, -1, 3,
.1, 5000, -1, 5, .1, 5000, -1, 5, .1, 5000, -1, 5,
giSine ftgen 3,0,256,10, 1

FLpanel "This Panel contains a Slider Bank",600,600
FLslidBnk2 "mod1@mod2@mod3@amp@freq1@freq2@freq3@freqPo", giElem, giOutTab, giConfigTab, 400, 500, 100
FLpanel_end

FLrun

instr 1

kmodindex1 init 0
kmodindex2 init 0
kmodindex3 init 0
kamp init 0
kfreq1 init 0
kfreq2 init 0
kfreq3 init 0
kfreq4 init 0

vtablelk giOutTab, kmodindex1 , kmodindex2, kmodindex3, kamp, kfreq1, kfreq2 , kfreq3, kfreq4

amod1 oscili kmodindex1, kfreq1, giSine
amod2 oscili kmodindex2, kfreq2, giSine
amod3 oscili kmodindex3, kfreq3, giSine
aout oscili kamp, kfreq4+amod1+amod2+amod3, giSine

outs aout, aout
endin

</CsInstruments>
<CsScore>

i1 0 3600
f0 3600

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*FLslider, FLslidBnk, FLvslidBnk, FLvslidBnk2*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# FLslidBnkGetHandle

FLslidBnkGetHandle — récupère l'identifiant du dernier banc de réglettes créé.

## Description

*FLslidBnkGetHandle* récupère l'identifiant du dernier banc de réglettes créé.

## Syntaxe

```
ihandle FLslidBnkGetHandle
```

## Initialisation

*ihandle* - identifiant du sliderBnk (à utiliser pour fixer ses valeurs).

## Exécution

Il n'y a pas d'argument de taux-k, même si les cellules de la table en sortie (ou l'espace zak) sont mis à jour au taux-k.

*FLslidBnkGetHandle* récupère l'identifiant du dernier banc de réglettes créé. Cet opcode doit suivre immédiatement un opcode *FLslidBnk* (ou *FLvslidBnk*, *FLslidBnk2* et *FLvslidBnk2*), afin de récupérer son identifiant.

Voir l'entrée *FLslidBnk2Setk* pour un exemple d'utilisation.

## Voir aussi

*FLslider*, *FLslidBnk*, *FLslidBnk2*, *FLvslidBnk*, *FLvslidBnk2*, *FLslidBnk2Set*, *FLslidBnk2Setk*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# FLslidBnkSet

FLslidBnkSet — modifie les valeurs d'un banc de réglettes.

## Description

*FLslidBnkSet* modifie les valeurs d'un banc de réglettes selon un ensemble de valeurs stockées dans une table.

## Syntaxe

```
FLslidBnkSet ihandle, ifn [, istartIndex, istartSlid, inumSlid]
```

## Initialisation

*ihandle* - identifiant du sliderBnk (à utiliser pour fixer ses valeurs).

*ifn* - numéro d'une table contenant un ensemble de valeurs à affecter à chaque réglette.

*istartIndex* - (facultatif) indice dans la table du premier élément à être évalué. La valeur par défaut est zéro.

*istartSlid* - (facultatif) première réglette à évaluer. 0 par défaut, indiquant la première réglette.

*inumSlid* - (facultatif) nombre de réglettes à mettre à jour. 0 par défaut, indiquant toutes les réglettes.

## Exécution

*FLslidBnkSet* modifie les valeurs d'un banc de réglettes (créé avec *FLslidBnk* ou avec *FLvslidBnk*) selon un ensemble de valeurs stockées dans la table *ifn*. Il permet actuellement de mettre à jour un banc de réglettes *FLslidBnk* (ou *FLvslidBnk*), (par exemple en utilisant l'opcode *slider8table*) avec un ensemble de valeurs situées dans une table. Il faut mettre dans l'argument *ihandle* l'identifiant reçu de l'opcode *FLslidBnkGetHandle*. Il ne travaille qu'au taux-i. Il est possible de ne réinitialiser qu'une partie des réglettes, en utilisant les arguments facultatifs *istartIndex*, *istartSlid*, *inumSlid*.

Il y a une version de taux-k de cet opcode appelée *FLslidBnkSetk*.

## Voir aussi

*FLslider*, *FLslidBnkGetHandle*, *FLslidBnk*, *FLslidBnk2*, *FLvslidBnk*, *FLvslidBnk2*, *FLslidBnk2Set*, *FLslidBnkSetk*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# FLslidBnkSetk

FLslidBnkSetk — modifie les valeurs d'un banc de réglettes.

## Description

*FLslidBnkSetk* modifie les valeurs d'un banc de réglettes selon un ensemble de valeurs stockées dans une table.

## Syntaxe

```
FLslidBnkSetk ktrig, ihandle, ifn [, istartIndex, istartSlid, inumSlid]
```

## Initialisation

*ihandle* - identifiant du sliderBnk (à utiliser pour fixer ses valeurs).

*ifn* - numéro d'une table contenant un ensemble de valeurs à affecter à chaque réglette.

*istartIndex* - (facultatif) indice dans la table du premier élément à être évalué. La valeur par défaut est zéro.

*istartSlid* - (facultatif) première réglette à évaluer. 0 par défaut, indiquant la première réglette.

*inumSlid* - (facultatif) nombre de réglettes à mettre à jour. 0 par défaut, indiquant toutes les réglettes.

## Exécution

*ktrig* – la sortie de *FLslidBnkSetk* est un déclencheur qui indique si les réglettes doivent être mises à jour ou pas. Une valeur non nulle force la mise à jour des réglettes.

*FLslidBnkSetk* est semblable à *FLslidBnkSet* mais il permet de modifier les valeurs de *FLslidBnk* au taux-*k* (on peut aussi utiliser *FLslidBnkSetk* avec *FLvslidBnk*, obtenant un résultat identique). Il permet aussi de relier le banc de réglettes au MIDI. Si l'on utilise le MIDI (par exemple au moyen de l'opcode *slider8table*), *FLslidBnkSetk* change les valeurs du banc de réglettes *FLslidBnk* avec un ensemble de valeurs situées dans une table. Cet opcode est ainsi capable de servir de pont MIDI vers le widget *FLslidBnk* lorsqu'il est utilisé avec la famille d'opcodes *sliderXXtable* (voir l'entrée *slider8table* pour plus d'information). Noter que, si l'on veut utiliser l'indexation de table comme une courbe de réponse, il est impossible de le faire directement dans la configuration *iconfigtable* de *FLslidBnk2*, lorsque l'on a l'intention d'utiliser l'opcode *FLslidBnkSetk*. En fait, l'élément correspondant de l'élément *inputTable* de *FLslidBnkSetk* doit être positionné en mode linéaire et respecter l'intervalle de 0 à 1. Même les éléments correspondants de *sliderXXtable* doivent être positionnés en mode linéaire dans l'intervalle normalisé. On peut indexer la table plus tard au moyen des opcodes *tab* et *tb*, et recadrer la sortie en fonction des valeurs max et min. D'un autre côté, il est possible d'utiliser une courbe de réponse linéaire ou exponentielle directement, en fixant l'intervalle min-max courant ainsi que l'indicateur à la fois dans l'*iconfigtable* du *FLslidBnk2* correspondant et dans *sliderXXtable*.

*FLslidBnkSetk* est la version de taux-*k* de *FLslidBnk2Set*.

## Voir aussi

*FLslider*, *FLslidBnkGetHandle*, *FLslidBnk*, *FLslidBnk2*, *FLvslidBnk*, *FLvslidBnk2*, *FLslidBnkSet*, *FLslidBnk2Set*, *slider8table*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# FLslidBnk2Set

FLslidBnk2Set — modifie les valeurs d'un banc de réglettes.

## Description

*FLslidBnk2Set* modifie les valeurs d'un banc de réglettes selon un ensemble de valeurs stockées dans une table.

## Syntaxe

```
FLslidBnk2Set ihandle, ifn [, istartIndex, istartSlid, inumSlid]
```

## Initialisation

*ihandle* - identifiant du sliderBnk (à utiliser pour fixer ses valeurs).

*ifn* - numéro d'une table contenant un ensemble de valeurs à affecter à chaque réglette.

*istartIndex* - (facultatif) indice dans la table du premier élément à être évalué. La valeur par défaut est zéro.

*istartSlid* - (facultatif) première réglette à évaluer. 0 par défaut, indiquant la première réglette.

*inumSlid* - (facultatif) nombre de réglettes à mettre à jour. 0 par défaut, indiquant toutes les réglettes.

## Exécution

*FLslidBnk2Set* modifie les valeurs d'un banc de réglettes (créé avec *FLslidBnk2* ou avec *FLvslidBnk2*) selon un ensemble de valeurs stockées dans la table *ifn*. Il permet actuellement de mettre à jour un banc de réglettes *FLslidBnk2* (ou *FLvslidBnk2*), (par exemple en utilisant l'opcode *slider8table*) avec un ensemble de valeurs situées dans une table. Il faut mettre dans l'argument *ihandle* l'identifiant reçu de l'opcode *FLslidBnkGetHandle*. Il ne travaille qu'au taux-i. Il est possible de ne réinitialiser qu'une partie des réglettes, en utilisant les arguments facultatifs *istartIndex*, *istartSlid*, *inumSlid*.

*FLslidBnk2Set* est identique à *FLslidBnkSet*, mais il travaille sur *FLslidBnk2* et *FLvslidBnk2* au lieu de *FLslidBnk* et de *FLvslidBnk*.

Il y a une version de taux-k de cet opcode appelée *FLslidBnk2Setk*.

## Voir aussi

*FLslider*, *FLslidBnkGetHandle*, *FLslidBnk*, *FLslidBnk2*, *FLvslidBnk*, *FLvslidBnk2*, *FLslidBnkSet*, *FLslidBnk2Setk*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06



# FLslidBnk2Setk

FLslidBnk2Setk — modifie les valeurs d'un banc de réglettes.

## Description

*FLslidBnk2Setk* modifie les valeurs d'un banc de réglettes selon un ensemble de valeurs stockées dans une table.

## Syntaxe

```
FLslidBnk2Setk ktrig, ihandle, ifn [, istartIndex, istartSlid, inumSlid]
```

## Initialisation

*ihandle* - identifiant du sliderBnk (à utiliser pour fixer ses valeurs).

*ifn* - numéro d'une table contenant un ensemble de valeurs à affecter à chaque réglette.

*istartIndex* - (facultatif) indice dans la table du premier élément à être évalué. La valeur par défaut est zéro.

*istartSlid* - (facultatif) première réglette à évaluer. 0 par défaut, indiquant la première réglette.

*inumSlid* - (facultatif) nombre de réglettes à mettre à jour. 0 par défaut, indiquant toutes les réglettes.

## Exécution

*ktrig* – la sortie de *FLslidBnk2Setk* est un déclencheur qui indique si les réglettes doivent être mises à jour ou pas. Une valeur non nulle force la mise à jour des réglettes.

*FLslidBnk2Setk* est semblable à *FLslidBnkSet* mais il permet de modifier les valeurs de *FLslidBnk2* au taux-k (on peut aussi utiliser *FLslidBnk2Setk* avec *FLvslidBnk2*, obtenant un résultat identique). Il permet aussi de relier le banc de réglettes au MIDI. Si l'on utilise le MIDI (par exemple au moyen de l'opcode *slider8table*), *FLslidBnk2Setk* change les valeurs du banc de réglettes *FLslidBnk2* avec un ensemble de valeurs situées dans une table. Cet opcode est ainsi capable de servir de pont MIDI vers le widget *FLslidBnk2* lorsqu'il est utilisé avec la famille d'opcodes *sliderXXtable* (voir l'entrée *slider8table* pour plus d'information). Noter que, si l'on veut utiliser l'indexation de table comme une courbe de réponse, il est impossible de le faire directement dans la configuration *iconfigtable* de *FLslidBnk2*, lorsque l'on a l'intention d'utiliser l'opcode *FLslidBnk2Setk*. En fait, l'élément correspondant de l'élément *inputTable* de *FLslidBnk2Setk* doit être positionné en mode linéaire et respecter l'intervalle de 0 à 1. Même les éléments correspondants de *sliderXXtable* doivent être positionnés en mode linéaire dans l'intervalle normalisé. On peut indexer la table plus tard au moyen des opcodes *tab* et *tb*, et recadrer la sortie en fonction des valeurs max et min. D'un autre côté, il est possible d'utiliser une courbe de réponse linéaire ou exponentielle directement, en fixant l'intervalle min-max courant ainsi que l'indicateur à la fois dans l'*iconfigtable* du *FLslidBnk2* correspondant et dans *sliderXXtable*.

*FLslidBnk2Setk* est la version de taux-k de *FLslidBnk2Set*.

## Exemples

Voici un exemple de l'opcode *FLslidBnk2Setk*. Il utilise le fichier *FLslidBnk2Setk.csd* [exemples/FLslidBnk2Setk.csd].

### Exemple 316. Exemple de l'opcode FLslidBnk2Setk.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Audio out   Audio in   No messages
-odac         -iadc      -d          ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 10
nchnls  = 2

;Example by Gabriel Maldonado 2007
giElem  init 8
giOutTab ftgen 1,0,128, 2, 0
giSine   ftgen 3,0,256,10, 1
giOutTab2 ftgen 4,0,128, 2, 0
itab     ftgen 29, 0, 129, 5, .002, 128, 1 ;** exponential ascending curve for slider mapping
giExpTab ftgen 30, 0, 129, -24, itab, 0, 1 ;** rescaled curve for slider mapping
giConfigTab ftgen 2,0,128,-2,          1, 500, -1, 13, \
          1, 500, -1, 13, \
          1, 500, -1, 13, \
          1, 5000, -1, 13, \
\
          1, 1000, -1, 5, \
          1, 1000, -1, 5, \
          1, 1000, -1, 5, \
          1, 5000, -1, 5

FLpanel "Multiple FM",600,600
FLslidBnk2 "mod1@mod2@mod3@amp@freq1@freq2@freq3@freqPo", giElem, giOutTab2, giConfigTab, 400, 500, 10
giHandle FLslidBnkGetHandle
FLpanel_end

FLrun
instr 1
ktrig slider8table 1, giOutTab, 0, \ ; ctl min max init func
27, 1, 500, 3, -1, \ ;1 repeat rate
28, 1, 500, 4, -1, \ ;2 random freq. amount
29, 1, 500, 1, -1, \ ;3 random amp. amount
30, 1, 5000, 1, -1, \ ;4 number of concurrent loop points
\
31, 1, 1000, 1, -1, \;5 kloop1
32, 1, 1000, 1, -1, \;6 kloop2
33, 1, 1000, 1, -1, \;7 kloop3
34, 1, 1000, 1, -1 ;8 kloop4
kmodindex1 init 0
kmodindex2 init 0
kmodindex3 init 0
kamp init 0
kfreq1 init 0
kfreq2 init 0
kfreq3 init 0
kfreq4 init 0
vtablelk giOutTab2, kmodindex1, kmodindex2, kmodindex3, kamp, kfreq1, kfreq2, kfreq3, kfreq4
; *kflag, *ihandle, *ifn, *startInd, *startSlid, *numSlid;
FLslidBnk2Setk ktrig, giHandle, giOutTab, 0, 0, giElem
printk2 kmodindex1
printk2 kmodindex2,10
printk2 kmodindex3,20
printk2 kamp,30
amod1 oscili kmodindex1, kfreq1, giSine
```

```
amod2 oscili kmodindex2, kfreq2, giSine
amod3 oscili kmodindex3, kfreq3, giSine
aout oscili kamp,          kfreq4+amod1+amod2+amod3, giSine
outs aout, aout
endin
</CsInstruments>

<CsScore>
i1 0 3600
</CsScore>

</CsoundSynthesizer>
```

## Voir aussi

*FLslider, FLslidBnkGetHandle, FLslidBnk, FLslidBnk2, FLvslidBnk, FLvslidBnk2 FLslidBnkSet, FLslidBnk2Set, slider8table*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# FLslider

FLslider — Dispose une réglette dans le conteneur FLTK correspondant.

## Description

*FLslider* dispose une réglette dans le conteneur correspondant.

## Syntaxe

```
kout, ihandle FLslider "label", imin, imax, iexp, itype, idisp, iwidth, \
    iheight, ix, iy
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*). Il est automatiquement retourné par *FLslider* et ne doit pas être fixé par l'étiquette de l'utilisateur. (L'étiquette de l'utilisateur est une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.)

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*imin* -- valeur minimale de l'intervalle de sortie (correspond à la valeur à gauche pour les réglettes horizontales et à la valeur du haut pour les réglettes verticales).

*imax* -- valeur maximale de l'intervalle de sortie (correspond à la valeur à droite pour les réglettes horizontales et à la valeur du bas pour les réglettes verticales).

L'argument *imin* peut être supérieur à l'argument *imax*. Cela a pour effet d'« inverser » l'objet si bien que les valeurs supérieures sont dans la direction opposée. L'extrémité remplie des réglettes pleines est aussi inversée.

*iexp* -- un nombre entier indiquant le comportement du valuateur :

- 0 = la sortie est linéaire
- -1 = la sortie est exponentielle

Tout autre nombre positif pour *iexp* indique le numéro d'une table existante lue par indexation avec interpolation linéaire. Un numéro de table négatif supprime l'interpolation.



### IMPORTANT !

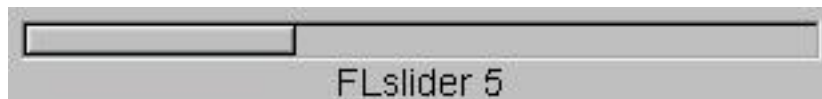
Noter que les tables utilisées par les valuateurs doivent être créées avec l'opcode *ftgen* et placées dans l'orchestre avant le valuateur correspondant. On ne peut pas les placer dans la partition. En effet, les tables placées dans la partition sont créées après l'initialisation des opcodes placés dans la section d'en-tête de l'orchestre.

*itype* -- un nombre entier indiquant l'apparence du valuateur.

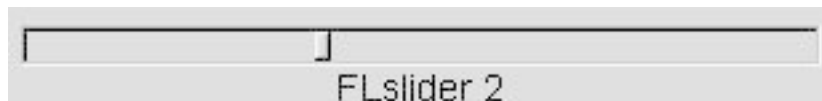
L'argument *itype* accepte les valeurs suivantes :

- 1 - une réglette horizontale pleine

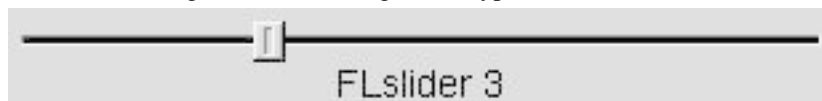
- 2 - une réglette verticale pleine
- 3 - une réglette horizontale gravée
- 4 - une réglette verticale gravée
- 5 - une réglette horizontale stylée
- 6 - une réglette verticale stylée
- 7 - une réglette horizontale stylée saillante
- 8 - une réglette verticale stylée saillante



FLslider - une réglette horizontale pleiner (itype=1).



FLslider - une réglette horizontale gravée (itype=3).



FLslider - une réglette horizontale stylée (itype=5).

On peut aussi créer des réglettes à l'aspect "plastique" en ajoutant 20 à *itype*.

*idisp* -- un identifiant retourné par une instance précédente de l'opcode *FLvalue* pour afficher la valeur courante du valuateur dans le widget *FLvalue*. Si l'on ne veut pas utiliser cette possibilité d'affichage des valeurs courantes, il faut donner à cet identifiant un nombre négatif.

*iwidth* -- largeur du widget.

*iheight* -- largeur du widget.

*ix* -- position horizontale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

## Exécution

*kout* -- valeur en sortie.

Les réglettes sont créées avec la valeur minimale située par défaut à gauche/en haut. Si l'on veut inverser la réglette, il faut inverser les valeurs. Voir l'exemple ci-dessous.

## Exemples

Voici un exemple de l'opcode FLslider. Il utilise le fichier *FLslider.csd* [exemples/FLslider.csd].

**Exemple 317. Exemple de l'opcode FLslider.**

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o FLslider.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; A sine with oscillator with flslider controlled frequency
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Frequency Slider", 900, 400, 50, 50
; Minimum value output by the slider
imin = 200
; Maximum value output by the slider
imax = 5000
; Logarithmic type slider selected
iexp = -1
; Slider graphic type (5='nice' slider)
itype = 5
; Display handle (-1=not used)
idisp = -1
; Width of the slider in pixels
iwidth = 750
; Height of the slider in pixels
iheight = 30
; Distance of the left edge of the slider
; from the left edge of the panel
ix = 125
; Distance of the top edge of the slider
; from the top edge of the panel
iy = 50

gkfreq, ihandle FLslider "Frequency", imin, imax, iexp, itype, idisp, iwidth, iheight, ix, iy
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

;Set the widget's initial value
FLsetVal_i 300, ihandle

instr 1
iamp = 15000
ifn = 1
kfreq portk gkfreq, 0.005 ;Smooth gkfreq to avoid zipper noise
asig oscili iamp, kfreq, ifn
out asig
endin

</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.

```

```
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e
```

```
</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode FLslider, montrant les types de réglettes et d'autres options. Il montre aussi l'utilisation de *FLvalue* pour afficher le contenu d'un widget. Il utilise le fichier *FLslider-2.csd* [exemples/FLslider-2.csd].

### Exemple 318. Exemple plus complexe de l'opcode FLslider.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLslider-2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 441
ksmps = 100
nchnls = 1

;By Andres Cabrera 2007

FLpanel "Slider Types", 410, 260, 50, 50
; Distance of the left edge of the slider
; from the left edge of the panel
ix = 10
; Distance of the top edge of the slider
; from the top edge of the panel
iy = 10
; Create boxes to display widget values
givalue1 FLvalue "1", 60, 20, ix + 330, iy
givalue3 FLvalue "3", 60, 20, ix + 330, iy + 40
givalue5 FLvalue "5", 60, 20, ix + 330, iy + 80

givalue2 FLvalue "2", 60, 20, ix + 60, iy + 140
givalue4 FLvalue "4", 60, 20, ix + 195, iy + 140
givalue6 FLvalue "6", 60, 20, ix + 320, iy + 140

;Horizontal sliders
gkdummy1, gihandle1 FLslider "Type 1", 200, 5000, -1, 1, givalue1, 320, 20, ix, iy
gkdummy3, gihandle3 FLslider "Type 3", 0, 15000, 0, 3, givalue3, 320, 20, ix, iy + 40
; Reversed slider
gkdummy5, gihandle5 FLslider "Type 5", 1, 0, 0, 5, givalue5, 320, 20, ix, iy + 80

;Vertical sliders
gkdummy2, gihandle2 FLslider "Type 2", 0, 1, 0, 2, givalue2, 20, 100, ix+ 30 , iy + 120
; Reversed slider
gkdummy4, gihandle4 FLslider "Type 4", 1, 0, 0, 4, givalue4, 20, 100, ix + 165 , iy + 120
gkdummy6, gihandle6 FLslider "Type 6", 0, 1, 0, 6, givalue6, 20, 100, ix + 290 , iy + 120
FLpanelEnd

FLpanel "Plastic Slider Types", 410, 300, 150, 150
; Distance of the left edge of the slider
; from the left edge of the panel
```

```

ix = 10
; Distance of the top edge of the slider
; from the top edge of the panel
iy = 10
; Create boxes to display widget values
givalue21 FLvalue "21", 60, 20, ix + 330, iy
givalue23 FLvalue "23", 60, 20, ix + 330, iy + 40
givalue25 FLvalue "25", 60, 20, ix + 330, iy + 80

givalue22 FLvalue "22", 60, 20, ix + 60, iy + 140
givalue24 FLvalue "24", 60, 20, ix + 195, iy + 140
givalue26 FLvalue "26", 60, 20, ix + 320, iy + 140

;Horizontal sliders
gkdummy21, gihandle21 FLslider "Type 21", 200, 5000, -1, 21, givalue21, 320, 20, ix, iy
gkdummy23, gihandle23 FLslider "Type 23", 0, 15000, 0, 23, givalue23, 320, 20, ix, iy + 40
; Reversed slider
gkdummy25, gihandle25 FLslider "Type 25", 1, 0, 0, 25, givalue25, 320, 20, ix, iy + 80

;Vertical sliders
gkdummy22, gihandle22 FLslider "Type 22", 0, 1, 0, 22, givalue22, 20, 100, ix+ 30 , iy + 120
; Reversed slider
gkdummy24, gihandle24 FLslider "Type 24", 1, 0, 0, 24, givalue24, 20, 100, ix + 165 , iy + 120
gkdummy26, gihandle26 FLslider "Type 26", 0, 1, 0, 26, givalue26, 20, 100, ix + 290 , iy + 120
;Button to add color to the sliders
gkcolors, ihdummy FLbutton "Color", 1, 0, 21, 150, 30, 30, 260, 0, 10, 0, 1
FLpanelEnd
FLrun

;Set some widget's initial value
FLsetVal_i 500, gihandle1
FLsetVal_i 1000, gihandle3

instr 10
; Set the color of widgets
FLsetColor 200, 230, 0, gihandle1
FLsetColor 0, 123, 100, gihandle2
FLsetColor 180, 23, 12, gihandle3
FLsetColor 10, 230, 0, gihandle4
FLsetColor 0, 0, 0, gihandle5
FLsetColor 0, 0, 0, gihandle6

FLsetColor 200, 230, 0, givalue1
FLsetColor 0, 123, 100, givalue2
FLsetColor 180, 23, 12, givalue3
FLsetColor 10, 230, 0, givalue4
FLsetColor 255, 255, 255, givalue5
FLsetColor 255, 255, 255, givalue6

FLsetColor2 20, 23, 100, gihandle1
FLsetColor2 200,0 ,123 , gihandle2
FLsetColor2 180, 180, 100, gihandle3
FLsetColor2 180, 23, 12, gihandle4
FLsetColor2 180, 180, 100, gihandle5
FLsetColor2 180, 23, 12, gihandle6

FLsetColor 200, 230, 0, gihandle21
FLsetColor 0, 123, 100, gihandle22
FLsetColor 180, 23, 12, gihandle23
FLsetColor 10, 230, 0, gihandle24
FLsetColor 0, 0, 0, gihandle25
FLsetColor 0, 0, 0, gihandle26

FLsetColor 200, 230, 0, givalue21
FLsetColor 0, 123, 100, givalue22

```



```

FLsetColor 180, 23, 12, givalue23
FLsetColor 10, 230, 0, givalue24
FLsetColor 255, 255, 255, givalue25
FLsetColor 255, 255, 255, givalue26

FLsetColor2 20, 23, 100, gihandle21
FLsetColor2 200,0 ,123 , gihandle22
FLsetColor2 180, 180, 100, gihandle23
FLsetColor2 180, 23, 12, gihandle24
FLsetColor2 180, 180, 100, gihandle25
FLsetColor2 180, 23, 12, gihandle26

; Slider values must be updated for colors to change
FLsetVal_i 250, gihandle1
FLsetVal_i 0.5, gihandle2
FLsetVal_i 0, gihandle3
FLsetVal_i 0, gihandle4
FLsetVal_i 0, gihandle5
FLsetVal_i 0.5, gihandle6
FLsetVal_i 250, gihandle21
FLsetVal_i 0.5, gihandle22
FLsetVal_i 500, gihandle23
FLsetVal_i 0, gihandle24
FLsetVal_i 0, gihandle25
FLsetVal_i 0.5, gihandle26

endin

</CsInstruments>
<CsScore>
f 0 3600 ;Dummy table to make csound wait for realtime events

e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*FLcount, FLjoy, FLknob, FLroller, FLslidBnk, FLtext*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Février 2004. Grâce à une note de Dave Phillips, le paramètre étranger *istep* a été effacé.

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLtabs

FLtabs — Crée une interface FLTK à onglets.

## Description

*FLtabs* est une interface à onglets qui est utile pour afficher alternativement plusieurs zones contenant des widgets dans la même fenêtre. Il doit être utilisé en même temps qu'un *FLgroup*, un autre conteneur qui regroupe des widgets enfants.

## Syntaxe

**FLtabs** *iwidth, iheight, ix, iy*

## Initialisation

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du conteneur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

## Exécution

Les conteneurs sont utiles pour formater l'apparence graphique des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valueurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de taux-k dans les conteneurs.

*FLtabs* est une interface à onglets qui est utile pour afficher alternativement plusieurs zones contenant des widgets dans la même fenêtre.



FLtabs.

Il doit être utilisé en même temps qu'un *FLgroup*, un autre opcode de conteneur FLTK qui regroupe des widgets enfants.

## Exemples

Le code de l'exemple suivant :

```

FLpanel "Panel1", 450, 550, 100, 100
FLscroll 450, 550, 0, 0
FLtabs 400, 550, 5, 5

FLgroup "sliders", 380, 500, 10, 40, 1
gk1, ihs FLslider "FLslider 1", 500, 1000, 2 ,1, -1, 300,15, 20,50
gk2, ihs FLslider "FLslider 2", 300, 5000, 2 ,3, -1, 300,15, 20,100
gk3, ihs FLslider "FLslider 3", 350, 1000, 2 ,5, -1, 300,15, 20,150
gk4, ihs FLslider "FLslider 4", 250, 5000, 1 ,11, -1, 300,30, 20,200
gk5, ihs FLslider "FLslider 5", 220, 8000, 2 ,1, -1, 300,15, 20,250
gk6, ihs FLslider "FLslider 6", 1, 5000, 1 ,13, -1, 300,15, 20,300
gk7, ihs FLslider "FLslider 7", 870, 5000, 1 ,15, -1, 300,30, 20,350
gk8, ihs FLslider "FLslider 8", 20, 20000, 2 ,6, -1, 30,400, 350,50
FLgroupEnd

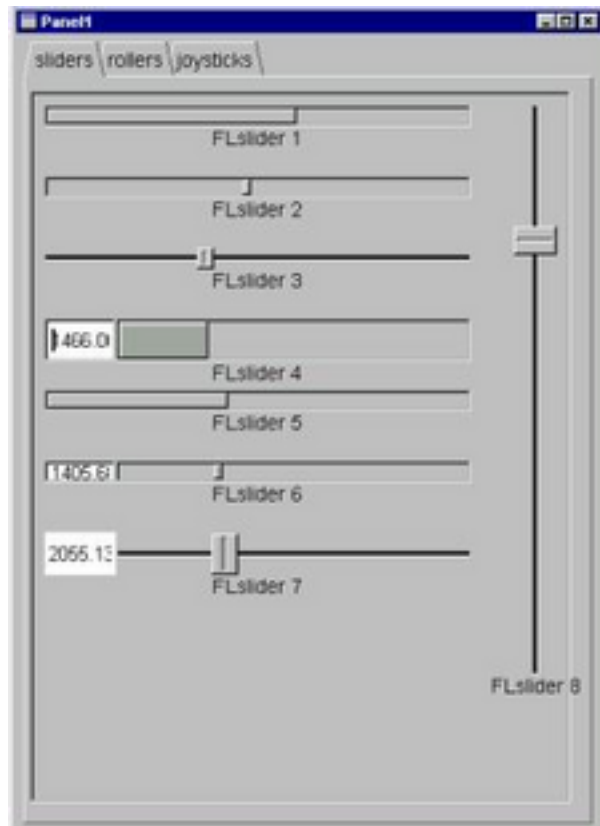
FLgroup "rollers", 380, 500, 10, 30, 2
gk1, ihr FLroller "FLroller 1", 50, 1000,.1,2 ,1 ,-1, 200,22, 20,50
gk2, ihr FLroller "FLroller 2", 80, 5000,1,2 ,1 ,-1, 200,22, 20,100
gk3, ihr FLroller "FLroller 3", 50, 1000,.1,2 ,1 ,-1, 200,22, 20,150
gk4, ihr FLroller "FLroller 4", 80, 5000,1,2 ,1 ,-1, 200,22, 20,200
gk5, ihr FLroller "FLroller 5", 50, 1000,.1,2 ,1 ,-1, 200,22, 20,250
gk6, ihr FLroller "FLroller 6", 80, 5000,1,2 ,1 ,-1, 200,22, 20,300
gk7, ihr FLroller "FLroller 7",50, 5000,1,1 ,2 ,-1, 30,300, 280,50
FLgroupEnd

FLgroup "joysticks", 380, 500, 10, 40, 3
gk1, gk2, ihj1, ihj2 FLjoy "FLjoy", 50, 18000, 50, 18000, 2, 2, -1, -1, 300, 300, 30, 60
FLgroupEnd

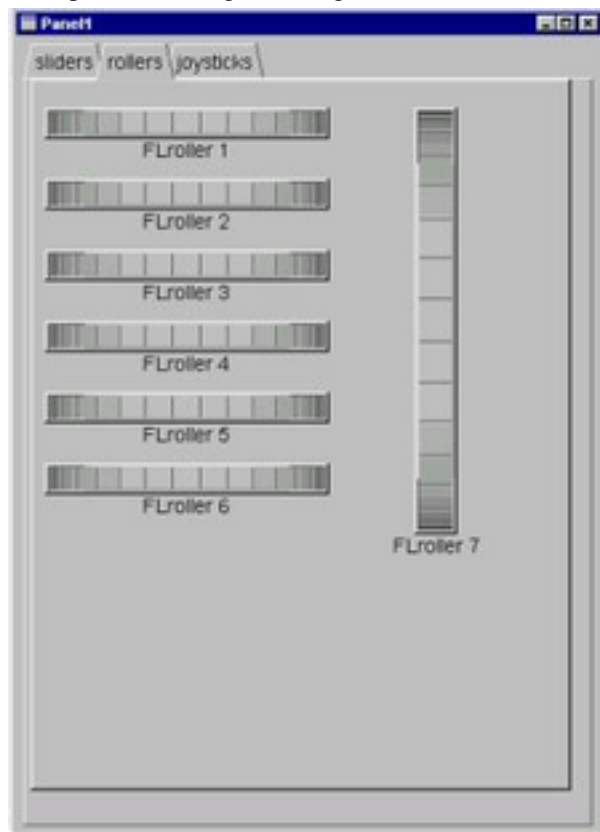
FLtabsEnd
FLscrollEnd
FLpanelEnd

```

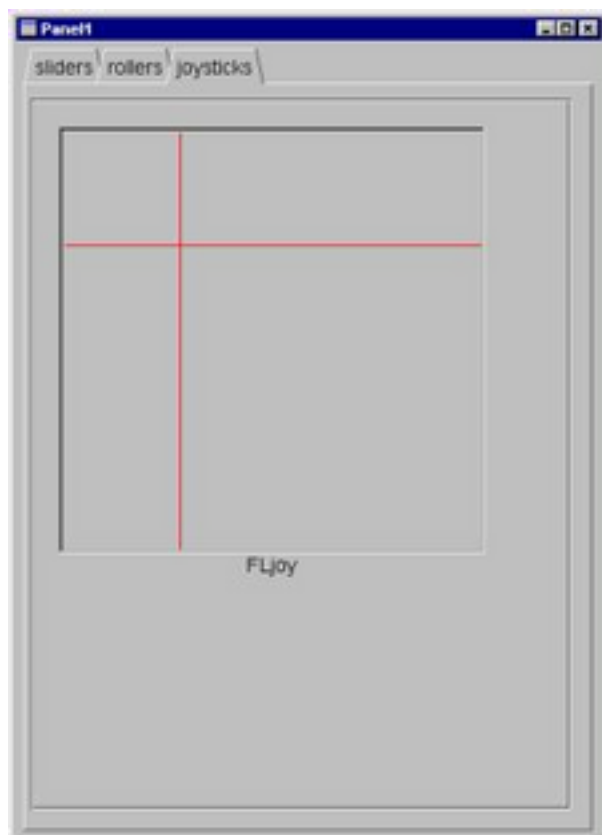
... produira le résultat suivant :



Exemple FLtabs, onglet des réglages.



Exemple FLtabs, onglet des molettes.



Exemple FLtabs, onglet du joystick.

(Chaque image montre un onglet différent sélectionné dans la même fenêtre.)

## Exemples

Voici un exemple de l'opcode FLtabs. Il utilise le fichier *FLtabs.csd* [examples/FLtabs.csd].

### Exemple 319. Exemple de l'opcode FLtabs.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o FLtabs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; A single oscillator with frequency, amplitude and
; panning controls on separate file tab cards
sr = 44100
kr = 441
ksmps = 100
nchnls = 2
```

```

FLpanel "Tabs", 300, 350, 100, 100
itabswidth = 280
itabsheight = 330
ix = 5
iy = 5
FLtabs itabswidth,itabsheight, ix,iy

    itab1width = 280
    itab1height = 300
    itab1x = 10
    itab1y = 40
    FLgroup "Tab 1", itab1width, itab1height, itab1x, itab1y
        gkfreq, i1 FLknob "Frequency", 200, 5000, -1, 1, -1, 70, 70, 130
        FLsetVal_i 400, i1
    FLgroupEnd

    itab2width = 280
    itab2height = 300
    itab2x = 10
    itab2y = 40
    FLgroup "Tab 2", itab2width, itab2height, itab2x, itab2y
        gkamp, i2 FLknob "Amplitude", 0, 15000, 0, 1, -1, 70, 70, 130
        FLsetVal_i 15000, i2
    FLgroupEnd

    itab3width = 280
    itab3height = 300
    itab3x = 10
    itab3y = 40
    FLgroup "Tab 3", itab3width, itab3height, itab3x, itab3y
        gkpan, i3 FLknob "Pan position", 0, 1, 0, 1, -1, 70, 70, 130
        FLsetVal_i 0.5, i3
    FLgroupEnd
FLtabsEnd
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
    ifn = 1
    asig oscili gkamp, gkfreq, ifn
    outs asig*(1-gkpan), asig*gkpan
endin

</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e

</CsScore>
</CsSoundSynthesizer>

```

## Voir aussi

*FLgroup, FLgroupEnd, FLpack, FLpackEnd, FLpanel, FLpanelEnd, FLscroll, FLscrollEnd, FLtabsEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLtabsEnd

FLtabsEnd — Marque la fin d'une interface FLTK à onglets.

## Description

Marque la fin d'une interface FLTK à onglets.

## Syntaxe

`FLtabsEnd`

## Exécution

Les conteneurs sont utiles pour formater l'apparence graphiques des widgets. Le conteneur le plus important est *FLpanel*, qui crée une fenêtre. Il peut être rempli avec d'autres conteneurs et/ou des valueurs ou d'autres sortes de widgets.

Il n'y a pas d'arguments de taux-k dans les conteneurs.

## Voir aussi

*FLgroup*, *FLgroupEnd*, *FLpack*, *FLpackEnd*, *FLpanel*, *FLpanelEnd*, *FLscroll*, *FLscrollEnd*, *FLtabs*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22



# FLtabs\_end

FLtabs\_end — Marque la fin d'une interface FLTK à onglets.

## Description

Marque la fin d'une interface FLTK à onglets. C'est un autre nom pour **FLtabsEnd** fourni pour des raisons de compatibilité. Voir *FLtabsEnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

# FLtext

FLtext — Un opcode de widget FLTK qui crée une boîte de texte.

## Description

*FLtext* permet à l'utilisateur de modifier la valeur d'un paramètre en la tapant directement dans un champ de texte.

## Syntaxe

```
kout, ihandle FLtext "label", imin, imax, istep, itype, iwidth, \  
iheight, ix, iy
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le widget correspondant. Il est utilisé par d'autres opcodes qui modifient les propriétés du widget (voir *Modifier l'Apparence des Widgets FLTK*). Il est automatiquement retourné par *FLtext* et ne doit pas être fixé par l'étiquette de l'utilisateur. (L'étiquette de l'utilisateur est une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.)

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*imin* -- valeur minimale de l'intervalle de sortie.

*imax* -- valeur maximale de l'intervalle de sortie.

*istep* -- un nombre en virgule flottante indiquant le pas d'incréméntation du valuateur correspondant au glissé de souris. L'argument *istep* permet de ralentir le glissé autorisant une précision arbitraire.

*itype* -- un nombre entier indiquant l'apparence du valuateur.

L'argument *itype* accepte les valeurs suivantes :

- 1 - comportement normal
- 2 - l'opération du glissé de souris est supprimée, deux boutons fléchés la remplacent. Un clic de souris sur un de ces boutons peut accroître/diminuer la valeur en sortie.
- 3 - l'édition du texte est supprimée, seul le glissé de souris modifie la valeur en sortie.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

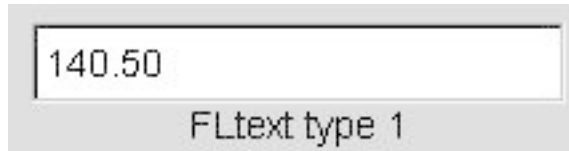
*ix* -- position horizontale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

## Exécution

*kout* -- valeur en sortie.

*FLtext* permet à l'utilisateur de modifier la valeur d'un paramètre en la tapant directement dans un champ de texte.



*FLtext*.

On peut aussi modifier sa valeur en le cliquant et en glissant la souris horizontalement. L'argument *istep* permet à l'utilisateur de fixer arbitrairement la réponse au glissé de souris.

## Exemples

Voici un exemple de l'opcode *FLtext*. Il utilise le fichier *FLtext.csd* [examples/FLtext.csd].

### Exemple 320. Exemple de l'opcode *FLtext*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLtext.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; A sine with oscillator with fltext box controlled
; frequency either click and drag or double click and
; type to change frequency value
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Frequency Text Box", 270, 600, 50, 50
; Minimum value output by the text box
imin = 200
; Maximum value output by the text box
imax = 5000
; Step size
istep = 1
; Text box graphic type
itype = 1
; Width of the text box in pixels
iwidth = 70
; Height of the text box in pixels
iheight = 30
; Distance of the left edge of the text box
; from the left edge of the panel
ix = 100
; Distance of the top edge of the text box
; from the top edge of the panel
iy = 300

gkfreq,ihandle FLtext "Enter the frequency", imin, imax, istep, itype, iwidth, iheight, ix, iy
; End of panel contents
```

```

FLpanelEnd
; Run the widget thread!
FLrun

instr 1
  iamp = 15000
  ifn = 1
  asig oscili iamp, gkfreq, ifn
  out asig
endin

</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*FLcount, FLjoy, FLknob, FLroller, FLslider*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLupdate

FLupdate — Identique à l'opcode FLrun.

## Description

Identique à l'opcode *FLrun*.

## Syntaxe

`FLupdate`

# fluidAllOut

fluidAllOut — Rassemble toutes les données audio depuis tous les moteurs Fluidsynth dans une exécution.

## Syntaxe

```
aleft, aright fluidAllOut
```

## Description

Opcodes du greffon fluidOpcodes.

Rassemble toutes les données audio depuis tous les moteurs Fluidsynth dans une exécution.

## Exécution

*aleft* -- Canal de sortie audio gauche.

*aright* -- Canal de sortie audio droite.

Appelez fluidAllOut dans une définition d'instrument dont le numéro est supérieur à ceux de toutes les définitions d'instrument de contrôle de fluid. Tous les SoundFonts envoient leur sortie audio à cet opcode. Envoyez une note de durée indéterminée à cet instrument afin d'activer les SoundFonts pour une durée suffisante.

Dans cette implémentation, les effets SoundFont tels que chorus ou réverbération sont utilisés si et seulement s'ils sont présents par défaut pour le preset. Il n'y a aucun moyen d'activer ou d'arrêter de tels effets, ou de changer leurs paramètres, depuis Csound.

## Exemples

Voici un exemple de l'opcode fluidAllOut. Il utilise le fichier *fluidAllOut.csd* [examples/fluidAllOut.csd].

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0      ;;realtime audio out and realtime midi in
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o fluidAllOut.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giengine1 fluidEngine
isfnum1 fluidLoad "sf_GMbank.sf2", giengine1, 1
fluidProgramSelect giengine1, 1, isfnum1, 0, 0

giengine2 fluidEngine
; soundfont path to manual/examples
isfnum2 fluidLoad "22Bassoon.sf2", giengine2, 1
```

```

    fluidProgramSelect giengine2, 1, isfnum2, 0, 70

instr 1

    mididefault    60, p3
    midinoteonkey p4, p5
ikey init p4
ivel init p5
    fluidNote giengine1, 1, ikey, ivel

endin

instr 2

    mididefault    60, p3
    midinoteonkey p4, p5
ikey init p4
ivel init p5
    fluidNote giengine2, 1, ikey, ivel

endin

instr 100

imvol init 7 ;amplify a bit
asigl, asigr fluidAllOut
    outs asigl*imvol, asigr*imvol

endin
</CsInstruments>
<CsScore>

i 1 0 2 60 127 ;play one note on instr 1
i 2 2 2 60 127 ;play another note on instr 2 and...
i 100 0 60      ;play virtual midi keyboard
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*fluidEngine, fluidNote, fluidLoad*

Plus d'information sur soundfonts dans les Floss Manuals : <http://write.flossmanuals.net/csound/d-reading-midi-files> [<http://write.flossmanuals.net/csound/d-reading-midi-files>]

D'autres informations sur soundfonts dans Wikipedia : <http://en.wikipedia.org/wiki/Soundfont> [<http://en.wikipedia.org/wiki/Soundfont>]

## Crédits

Opcode par Michael Gogins (gogins@pipeline.com). Merci à Peter Hanappe pour Fluidsynth, et à Steven Yi pour avoir réalisé qu'il était nécessaire de diviser Fluidsynth en plusieurs opcodes Csound différents.

# fluidCCi

fluidCCi — Envoie un message de données de contrôleur MIDI à fluid.

## Syntaxe

```
fluidCCi iEngineNumber, iChannelNumber, iControllerNumber, iValue
```

## Description

Opcode du greffon fluidOpcodes.

Envoie un message de données de contrôleur MIDI (numéro du contrôleur MIDI et valeur à utiliser) à un moteur fluid spécifié par son numéro, sur le numéro de canal MIDI indiqué.

## Initialisation

*iEngineNumber* -- numéro du moteur affecté par fluidEngine

*iChannelNumber* -- numéro du canal MIDI auquel le programme Fluidsynth est affecté : de 0 à 255. Les canaux MIDI dont le numéro est supérieur ou égal à 16 sont des canaux virtuels.

*iControllerNumber* -- numéro du contrôleur MIDI à utiliser pour ce message

*iValue* -- valeur à affecter au contrôleur (habituellement 0-127)

## Exécution

Cet opcode est utilisé pour affecter des valeurs de contrôleur pendant l'initialisation. Pour des changements continus, utilisez fluidCCk.

## Exemples

Voici un exemple de l'opcode fluidCCi. Il utilise le fichier *fluidCCi.csd* [examples/fluidCCi.csd].

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0      ;;realtime audio out and realtime midi in
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o fluidCCi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giengine fluidEngine
; soundfont path to manual/examples
isfnum fluidLoad "19Trumpet.sf2", giengine, 1
fluidProgramSelect giengine, 1, isfnum, 0, 56
```



```

instr 1

  mididefault 60, p3
  midinoteonkey p4, p5
  ikey init p4
  ivel init p5
  fluidCCi giengine, 1, 93, 127 ;full chorus &
  fluidCCi giengine, 1, 91, 127 ;full reverb!
  fluidNote giengine, 1, ikey, ivel

endin

instr 99

  imvol init 7
  asigl, asigr fluidOut giengine
  outs asigl*imvol, asigr*imvol

endin
</CsInstruments>
<CsScore>

i 1 0 5 60 100 ;play one note from score and...
i 99 0 60 ;play virtual keyboard for 60 sec.
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*fluidEngine, fluidNote, fluidLoad, fluidCCk*

Plus d'information sur soundfonts dans les Floss Manuals : <http://write.flossmanuals.net/csound/d-reading-midi-files> [http://write.flossmanuals.net/csound/d-reading-midi-files]

D'autres informations sur soundfonts dans Wikipedia : <http://en.wikipedia.org/wiki/Soundfont>

## Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

# fluidCCk

fluidCCk — Envoie un message de données de contrôleur MIDI à fluid.

## Syntaxe

```
fluidCCk iEngineNumber, iChannelNumber, iControllerNumber, kValue
```

## Description

Opcode du greffon fluidOpcodes.

Envoie un message de données de contrôleur MIDI (numéro du contrôleur MIDI et valeur à utiliser) à un moteur fluid spécifié par son numéro, sur le numéro de canal MIDI indiqué.

## Initialisation

*iEngineNumber* -- numéro du moteur affecté par fluidEngine

*iChannelNumber* -- numéro du canal MIDI auquel le programme Fluidsynth est affecté : de 0 à 255. Les canaux MIDI dont le numéro est supérieur ou égal à 16 sont des canaux virtuels.

*iControllerNumber* -- numéro du contrôleur MIDI à utiliser pour ce message

## Exécution

*kValue* -- valeur à affecter au contrôleur (habituellement 0-127)

## Exemples

Voici un exemple de l'opcode fluidCCk. Il utilise le fichier *fluidCCk.csd* [exemples/fluidCCk.csd].

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o fluidCCk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giengine fluidEngine
; soundfont path to manual/examples
isfnum fluidLoad "22Bassoon.sf2", giengine, 1
    fluidProgramSelect giengine, 1, isfnum, 0, 70

instr 1

    mididefault 60, p3
```

```
midinoteonkey p4, p5
ikey init p4
ivel init p5
kpan line 0, p3, 127 ;panning from left to right
fluidCCk giengine, 1, 10, kpan ;CC 10 = pan
fluidNote giengine, 1, ikey, ivel

endin

instr 99

imvol init 7
asigl, asigr fluidOut giengine
outs asigl*imvol, asigr*imvol

endin
</CsInstruments>
<CsScore>

i 1 0 4 48 100
i 1 4 2 50 120
i 1 6 1 53 80
i 1 7 1 45 70
i 1 8 1.5 48 80

i 99 0 10 ;keep instr 99 active
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*fluidEngine, fluidNote, fluidLoad, fluidCCi*

Plus d'information sur soundfonts dans les Floss Manuals : <http://write.flossmanuals.net/csound/d-reading-midi-files> [<http://write.flossmanuals.net/csound/d-reading-midi-files>]

D'autres informations sur soundfonts dans Wikipedia : <http://en.wikipedia.org/wiki/Soundfont>

## Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

# fluidControl

fluidControl — Envoie un note on, un note off, et d'autres messages MIDI à un preset SoundFont.

## Syntaxe

```
fluidControl ienginenum, kstatus, kchannel, \  
          kdata1, kdata2 [,imsgs]
```

## Description

Opcode du greffon fluidOpcodes.

Les opcodes fluid fournissent une intégration simple dans des opcodes de Csound du synthétiseur Fluidsynth SoundFont2 de Peter Hanappe. Cette implémentation accepte les messages MIDI de note on, note off, de contrôleur, de pitch bend ou de changement de programme au taux-k. La polyphonie maximale est de 4096 voix simultanées. N'importe quel nombre de SoundFonts peuvent être chargés et joués simultanément.

## Initialisation

*ienginenum* -- numéro du moteur affecté par fluidEngine

*imsgs* -- s'il est nul, suppression de l'affichage des messages à l'arrivée des commandes. Vaut 1 par défaut.

## Exécution

*kstatus* -- octet d'état du message de canal MIDI : 128 pour note off, 144 pour note on, 176 pour control change, 192 for program change, ou 224 pour pitch bend.

*kchannel* -- numéro du canal MIDI auquel le programme Fluidsynth est affecté : de 0 à 255. Les canaux MIDI dont le numéro est supérieur ou égal à 16 sont des canaux virtuels.

*kdata1* -- Pour note on, numéro de touche MIDI : de 0 (le plus bas) à 127 (le plus haut), où 60 est le do médian. Pour les messages de contrôleur continu, le numéro du contrôleur.

*kdata2* -- Pour note on, la vélocité de touche MIDI : de 0 (pas de son) à 127 (le plus fort). Pour les messages de contrôleur continu, la valeur du contrôleur.

Appelez fluidControl dans les définitions d'instrument qui jouent réellement des notes et qui envoient des messages de contrôle. Chaque définition d'instrument doit utiliser de manière cohérente un canal MIDI qui a été affecté à un programme Fluidsynth au moyen de fluidLoad.

Dans cette implémentation, les effets SoundFont tels que chorus ou réverbération sont utilisés si et seulement s'ils sont présents par défaut pour le preset. Il y a quelques moyens d'activer ou d'arrêter les effets de chorus et de réverbération en utilisant fluidEngine, et de changer certains de leurs paramètres avec fluidCCi et fluidCCk.

## Exemples

Voici un exemple plus complexe des opcodes fluidsynth. Il utilise le fichier *fluidcomplex.csd* [examples/fluidcomplex.csd].

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
;Anna.mid is a midi file, a song by The Beatles and can be found on the internet
-odac -T -F Anna.mid;;;realtime audio I/O and midifile in
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fluidcomplex.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

; Example by Istvan Varga

; disable triggering of instruments by MIDI events

ichn = 1
lp1:
    massign    ichn, 0
    loop_le    ichn, 1, 16, lp1
    pgmassign  0, 0

; initialize FluidSynth

gifld    fluidEngine
gisf2    fluidLoad "sf_GMbank.sf2", gifld, 1

; k-rate version of fluidProgramSelect

opcode fluidProgramSelect_k, 0, kkkkk
    keng, kchn, ksf2, kbnk, kpre xin
    igoto    skipInit
doInit:
    fluidProgramSelect i(keng), i(kchn), i(ksf2), i(kbnk), i(kpre)
    reinit    doInit
    rireturn
skipInit:
endop

instr 1
; initialize channels
kchn    init 1
if (kchn == 1) then
lp2:
    fluidControl gifld, 192, kchn - 1, 0, 0
    fluidControl gifld, 176, kchn - 1, 7, 100
    fluidControl gifld, 176, kchn - 1, 10, 64
    loop_le    kchn, 1, 16, lp2
endif

; send any MIDI events received to FluidSynth
nxt:
    kst, kch, kd1, kd2 midiin
    if (kst != 0) then
        if (kst != 192) then
            fluidControl gifld, kst, kch - 1, kd1, kd2
        else
            fluidProgramSelect_k gifld, kch - 1, gisf2, 0, kd1
        endif
        kgoto nxt
    endif
endif

```

```

; get audio output from FluidSynth
ivol    init 3 ;a bit louder
aL, aR fluidOut gfl d
        outs      aL*ivol, aR*ivol
endin

</CsInstruments>
<CsScore>

i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*fluidEngine, fluidNote, fluidLoad*

Plus d'information sur soundfonts dans les Floss Manuals : <http://write.flossmanuals.net/csound/d-reading-midi-files> [http://write.flossmanuals.net/csound/d-reading-midi-files]

D'autres informations sur soundfonts dans Wikipedia : <http://en.wikipedia.org/wiki/Soundfont>

## Crédits

Opcodes par Michael Gogins (gogins@pipeline.com). Merci à Peter Hanappe pour Fluidsynth, et à Steven Yi pour avoir réalisé qu'il était nécessaire de diviser Fluidsynth en plusieurs opcodes Csound différents.

Nouveau dans Csound5.00

Le paramètre facultatif `imsgs` a été introduit dans la version 6.14.

# fluidEngine

fluidEngine — Crée une instance de moteur fluidsynth.

## Syntaxe

```
ienginenum fluidEngine [iChorusEnabled] [, iReverbEnabled] [, iNumChannels] [, iPolyphony]
```

## Description

Opcodes du greffon fluidOpCodes.

Crée une instance de moteur fluidsynth, et retourne *ienginenum* pour identifier le moteur. *ienginenum* est passé à d'autres opcodes pour charger et jouer des SoundFonts et pour assembler le son généré.

## Initialisation

*ienginenum* -- numéro du moteur affecté par fluidEngine

*iChorusEnabled* -- fixé de manière facultative à 0 pour désactiver d'éventuels effets de chorus dans les SoundFonts chargés.

*iReverbEnabled* -- fixé de manière facultative à 0 pour désactiver d'éventuels effets de réverbération dans les SoundFonts chargés.

*iNumChannels* -- nombre de canaux à utiliser ; de 16 à 256, la valeur par défaut de Csound est 256 (la valeur par défaut de Fluidsynth est 16).

*iPolyphony* -- nombre de voix à jouer en parallèle ; de 16 à 4096, la valeur par défaut de Csound est 4096 (la valeur par défaut de Fluidsynth est 256). Note : ce n'est pas le nombre de notes jouées simultanément car une seule note peut utiliser plusieurs voix en fonction des zones d'instrument et de la vélocité et/ou du numéro de touche de la note jouée.

## Exemples

Voici un exemple des opcodes fluidsynth utilisant 2 moteurs. Il utilise les fichiers *fluidEngine.csd* [examples/fluidEngine.csd] et *midichn\_advanced.mid* [examples/midichn\_advanced.mid].

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -F midichn_advanced.mid ;; realtime audio out and midifile in
;-iadc ;; uncomment -iadc if realtime audio input is needed too
; For Non-realtime output leave only the line below:
; -o fluidEngine.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

; LOAD SOUNDFONTS
gienginenum1 fluidEngine
gienginenum2 fluidEngine
isfnum1 fluidLoad "sf_GMbank.sf2", gienginenum1, 1
```

```

        ; Piano 2, program 1, channel 1
fluidProgramSelect gienginenum1, 1, isfnum1, 0, 1
        ; Piano 3, program 2, channel 2
fluidProgramSelect gienginenum1, 2, isfnum1, 0, 2
isfnum2 fluidLoad "19Trumpet.sf2", gienginenum2, 1
        ; Trumpet, program 56, channel 3
fluidProgramSelect gienginenum2, 3, isfnum2, 0, 56

;Look for midifile in folder manual/examples
;"midichn_advanced.mid" sends notes to the soundfonts

instr 1 ; GM soundfont
; INITIALIZATION
mididefault 60, p3 ; Default duration of 60 -- overridden by score.
midinoteonkey p4, p5 ; Channels MIDI input to pfields.
; Use channel assigned in fluidload.
ichannel = 1
ikey = p4
ivelocity = p5
fluidNote gienginenum1, ichannel, ikey, ivelocity
endin

instr 2 ; GM soundfont
; INITIALIZATION
mididefault 60, p3 ; Default duration of 60 -- overridden by score.
midinoteonkey p4, p5 ; Channels MIDI input to pfields.
; Use channel assigned in fluidload.
ichannel = 2
ikey = p4
ivelocity = p5
fluidNote gienginenum1, ichannel, ikey, ivelocity
endin

instr 3 ; Trumpet
; INITIALIZATION
mididefault 60, p3 ; Default duration of 60 -- overridden by score.
midinoteonkey p4, p5 ; Channels MIDI input to pfields.
; Use channel assigned in fluidload.
ichannel = 3
ikey = p4
ivelocity = p5
fluidNote gienginenum2, ichannel, ikey, ivelocity
endin

; COLLECT AUDIO FROM ALL SOUND FONTS

instr 100 ; Fluidsynth output

iampplitude1 = 7
iampplitude2 = 7

; AUDIO
aleft1, aright1 fluidOut gienginenum1
aleft2, aright2 fluidOut gienginenum2
outs (aleft1 * iampplitude1) + (aleft2 * iampplitude2), \
      (aright1 * iampplitude1) + (aright2 * iampplitude2)

endin
</CsInstruments>
<CsScore>

i 1 0 3 60 100
i 2 1 3 60 100
i 3 3 3 63 100
i 100 0 10 ;run for 10 seconds
e
</CsScore>
</CsoundSynthesizer>

```



## Voir aussi

*fluidNote, fluidLoad*

Plus d'information sur soundfonts dans les Floss Manuals : <http://write.flossmanuals.net/csound/d-reading-midi-files> [<http://write.flossmanuals.net/csound/d-reading-midi-files>]

D'autres informations sur soundfonts dans Wikipedia : <http://en.wikipedia.org/wiki/Soundfont>

## Crédits

Michael Gogins ([gogins@pipeline.com](mailto:gogins@pipeline.com)), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

Les paramètres facultatifs *iNumChannels* et *iPolyphony* ont été ajoutés dans la version 5.07

L'ordre des arguments a été corrigé dans le manuel en août 2019.

# fluidInfo

fluidInfo — Retrouve l'information de programme du SoundFont courant.

## Syntaxe

```
SPrograms[] fluidInfo ienginenum
```

## Description

Opcodes du greffon fluidOpcodes.

Parcourt le SoundFont courant pour en extraire l'information de programme. Cette information est retournée dans un tableau de chaînes de caractères.

## Initialisation

*SPrograms[]* -- Tableau de chaînes de caractères contenant l'information de programme.

*ienginenum* -- numéro du moteur affecté par fluidEngine.

## Exécution

fluidInfo ne fonctionne que durant l'initialisation.

## Exemples

Voici un exemple de l'opcode fluidInfo. Il utilise le fichier *fluidInfo.csd* [examples/fluidInfo.csd] and *sf\_GMbank.sf2* [examples/sf\_GMbank.sf2].

### Exemple 321. Exemple de l'opcode fluidInfo.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;;realtime audio out and realtime midi in
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giengine fluidEngine
isfnum fluidLoad "sf_GMbank.sf2", giengine, 1

instr 1
iCnt init 0
SSoundFontPrograms[] fluidInfo giengine
iNumberOfPrograms lenarray SSoundFontPrograms
```

```
until iCnt>=iNumberOfPrograms do
  printf_i "%s\n", 1, SSoundFontPrograms[iCnt]
  iCnt = iCnt+1
od
endin

</CsInstruments>
<CsScore>
i1 0 1
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*fluidEngine, fluidLoad, fluidNote*

Plus d'information sur soundfonts dans les Floss Manuals : <http://write.flossmanuals.net/csound/d-reading-midi-files> [<http://write.flossmanuals.net/csound/d-reading-midi-files>]

D'autres informations sur soundfonts dans Wikipedia : <http://en.wikipedia.org/wiki/Soundfont> [<http://en.wikipedia.org/wiki/Soundfont>]

## Crédits

Rory Walsh, basé sur l'opcode fluidLoad de Steven Yi.

Nouveau dans Csound 6.12

# fluidLoad

**fluidLoad** — Charge un SoundFont dans un fluidEngine, en listant éventuellement le contenu du SoundFont.

## Syntaxe

```
isfnum fluidLoad soundfont, ienginenum[, ilistpresets]
```

## Description

Opcode du greffon fluidOpcodes.

Charge un SoundFont dans une instance d'un fluidEngine, en listant éventuellement les banques et les presets du SoundFont.

## Initialisation

*isfnum* -- numéro affecté au soundfont qui vient d'être chargé.

*soundfont* -- chaîne spécifiant le nom de fichier d'un SoundFont. Notez que n'importe quel nombre de SoundFonts peuvent être chargés (évidemment, par différents appels de fluidLoad).

*ienginenum* -- numéro du moteur affecté par fluidEngine

*ilistpresets* -- facultatif, s'il est spécifié, tous les programmes Fluidsynth du SoundFont qui vient d'être chargé sont listés. Un programme FluidSynth est une combinaison d'ID de SoundFont, de numéro de banque, et de numéro de preset qui est affecté à un canal MIDI.

## Exécution

Appelez fluidLoad dans l'en-tête de l'orchestre, autant de fois que vous voulez. Le même SoundFont peut être appelé pour affecter des programmes à des canaux MIDI autant de fois que l'on veut ; le SoundFont n'est chargé que la première fois.

## Exemples

Voici un exemple de l'opcode fluidLoad. Il utilise les fichiers *fluidLoad.csd* [examples/fluidLoad.csd] et *07AcousticGuitar.sf2* [examples/07AcousticGuitar.sf2].

### Exemple 322. Exemple de l'opcode fluidLoad.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac  -+rtmidi=virtual  -M0      ;;;realtime audio out and realtime midi in
;-iadc      ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o fluidLoad.wav -W ;;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giengine fluidEngine
; soundfont path to manual/examples
isfnum fluidLoad "07AcousticGuitar.sf2", giengine, 1
fluidProgramSelect giengine, 1, isfnum, 0, 0

instr 1

mididefault 60, p3
midinoteonkey p4, p5
ikey init p4
ivel init p5
fluidNote giengine, 1, ikey, ivel

endin

instr 99

imvol init 7
asigl, asigr fluidOut giengine
outs asigl*imvol, asigr*imvol

endin
</CsInstruments>
<CsScore>

i 1 0 2 60 100 ;play one note from score and...
i 99 0 60 ;play virtual keyboard for 60 sec.
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra une ligne comme celle-ci :

```
SoundFont: 1 Bank: 0 Preset: 0 Seagul Acoustic Git
```

## Voir aussi

*fluidEngine, fluidNote*

Plus d'information sur soundfonts dans les Floss Manuals : <http://write.flossmanuals.net/csound/d-reading-midi-files> [http://write.flossmanuals.net/csound/d-reading-midi-files]

D'autres informations au sujet de soundfonts sur Wikipedia : <http://en.wikipedia.org/wiki/Soundfont>

## Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

Nouveau dans Csound5.00

# fluidNote

fluidNote — Joue une note sur un canal dans un moteur fluidsynth.

## Syntaxe

**fluidNote** *iengine*num, *ichannel*num, *imidi*key, *imidi*vel

## Description

Opcodes du greffon fluidOpcodes.

Joue une note de hauteur *imidi*key et de vélocité *imidi*vel sur le canal *ichannel*num du fluidEngine numéro *iengine*num.

## Initialisation

*iengine*num -- numéro du moteur affecté par fluidEngine

*ichannel*num -- numéro de canal sur lequel jouer la note dans le fluidEngine donné

*imidi*key -- touche MIDI de la note (0-127)

*imidi*vel -- vélocité MIDI de la note (0-127)

## Exemples

Voici un exemple de l'opcode fluidNote. Il utilise les fichiers *fluidNote.csd* [examples/fluidNote.csd] et *19Trumpet.sf2* [examples/19Trumpet.sf2].

### Exemple 323. Exemple de l'opcode fluidNote.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac  -+rtmidi=virtual  -M0      ;;realtime audio out and realtime midi in
;-iadc  ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o fluidNote.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giengine fluidEngine
; soundfont path to manual/examples
isfnum fluidLoad "19Trumpet.sf2", giengine, 1
fluidProgramSelect giengine, 1, isfnum, 0, 56
```

```

instr 1

    mididefault 60, p3
    midinoteonkey p4, p5
    ikey init p4
    ivel init p5
    fluidNote giengine, 1, ikey, ivel

endin

instr 99

    imvol init 7
    asigl, asigr fluidOut giengine
    outs asigl*imvol, asigr*imvol

endin
</CsInstruments>
<CsScore>

i 1 0 2 60 100 ;play one note from score and...
i 99 0 60      ;play virtual keyboard for 60 sec.
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra une ligne comme celle-ci :

```
SoundFont: 1 Bank: 0 Preset: 56 Trumpet metallic
```

## Voir aussi

*fluidEngine, fluidLoad*

Plus d'information sur soundfonts dans les Floss Manuals : <http://write.flossmanuals.net/csound/d-reading-midi-files> [http://write.flossmanuals.net/csound/d-reading-midi-files]

D'autres informations au sujet de soundfonts sur Wikipedia : <http://en.wikipedia.org/wiki/Soundfont>

## Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

# fluidOut

fluidOut — Envoie en sortie le son d'un fluidEngine donné.

## Syntaxe

```
aleft, aright fluidOut ienginenum
```

## Description

Opcode du greffon fluidOpcodes.

Envoie en sortie le son d'un fluidEngine donné.

## Initialisation

*ienginenum* -- numéro du moteur affecté par fluidEngine

## Exécution

*aleft* -- Canal de sortie audio gauche.

*aright* -- Canal de sortie audio droite.

Appelez fluidOut dans une définition d'instrument dont le numéro est supérieur à ceux de toutes les définitions d'instrument de contrôle de fluid. Tous les SoundFonts utilisés par le fluidEngine numéro *ienginenum* envoient leur sortie audio à cet opcode. Envoyez une note de durée indéterminée à cet instrument afin d'activer les SoundFonts pour une durée suffisante.

## Exemples

Voici un exemple de l'opcode fluidOut. Il utilise les fichiers *fluidOut.csd*, *01hpschd.sf2* [examples/01hpschd.sf2] et [examples/fluidOut.csd]*22Bassoon.sf2* [examples/22Bassoon.sf2].

### Exemple 324. Exemple de l'opcode fluidOut.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac  -+rtmidi=virtual  -M0      ;;;realtime audio out and realtime midi in
;-iadc  ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o fluidOut.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
```



```

giengine1 fluidEngine
; soundfont path to manual/examples
isfnum1 fluidLoad "01hpschd.sf2", giengine1, 1
fluidProgramSelect giengine1, 1, isfnum1, 0, 0

giengine2 fluidEngine
; soundfont path to manual/examples
isfnum2 fluidLoad "22Bassoon.sf2", giengine2, 1
fluidProgramSelect giengine2, 1, isfnum2, 0, 70

instr 1

mididefault 60, p3
midinoteonkey p4, p5
ikey init p4
ivel init p5
fluidNote giengine1, 1, ikey, ivel

endin

instr 2

mididefault 60, p3
midinoteonkey p4, p5
ikey init p4
ivel init p5
fluidNote giengine2, 1, ikey, ivel

endin

instr 98

imvol init 7
asigl, asigr fluidOut giengine1
outs asigl*imvol, asigr*imvol

endin

instr 99

imvol init 4
asigl, asigr fluidOut giengine2 ;add a stereo flanger
adelL linseg 0, p3*.5, 0.02, p3*.5, 0 ;max delay time =20ms
adelR linseg 0.02, p3*.5, 0, p3*.5, 0.02 ;max delay time =20ms
asigl flanger asigl, adelL, .6
asigr flanger asigr, adelR, .6
outs asigl*imvol, asigr*imvol

endin
</CsInstruments>
<CsScore>

i 1 0 2 60 100 ;play one note of instr 1
i 2 2 2 60 100 ;play another note of instr 2 and...
i 98 0 60 ;play virtual keyboard for 60 sec.
i 99 0 60
e
</CsScore>
</CsoundSynthesizer>

```

La sortie contiendra des lignes comme celles-ci :

```

chnl 1 using instr 1
chnl 2 using instr 2

SoundFont: 1 Bank: 0 Preset: 0 Harpsichord I-8
SoundFont: 1 Bank: 0 Preset: 70 Ethan Bassoon mono

```

## Voir aussi

*fluidEngine, fluidNote, fluidLoad*

Plus d'information sur soundfonts dans les Floss Manuals : <http://write.flossmanuals.net/csound/d-reading-midi-files> [<http://write.flossmanuals.net/csound/d-reading-midi-files>]

D'autres informations au sujet de soundfonts sur Wikipedia : <http://en.wikipedia.org/wiki/Soundfont>

## Crédits

Michael Gogins ([gogins@pipeline.com](mailto:gogins@pipeline.com)), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

Nouveau dans Csound5.00

# fluidProgramSelect

fluidProgramSelect — Affecte un preset d'un SoundFont à un canal d'un fluidEngine.

## Syntaxe

```
fluidProgramSelect ienginenum, ichannelnum, isfnum, ibanknum, ipresetnum
```

## Description

Opcodes du greffon fluidOpcodes.

Affecte un preset d'un SoundFont à un canal d'un fluidEngine.

## Initialisation

*ienginenum* -- numéro du moteur affecté par fluidEngine

*ichannelnum* -- numéro du canal auquel affecter le preset dans le fluidEngine donné

*isfnum* -- numéro du SoundFont duquel le preset est issu

*ibanknum* -- numéro de la banque dans le SoundFont de laquelle le preset est issu

*ipresetnum* -- numéro du preset à affecter

## Exemples

Voici un exemple de l'opcode fluidProgramSelect. Il utilise le fichier *fluidProgramSelect.csd* [exemples/fluidProgramSelect.csd].

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0    ;;realtime audio out and realtime midi in
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o fluidProgramSelect.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giengine fluidEngine
; soundfont path to manual/examples
isfnum fluidLoad "19Trumpet.sf2", giengine, 1
    fluidProgramSelect giengine, 1, isfnum, 0, 56

instr 1

    mididefault 60, p3
    midinoteonkey p4, p5
    ikey init p4
```

```

ivel init p5
fluidNote giengine, 1, ikey, ivel

endin

instr 99

imvol init 7
asigl, asigr fluidOut giengine
outs asigl*imvol, asigr*imvol

endin
</CsInstruments>
<CsScore>

i 1 0 2 60 100 ;play one note from score and...
i 99 0 60 ;play virtual keyboard for 60 sec.
e

</CsScore>
</CsoundSynthesizer>

```

Voici un autre exemple de l'opcode fluidProgramSelect écrit par Istvan Varga. Il utilise le fichier *fluid-complex.csd* [examples/fluidcomplex.csd].

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
;Anna.mid is a midi file, a song by The Beatles and can be found on the internet
-odac -T -F Anna.mid;;;realtime audio I/O and midifile in
;-iadc ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fluidcomplex.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

; Example by Istvan Varga

; disable triggering of instruments by MIDI events

ichn = 1
lp1:
massign ichn, 0
loop_le ichn, 1, 16, lp1
pgmassign 0, 0

; initialize FluidSynth

gifld fluidEngine
gisf2 fluidLoad "sf_GMbank.sf2", gifld, 1

; k-rate version of fluidProgramSelect

opcode fluidProgramSelect_k, 0, kkkkk
keng, kchn, ksf2, kbnk, kpre xin
igoto skipInit
doInit:
fluidProgramSelect i(keng), i(kchn), i(ksf2), i(kbnk), i(kpre)
reinit doInit
rireturn
skipInit:

```

```

endop

instr 1
; initialize channels
kchn init 1
if (kchn == 1) then
lp2:
    fluidControl gifld, 192, kchn - 1, 0, 0
    fluidControl gifld, 176, kchn - 1, 7, 100
    fluidControl gifld, 176, kchn - 1, 10, 64
    loop_le kchn, 1, 16, lp2
endif

; send any MIDI events received to FluidSynth
nxt:
kst, kch, kd1, kd2 midiin
if (kst != 0) then
    if (kst != 192) then
        fluidControl gifld, kst, kch - 1, kd1, kd2
    else
        fluidProgramSelect_k gifld, kch - 1, gisf2, 0, kd1
    endif
    kgoto nxt
endif

; get audio output from FluidSynth
ivol init 3 ;a bit louder
aL, aR fluidOut gifld
outs aL*ivol, aR*ivol

endin

</CsInstruments>
<CsScore>

i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*fluidEngine, fluidNote, fluidLoad*

Plus d'information sur soundfonts dans les Floss Manuals : <http://write.flossmanuals.net/csound/d-reading-midi-files> [http://write.flossmanuals.net/csound/d-reading-midi-files]

D'autres informations sur soundfonts dans Wikipedia : <http://en.wikipedia.org/wiki/Soundfont>

## Crédits

Michael Gogins (gogins@pipeline.com), Steven Yi. Merci à Peter Hanappe pour Fluidsynth.

# fluidSetInterpMethod

fluidSetInterpMethod — Fixe la méthode d'interpolation pour un canal dans le moteur fluidsynth.

## Syntaxe

**fluidSetInterpMethod** ienginenum, ichannelnum, iInterpMethod

## Description

Opcodes du greffon fluidOpcodes.

Fixe la méthode d'interpolation pour un canal dans le moteur fluidsynth. Les méthodes d'interpolation d'ordre inférieur donnent une restitution plus rapide et de moindre qualité tandis que les méthodes d'interpolation d'ordre élevé donnent une restitution plus lente et de meilleure qualité. L'interpolation par défaut pour un canal est du quatrième ordre.

## Initialisation

*ienginenum* -- numéro du moteur alloué par *fluidEngine*

*ichannelnum* -- numéro de canal à utiliser pour le preset dans le moteur fluidsynth donné

*iInterpMethod* -- méthode d'interpolation, l'une des suivantes

- 0 -- Pas d'interpolation
- 1 -- Interpolation linéaire
- 4 -- Interpolation d'ordre 4 (par défaut)
- 7 -- Interpolation d'ordre 7 (la plus haute)

## Exemples

Voici un exemple de l'opcode fluidSetInterpMethod. Il utilise les fichiers *fluidSetInterpMethod.csd* [examples/fluidSetInterpMethod.csd] et *07AcousticGuitar.sf2* [examples/07AcousticGuitar.sf2].

### Exemple 325. Exemple de l'opcode fluidSetInterpMethod.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out and realtime midi in
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o fluidSetInterpMethod.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giengine fluidEngine
; soundfont path to manual/examples
isfnum fluidLoad "07AcousticGuitar.sf2", giengine, 1
    fluidProgramSelect giengine, 1, isfnum, 0, 0

instr 1

    mididefault 60, p3
    midinoteonkey p4, p5
ikey init p4
ivel init p5
iInterpMethod = p6
fluidSetInterpMethod giengine, 1, iInterpMethod
    fluidNote giengine, 1, ikey, ivel

endin

instr 99

imvol init 7
asigl, asigr fluidOut giengine
    outs asigl*imvol, asigr*imvol

endin
</CsInstruments>
<CsScore>
;hear the difference
i 1 0 2 60 120 0 ;no interpolation
i 1 3 2 72 120 0
i 1 6 2 60 120 7 ;7th order interpolation
i 1 9 2 72 120 7

i 99 0 12

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*fluidEngine*

Plus d'information sur soundfonts dans les Floss Manuals : <http://write.flossmanuals.net/csound/d-reading-midi-files> [http://write.flossmanuals.net/csound/d-reading-midi-files]

D'autres informations au sujet de soundfonts sur Wikipedia : <http://en.wikipedia.org/wiki/Soundfont>

## Crédits

Auteur : Steven Yi

Nouveau dans la version 5.07

# FLvalue

FLvalue — Montre la valeur courante d'un valuateur FLTK.

## Description

*FLvalue* montre la valeur courante d'un valuateur dans un champ texte.

## Syntaxe

```
ihandle FLvalue "label", iwidth, iheight, ix, iy
```

## Initialisation

*ihandle* -- un identifiant (un nombre entier) qui référence de manière univoque le valuateur correspondant. Il peut être utilisé comme argument *idisp* d'un valuateur.

« *label* » -- une chaîne entre guillemets contenant un texte fourni par l'utilisateur placé à côté du widget.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix* -- position horizontale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy* -- position verticale du coin supérieur gauche du valuateur, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

## Exécution

*FLvalue* montre la valeur courante d'un valuateur dans un champ texte. Il retourne *ihandle* qui peut être utilisé comme argument *idisp* d'un valuateur (voir la section *Valuateurs FLTK*). De cette manière, les valeurs de ce valuateur seront montrées dynamiquement dans un champ texte.



### Note

Noter que *FLvalue* n'est pas un valuateur et que sa valeur ne peut pas être modifiée. La valeur d'un widget *FLvalue* ne doit être fixée que par d'autres widgets, et PAS depuis *FLsetVal* ou *FLsetVal\_i* car cela pourrait planter Csound.

## Exemples

Voici un exemple de l'opcode FLvalue. Il utilise le fichier *FLvalue.csd* [exemples/FLvalue.csd].

### Exemple 326. Exemple de l'opcode FLvalue.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
```



```

<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLvalue.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Using the opcode flvalue to display the output of a slider
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

FLpanel "Value Display Box", 900, 200, 50, 50
; Width of the value display box in pixels
iwidth = 50
; Height of the value display box in pixels
iheight = 20
; Distance of the left edge of the value display
; box from the left edge of the panel
ix = 65
; Distance of the top edge of the value display
; box from the top edge of the panel
iy = 55

idisp FLvalue "Hertz", iwidth, iheight, ix, iy
gkfreq, ihandle FLslider "Frequency", 200, 5000, -1, 5, idisp, 750, 30, 125, 50
FLsetVal_i 500, ihandle
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
iamp = 15000
ifn = 1
asig oscili iamp, gkfreq, ifn
out asig
endin

</CsInstruments>
<CsScore>

; Function table that defines a single cycle
; of a sine wave.
f 1 0 1024 10 1

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e

</CsScore>
</CsSoundSynthesizer>

```

## Voir aussi

*FLbox, FLbutBank, FLbutton, FLprintk, FLprintk2*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.22

Exemple écrit par Iain McCurdy, édité par Kevin Conder.

# FLvkeybd

FLvkeybd — Un opcode de widget FLTK qui crée un widget de clavier virtuel.

## Description

Un opcode de widget FLTK qui crée un widget de clavier virtuel. Il doit être utilisé avec le pilote du clavier virtuel midi pour opérer correctement. Cet opcode est utile pour faire des versions de démonstration d'orchestres MIDI avec le clavier virtuel inclus dans la fenêtre principale.



### Note

La version widget du clavier virtuel ne comprend pas les réglettes MIDI que l'on trouve dans la version complète de la fenêtre du clavier virtuel.

## Syntaxe

```
FLvkeybd "keyboard.map", iwidth, iheight, ix, iy
```

## Initialisation

« *keyboard.map* » -- une chaîne de caractères entre guillemets contenant le mappage du clavier à utiliser. On peut fournir une chaîne vide ("" ) pour utiliser les valeurs de nom de banque/canal par défaut. Voir Clavier Virtuel Midi pour plus d'information sur les mappages du clavier.

*iwidth* -- largeur du widget.

*iheight* -- hauteur du widget.

*ix*-- position horizontale du coin supérieur gauche du clavier, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).

*iy*-- position verticale du coin supérieur gauche du clavier, relative au coin supérieur gauche de la fenêtre correspondante (exprimée en pixels).



### Note

La largeur et la hauteur standard du clavier virtuel est de 624x120 pour la version dialogue qui est montrée quand *FLvkeybd* n'est pas utilisé.

## Voir aussi

*FLbutton*, *FLbox*, *FLbutBank*, *FLprintk*, *FLprintk2*, *FLvalue*

## Crédits

Auteur : Steven Yi

Nouveau dans la version 5.05

# FLvslidBnk

FLvslidBnk — Un widget FLTK contenant un banc de réglettes verticales.

## Description

*FLvslidBnk* est un widget contenant un banc de réglettes verticales.

## Syntaxe

```
FLvslidBnk "names", inumsliders [, ioutable] [, iwidth] [, iheight] [, ix] \
    [, iy] [, itypetable] [, iexptable] [, istart_index] [, iminmaxtable]
```

## Initialisation

« *names* » -- une chaîne de caractères entre guillemets contenant le nom de chaque réglette. Chaque réglette peut avoir un nom différent. Chaque nom est séparé par un caractère « @ », par exemple : « fréquence@amplitude@coupure ». Il est possible de ne fournir aucun nom en donnant un seul espace « ». Dans ce cas, l'opcode affectera automatiquement un numéro en progression ascendante comme étiquette pour chaque réglette.

*inumsliders* -- le nombre de réglettes.

*ioutable* (facultatif, 0 par défaut) -- numéro d'une table allouée préalablement dans laquelle seront stockées les valeurs de sortie de chaque réglette. Il faut s'assurer que la taille de la table est suffisante pour contenir toutes les cellules de sortie, sinon Csound plantera avec une erreur de segmentation. En affectant zéro à cet argument, la sortie sera dirigée vers l'espace zak dans la zone de taux-k. Dans ce cas, l'espace zak doit avoir été alloué au préalable avec l'opcode *zakinit* et il faut s'assurer que la taille d'allocation est suffisante pour couvrir toutes les réglettes. La valeur par défaut est zéro (c'est-à-dire stockage de la sortie dans l'espace zak).

*istart\_index* (facultatif, 0 par défaut) -- un nombre entier indiquant un décalage des positions des cellules de sortie. Il peut être positif pour permettre l'allocation en sortie de plusieurs bancs de réglettes dans la même table ou dans l'espace zak. La valeur par défaut est zéro (pas de décalage).

*iminmaxtable* (facultatif, 0 par défaut) -- numéro d'une table définie au préalable contenant une liste de couples min-max pour chaque réglette. Une valeur de zéro signifie l'intervalle allant de 0 à 1 pour toutes les réglettes, sans fournir de table. La valeur par défaut est zéro.

*iexptable* (facultatif, 0 par défaut) -- numéro d'une table définie au préalable contenant une liste d'identifiants (des nombres entiers) fournis pour modifier le comportement de chaque réglette de manière indépendante. Les identifiants peuvent avoir les valeurs suivantes :

- -1 -- courbe de réponse exponentielle
- 0 -- réponse linéaire
- nombre > 0 -- suit la courbe d'une table définie au préalable pour mettre en forme la réponse de la réglette correspondante. Dans ce cas, ce nombre correspond au numéro de la table.

On peut souhaiter que toutes les réglettes du banc aient la même courbe de réponse (exponentielle ou linéaire). Dans ce cas, on peut affecter -1 ou 0 à *iexptable* sans se préoccuper de définir auparavant une

table. La valeur par défaut est zéro (toutes les réglettes ont une réponse linéaire sans avoir à fournir de table).

*ityetable* (facultatif, 0 par défaut) -- numéro d'une table définie au préalable contenant une liste d'identifiants (des nombres entiers) fournis pour modifier l'aspect de chaque réglette de manière indépendante. Les identifiants peuvent avoir les valeurs suivantes :

- 0 = Réglette stylée
- 1 = Réglette pleine
- 3 = Réglette normale
- 5 = Réglette stylée
- 7 = Réglette stylée en creux

On peut souhaiter que toutes les réglettes du banc aient le même aspect. Dans ce cas, on peut affecter un nombre négatif à *ityetable* sans se préoccuper de définir auparavant une table. Les nombres négatifs ont la même signification que les identifiants positifs correspondants sauf que le même aspect est affecté à toutes les réglettes. On peut aussi donner un aspect aléatoire à chaque réglette en affectant à *ityetable* un nombre négatif inférieur à -7. La valeur par défaut est 0 (toutes les réglettes sont stylées, sans avoir à fournir de table).

On peut ajouter 20 à une valeur dans la table pour donner l'aspect "plastique" à la réglette, ou soustraire 20 si l'on veut affecter la valeur à tous les widgets sans définir une table (par exemple -21 pour donner à toutes les réglettes le type Plastique Plein).

*iwidth* (facultatif) -- largeur de la zone rectangulaire contenant toutes les réglettes du banc, à l'exclusion des étiquettes qui sont placées sous cette zone.

*iheight* (facultatif) -- hauteur de la zone rectangulaire contenant toutes les réglettes du banc, à l'exclusion des étiquettes qui sont placées sous cette zone.

*ix* (facultatif) -- position horizontale du coin supérieur gauche de la zone rectangulaire contenant toutes les réglettes appartenant au banc. Il faut laisser suffisamment d'espace en-dessous de ce rectangle afin que les étiquettes des réglettes soient visibles. En effet, les étiquettes elles-mêmes sont situées à l'extérieur de la zone rectangulaire.

*iy* (facultatif) -- position verticale du coin supérieur gauche de la zone rectangulaire contenant toutes les réglettes appartenant au banc. Il faut laisser suffisamment d'espace en-dessous de ce rectangle afin que les étiquettes des réglettes soient visibles. En effet, les étiquettes elles-mêmes sont situées à l'extérieur de la zone rectangulaire.

## Exécution

Il n'y a pas d'argument de taux-k, même si les cellules de la table en sortie (ou l'espace zak) sont mis à jour au taux-k.

*FLvsIidBnk* est un widget contenant un banc de réglettes verticales. On peut y mettre n'importe quel nombre de réglettes (argument *inumsliders*). La sortie de toutes les réglettes est stockée dans une table allouée au préalable ou dans l'espace zak (argument *ioutable*). Il est possible de déterminer la première position de la table (ou de l'espace zak) dans lequel stocker la sortie de la première réglette au moyen de l'argument *istart\_index*.

Chaque réglette peut avoir une étiquette individuelle placée sous elle. Les étiquettes sont définies par l'argument « *names* ». L'intervalle de sortie de chaque réglette peut être fixé individuellement au moyen d'une

table externe (argument *iminmaxtable*). La courbe de réponse de chaque réglette peut être fixée individuellement, au moyen d'une liste d'identifiants placés dans une table (argument *iexptable*). Il est possible de définir l'aspect de chaque réglette indépendamment ou de donner le même aspect à toutes les réglettes (argument *ityetable*).

Les arguments *iwidth*, *iheight*, *ix* et *iy* déterminent la largeur, la hauteur, les positions horizontale et verticale de la zone rectangulaire contenant les réglettes. Noter que l'étiquette de chaque réglette est placée en-dessous et n'est pas incluse dans la zone rectangulaire contenant les réglettes. Ainsi l'utilisateur doit laisser assez d'espace sous le banc en affectant une valeur suffisante à *iy* afin que les étiquettes soient visibles.

*FLvslidBnk* est identique à *FLslidBnk* sauf qu'il contient des réglettes verticales plutôt qu'horizontales. Comme la largeur de chaque réglette est souvent petite, il est recommandé de ne laisser qu'un seul espace dans la chaîne de noms (" "), ce qui fait que chaque réglette sera numérotée automatiquement.



## IMPORTANT !

Noter que les tables utilisées par *FLvslidBnk* doivent être créées avec l'opcode *ftgen* et placées dans l'orchestre avant le valuateur correspondant. On ne peut pas les placer dans la partition. En effet, les tables placées dans la partition sont créées après l'initialisation des opcodes placés dans la section d'en-tête de l'orchestre.

## Exemples

Voici un exemple de l'opode *FLvslidBnk*. Il utilise le fichier *FLvslidBnk.csd* [exemples/FLvslidBnk.csd].

### Exemple 327. Exemple de l'opode *FLvslidBnk*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc    -d      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr = 44100
kr = 441
ksmps = 100
nchnls = 1

gitypetable ftgen 0, 0, 8, -2, 1, 1, 3, 3, 5, 5, 7, 7
giouttable ftgen 0, 0, 8, -2, 0, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8, 1

FLpanel "Slider Bank", 400, 400, 50, 50
;Number of sliders
inum = 8
; Table to store output
iouttable = giouttable
; Width of the slider bank in pixels
iwidth = 350
; Height of the slider in pixels
iheight = 160
; Distance of the left edge of the slider
; from the left edge of the panel
ix = 30
; Distance of the top edge of the slider
```

```

; from the top edge of the panel
iy = 10
; Table containing fader types
itypetable = gitypetable
FLvslidBnk "1@2@3@4@5@6@7@8@9@10@11@12@13@14@15@16", 16 , iouttable , iwidth , iheight , ix \
, iy , itypetable
FLvslidBnk " ", inum , iouttable , iwidth , iheight , ix \
, iy + 200 , -23
; End of panel contents
FLpanelEnd
; Run the widget thread!
FLrun

instr 1
;Dummy instrument
endin

</CsInstruments>
<CsScore>

; Instrument 1 will play a note for 1 hour.
i 1 0 3600
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*FLslider, FLslidBnk*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# FLvslidBnk2

FLvslidBnk2 — Un widget FLTK contenant un banc de réglettes verticales.

## Description

*FLvslidBnk2* est un widget contenant un banc de réglettes verticales.

## Syntaxe

**FLvslidBnk2** "names", inumsliders, ioutable, iconfigtable [,iwidth, iheight, ix, iy, istart\_index]

## Initialisation

« *names* » -- une chaîne de caractères entre guillemets contenant le nom de chaque réglette. Chaque réglette peut avoir un nom différent. Chaque nom est séparé par un caractère « @ », par exemple : « fréquence@amplitude@coupure ». Il est possible de ne fournir aucun nom en donnant un seul espace «   ». Dans ce cas, l'opcode affectera automatiquement un numéro en progression ascendante comme étiquette pour chaque réglette.

*inumsliders* -- le nombre de réglettes.

*ioutable* (facultatif, 0 par défaut) -- numéro d'une table allouée préalablement dans laquelle seront stockées les valeurs de sortie de chaque réglette. Il faut s'assurer que la taille de la table est suffisante pour contenir toutes les cellules de sortie, sinon Csound plantera avec une erreur de segmentation. En affectant zéro à cet argument, la sortie sera dirigée vers l'espace zak dans la zone de taux-k. Dans ce cas, l'espace zak doit avoir été alloué au préalable avec l'opcode *zakinit* et il faut s'assurer que la taille d'allocation est suffisante pour couvrir toutes les réglettes. La valeur par défaut est zéro (c'est-à-dire stockage de la sortie dans l'espace zak).

*iconfigtable* -- dans les opcodes *FLslidBnk2* et *FLvslidBnk2*, cette table remplace *iminmaxtable*, *iexptable* et *istyletable*, tous ces paramètres étant placés dans une seule table. Cette table doit être remplie avec un groupe de quatre paramètres pour chaque réglette de la façon suivante :

min1, max1, exp1, style1, min2, max2, exp2, style2, min3, max3, exp3, style3 etc.

par exemple en utilisant GEN02 on peut taper :

*inum fngen* 1,0,256, -2,   0,1,0,1,   100, 5000, -1, 3,   50, 200, -1, 5,..... [etc]

Dans cet exemple la première réglette reçoit les paramètres [0, 1, 0, 1] (valeurs comprises entre 0 et 1, réponse linéaire, aspect réglette pleine), la seconde réglette reçoit les paramètres [100, 5000, -1, 3] (valeurs comprises entre 100 et 5000, réponse exponentielle, aspect réglette normale), la troisième réglette reçoit les paramètres [50, 200, -1, 5] (valeurs comprises entre 50 et 200, réponse exponentielle, aspect réglette stylée), et ainsi de suite.

*iwidth* (facultatif) -- largeur de la zone rectangulaire contenant toutes les réglettes du banc, à l'exclusion des étiquettes qui sont placées sous cette zone.

*iheight* (facultatif) -- hauteur de la zone rectangulaire contenant toutes les réglettes du banc, à l'exclusion des étiquettes qui sont placées sous cette zone.

*ix* (facultatif) -- position horizontale du coin supérieur gauche de la zone rectangulaire contenant toutes les réglettes appartenant au banc. Il faut laisser suffisamment d'espace en-dessous de ce rectangle afin que



les étiquettes des réglettes soient visibles. En effet, les étiquettes elles-mêmes sont situées à l'extérieur de la zone rectangulaire.

*iy* (facultatif) -- position verticale du coin supérieur gauche de la zone rectangulaire contenant toutes les réglettes appartenant au banc. Il faut laisser suffisamment d'espace en-dessous de ce rectangle afin que les étiquettes des réglettes soient visibles. En effet, les étiquettes elles-mêmes sont situées à l'extérieur de la zone rectangulaire.

*istart\_index* (facultatif, 0 par défaut) -- un nombre entier indiquant un décalage des positions des cellules de sortie. Il peut être positif pour permettre l'allocation en sortie de plusieurs bancs de réglettes dans la même table ou dans l'espace zak. La valeur par défaut est zéro (pas de décalage).

## Exécution

Il n'y a pas d'argument de taux-k, même si les cellules de la table en sortie (ou l'espace zak) sont mis à jour au taux-k.

*FLvslidBnk2* est un widget contenant un banc de réglettes verticales. On peut y mettre n'importe quel nombre de réglettes (argument *inumsliders*). La sortie de toutes les réglettes est stockée dans une table allouée au préalable ou dans l'espace zak (argument *ioutable*). Il est possible de déterminer la première position de la table (ou de l'espace zak) dans laquelle stocker la sortie de la première réglette au moyen de l'argument *istart\_index*.

Chaque réglette peut avoir une étiquette individuelle placée sous elle. Les étiquettes sont définies par l'argument « *names* ». L'intervalle de sortie de chaque réglette peut être fixé individuellement au moyen des valeurs *min* et *max* dans la table *iconfigtable*. La courbe de réponse de chaque réglette peut être fixée individuellement, au moyen d'une liste d'identifiants placés dans la table *iconfigtable* (argument *exp*). Il est possible de définir l'aspect de chaque réglette indépendamment ou de donner le même aspect à toutes les réglettes (argument *style* dans la table *iconfigtable*).

Les arguments *iwidth*, *iheight*, *ix* et *iy* déterminent la largeur, la hauteur, les positions horizontale et verticale de la zone rectangulaire contenant les réglettes. Noter que l'étiquette de chaque réglette est placée en-dessous d'elle et n'est pas incluse dans la zone rectangulaire contenant les réglettes. Ainsi l'utilisateur doit laisser assez d'espace à la gauche du banc en affectant une valeur suffisante à *iy* afin que les étiquettes soient visibles.

*FLvslidBnk2* est identique à *FLslidBnk2* sauf qu'il contient des réglettes verticales plutôt qu'horizontales. Comme la largeur de chaque réglette est souvent petite, il est recommandé de ne laisser qu'un seul espace dans la chaîne de noms (" "), ce qui fait que chaque réglette sera numérotée automatiquement.



### IMPORTANT !

Noter que les tables utilisées par *FLvslidBnk2* doivent être créées avec l'opcode *ftgen* et placées dans l'orchestre avant le valuateur correspondant. On ne peut pas les placer dans la partition. En effet, les tables placées dans la partition sont créées après l'initialisation des opcodes placés dans la section d'en-tête de l'orchestre.

## Voir aussi

*FLslider*, *FLslidBnk*, *FLslidBnk2*, *FLvslidBnk2*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# FLxyin

FLxyin — Détecte la position de curseur de la souris dans une zone définie à l'intérieur d'un FLpanel.

## Description

Semblable à *xyin*, détecte la position de curseur de la souris dans une zone définie à l'intérieur d'un FLpanel.

## Syntaxe

```
koutx, kouty, kinside FLxyin ioutx_min, ioutx_max, iouty_min, iouty_max, \  
iwindx_min, iwindx_max, iwindy_min, iwindy_max [, iexpx, iexpy, ioutx, iouty]
```

## Initialisation

*ioutx\_min, ioutx\_max* - les valeurs limites de l'intervalle de sortie (X ou axe horizontal).

*iouty\_min, iouty\_max* - les valeurs limites de l'intervalle de sortie (Y ou axe vertical).

*iwindx\_min, iwindx\_max* - les coordonnées X des bords horizontaux de la zone sensible, relatives au FLpanel, en pixels.

*iwindy\_min, iwindy\_max* - les coordonnées Y des bords verticaux de la zone sensible, relatives au FLpanel, en pixels.

*iexpx, iexpy* - (facultatif) nombres entiers définissant le comportement des sorties x ou y : 0 -> la sortie est linéaire ; 1 -> la sortie est exponentielle ; tout autre nombre indique le numéro d'une table existante utilisée pour l'indexation. Noter que dans les opérations normales, la table doit être normalisée et unipolaire (tous les éléments de la table doivent être compris entre zéro et un). Dans ce cas, tous les éléments de la table seront mis à l'échelle en fonction de *imin* et de *imax*. Il est tout de même possible d'utiliser des tables non normalisées (créées avec un numéro de table négatif, qui peuvent contenir des éléments de n'importe quelle valeur), afin d'accéder aux valeurs courantes des éléments de la table, sans mise à l'échelle, en affectant 0 à *iout\_min* et 1 à *iout\_max*.

*ioutx, iouty* – (facultatif) valeurs de sortie initiales.

## Exécution

*koutx, kouty* - valeurs de sorties, mises à l'échelle selon les choix de l'utilisateur.

*kinside* - un drapeau indiquant si le curseur de la souris se trouve en dehors du rectangle de la zone définie. S'il est en dehors de la zone, *kinside* vaut zéro.

*FLxyin* détecte la position du curseur de la souris dans une zone définie à l'intérieur d'un *FLpanel*. Quand *FLxyin* est appelé, la position de la souris dans la zone choisie est retournée au taux-k. Il est possible de définir la zone sensible, ainsi que les valeurs minimale et maximale correspondant aux positions minimale et maximale de la souris. Il n'est pas nécessaire que les boutons de la souris soient appuyés pour que *FLxyin* fonctionne. Il est capable d'opérer correctement même si d'autres widgets (présents dans le *FLpanel*) chevauchent la zone sensible.

A l'inverse de la plupart des autres opcodes FLTK, *FLxyin* ne peut pas être utilisé dans l'en-tête, car ce n'est pas un widget. Ce n'est que la définition d'une zone de détection de la souris à l'intérieur d'un panneau FLTK.

## Exemples

Voici un exemple de l'opcode FLxyin. Il utilise le fichier *FLxyin.csd* [examples/FLxyin.csd].

### Exemple 328. Exemple de l'opcode FLxyin.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=48000
ksmps=128
nchnls=2

; Example by Andres Cabrera 2007

FLpanel "FLxyin", 200, 100, -1, -1, 3
FLpanelEnd
FLrun

instr 1
  koutx, kouty, kinside FLxyin 0, 10, 100, 1000, 10, 190, 10, 90
  aout buzz 10000, kouty, koutx, 1
  printk2 koutx
  outs aout, aout
endin

</CsInstruments>

<CsScore>
f 1 0 1024 10 1
i 1 0 3600

e

</CsScore>
</CsoundSynthesizer>
```

Voici une autre exemple de l'opcode FLxyin. Il utilise le fichier *FLxyin-2.csd* [examples/FLxyin-2.csd].

### Exemple 329. Exemple de l'opcode FLxyin.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac          -iadc      -d      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=44100
kr=441
ksmps=100
```

```

nchnls=2

; Example by Gabriel Maldonado

FLpanel "Move the mouse inside this panel to hear the effect",400,400
FLpanel_end
FLrun

instr 1

k1, k2, kinside FLxyin 50, 1000, 50, 1000, 100, 300, 50, 250, -2,-3
; if k1 <= 50 || k1 >=5000 || k2 <=100 || k2 >= 8000 kgoto end ; if cursor is outside bounds, then don't

a1 oscili 3000, k1, 1
a2 oscili 3000, k2, 1

outs a1,a2
printk2 k1
printk2 k2, 10
printk2 kinside, 20
end:
endin

</CsInstruments>
<CsScore>

f1 0 1024 10 1
f2 0 17 19 1 1 90 1
f3 0 17 19 2 1 90 1
i1 0 3600

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*FLpanel*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# fmanal

fmanal — Analyse MA/MF d'un signal en quadrature.

## Description

Cet opcode tente d'extraire les signaux MA et MF d'un signal en quadrature (par exemple d'une transformée de Hilbert).

## Syntaxe

am, af **fmanal** are, aim

## Exécution

*are* -- signal réel en entrée (phase cosinus)

*aim* -- signal imaginaire en entrée (phase sinus)

*am* -- enveloppe de la modulation d'amplitude.

*af* -- enveloppe de la modulation de fréquence.

*fmanal* prend un signal en quadrature (parties réelles et imaginaires) et restitue les signaux de MA et de MF estimés. Le premier est le module de chaque paire en entrée ( $|re + j*im|$ ) et le second est la dérivée d' $\arg(re + j*im)$ . Chaque échantillon en sortie est l'amplitude et la fréquence instantanées estimées du signal d'entrée.

## Exemple

Voici un exemple de l'opcode fmanal. Il utilise le fichier *fmanal.csd* [examples/fmanal.csd].

### Exemple 330. Exemple de l'opcode fmanal.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 64
nchnls = 2
odbfs = 1

instr 1

asig oscili p4, p5
a1,a2 hilbert2 asig,1024,256
am,afm fmanal a1,a2
ktrig metro 2
printf "AM=%.3f FM=%.1f\n",ktrig,k(am),k(afm)
outs a1, a2
```

```
endin
```

```
</CsInstruments>  
<CsScore>  
i1 0 10 0.5 440  
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
2017

# fmax

fmax — Fonction valeur maximale.

## Description

Retourne le maximum de ses deux arguments.

## Syntaxe

```
iRes[] fmax iArg1[], iArg2[]  
kRes[] fmax kArg1[], kArg2[]  
iRes[] fmax iArg1[], iArg2  
kRes[] fmax kArg[], kArg2
```

## Initialisation

*iArg[]1/2, iArg2* -- les opérandes.

## Exécution

*kArg[]1/2, kArg2* -- les opérandes.

## Exemples

Voici un exemple de l'opcode fmax. Il utilise le fichier *fmax.csd* [examples/fmax.csd].

### Exemple 331. Exemple de l'opcode fmax.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
</CsOptions>  
<CsInstruments>  
  
instr 1  
  
iArg1[] fillarray 1,8,3  
iArg2[] fillarray 4,5,6  
iRes[] fmax iArg1,iArg2  
ik init 0  
  
while ik < lenarray(iRes) do  
  print iRes[ik]  
  ik += 1  
od  
  
endin
```



```
</CsInstruments>  
<CsScore>  
i1 0 0  
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
2017

# fmb3

fmb3 — Utilise la synthèse FM pour créer un son d'orgue Hammond B3.

## Description

Utilise la synthèse FM pour créer un son d'orgue Hammond B3. Il provient d'une famille de sons FM qui utilisent tous 4 oscillateurs élémentaires et diverses architectures, comme dans le synthétiseur TX81Z.

## Syntaxe

```
ares fmb3 kamp, kfreq, kc1, kc2, kvdepth, kvrate[, ifn1, ifn2, ifn3, \
      ifn4, ivfn]
```

## Initialisation

*fmb3* prend 5 tables pour l'initialisation. Les 4 premières sont les entrées de base et la dernière est l'oscillateur basse fréquence (LFO) utilisé pour le vibrato. La dernière table contiendra habituellement une onde sinus.

Les formes d'onde initiales seront :

- *ifn1* -- onde sinus
- *ifn2* -- onde sinus
- *ifn3* -- onde sinus
- *ifn4* -- onde sinus

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée.

*kc1*, *kc2* -- Contrôles pour le synthétiseur :

- *kc1* -- Indice de modulation total
- *kc2* -- Fondu des deux modulateurs
- *Algorithme* -- 4

*kvdepth* -- Largeur du vibrato

*kvrate* -- Vitesse du vibrato

## Exemples

Voici un exemple de l'opcode fmb3. Il utilise le fichier *fmb3.csd* [examples/fmb3.csd].

### Exemple 332. Exemple de l'opcode fmb3.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fmb3.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kfreq = 220
kc1 = p4
kc2 = p5
kvrata = 6

kvdpth line 0, p3, p6
asig fmb3 .4, kfreq, kc1, kc2, kvdpth, kvrate, 1, 1, 1, 1, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
;sine wave.
f 1 0 32768 10 1

i 1 0 2 5 5 0.1
i 1 3 2 .5 .5 0.01
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*fmbell, fmmetal, fmpercfl, fmrhode, fmwurlie*

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
University of Bath, Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.47 de Csound

# fmbell

fmbell — Utilise la synthèse FM pour créer un son de cloche tube.

## Description

Utilise la synthèse FM pour créer un son de cloche tube. Il provient d'une famille de sons FM qui utilisent tous 4 oscillateurs élémentaires et diverses architectures, comme dans le synthétiseur TX81Z.

## Syntaxe

```
ares fmbell kamp, kfreq, kc1, kc2, kvdepth, kvrate[, ifn1, ifn2, ifn3, \
      ifn4, ivfn, isus]
```

## Initialisation

Tous ces opcodes prennent 5 tables pour l'initialisation. Les 4 premières sont les entrées de base et la dernière est l'oscillateur basse fréquence (LFO) utilisé pour le vibrato. La dernière table contiendra habituellement une onde sinus.

Les formes d'onde initiales seront :

- *ifn1* -- onde sinus
- *ifn2* -- onde sinus
- *ifn3* -- onde sinus
- *ifn4* -- onde sinus

L'argument facultatif *isus* contrôle la durée du son, ou sa vitesse de décroissance. Il vaut 4 par défaut.

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée.

*kc1*, *kc2* -- Contrôles pour le synthétiseur :

- *kc1* -- Indice de modulation 1
- *kc2* -- Fondu des deux sorties
- *Algorithme* -- 5

*kvdepth* -- Largeur du vibrato

*kvrate* -- Vitesse du vibrato

## Exemples

Voici un exemple de l'opcode fmbell. Il utilise le fichier *fmbell.csd* [examples/fmbell.csd].

### Exemple 333. Exemple de l'opcode fmbell.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fmbell.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kamp = p4
kfreq = 880
kc1 = p5
kc2 = p6
kvdepth = 0.005
kvrate = 6

asig fmbell kamp, kfreq, kc1, kc2, kvdepth, kvrate, 1, 1, 1, 1, 1
outs asig, asig
endin

instr 2

kamp = p4
kfreq = 880
kc1 = p5
kc2 = p6
kvdepth = 0.005
kvrate = 6

asig fmbell kamp, kfreq, kc1, kc2, kvdepth, kvrate, 1, 1, 1, 1, 1, p7
outs asig, asig
endin

</CsInstruments>
<CsScore>
; sine wave.
f 1 0 32768 10 1

i 1 0 3 .2 5 5
i 1 + 4 .3 .5 1
s
i 2 0 12 .2 5 5 16
i 2 + 12 .3 .5 1 12

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*fmb3, fmmetal, fmpercfl, fmrhode, fmwurlie*

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
University of Bath, Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.47 de Csound.

Argument facultatif ajouté dans la version 5.16.

Plus d'arguments sont facultatifs depuis la version 6.0

# fmin

fmin — Fonction valeur minimale.

## Description

Retourne le minimum de ses deux arguments.

## Syntaxe

```
iRes[] fmin iArg1[], iArg2[]  
kRes[] fmin kArg1[], kArg2[]  
iRes[] fmin iArg1[], iArg2  
kRes[] fmin kArg[], kArg2
```

## Initialisation

*iArg[]1/2, iArg2* -- les opérandes.

## Exécution

*kArg[]1/2, kArg2* -- les opérandes.

## Exemples

Voici un exemple de l'opcode fmin. Il utilise le fichier *fmin.csd* [examples/fmin.csd].

### Exemple 334. Exemple de l'opcode fmin.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
</CsOptions>  
<CsInstruments>  
  
instr 1  
  
iArg1[] fillarray 1,8,3  
iArg2[] fillarray 4,5,6  
iRes[] fmin iArg1,iArg2  
ik init 0  
  
while ik < lenarray(iRes) do  
  print iRes[ik]  
  ik += 1  
od  
  
endin
```

```
</CsInstruments>  
<CsScore>  
i1 0 0  
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
2017



# fmmetal

fmmetal — Utilise la synthèse FM pour créer un son de « Heavy Metal ».

## Description

Utilise la synthèse FM pour créer un son de « Heavy Metal ». Il provient d'une famille de sons FM qui utilisent tous 4 oscillateurs élémentaires et diverses architectures, comme dans le synthétiseur TX81Z.

## Syntaxe

```
ares fmmetal kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \  
      ifn4, ivfn
```

## Initialisation

Tous ces opcodes prennent 5 tables pour l'initialisation. Les 4 premières sont les entrées de base et la dernière est l'oscillateur basse fréquence (LFO) utilisé pour le vibrato. La dernière table contiendra habituellement une onde sinus.

Les formes d'onde initiales seront :

- *ifn1* -- onde sinus
- *ifn2* -- *twopeaks.aiff* [examples/twopeaks.aiff]
- *ifn3* -- *twopeaks.aiff* [examples/twopeaks.aiff]
- *ifn4* -- onde sinus



### Note

Le fichier « twopeaks.aiff » est aussi disponible à <ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/>.

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée.

*kc1*, *kc2* -- Contrôles pour le synthétiseur :

- *kc1* -- Indice de modulation total
- *kc2* -- Fondu des deux modulateurs
- *Algorithme* -- 3

*kvdepth* -- Largeur du vibrato

*kvrate* -- Vitesse du vibrato

## Exemples

Voici un exemple de l'opcode *fmmetal*. Il utilise les fichiers *fmmetal.csd* [examples/fmmetal.csd] et *two-peaks.aiff* [examples/twopeaks.aiff].

### Exemple 335. Exemple de l'opcode *fmmetal*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fmmetal.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kfreq = 440
kvdepth = 0
kvrate = 0
ifn1 = 1
ifn2 = 2
ifn3 = 2
ifn4 = 1
ivfn = 1
kc2 = p5

kc1 line p4, p3, 1
asig fmmetal .5, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn
outs asig, asig

endin
</CsInstruments>
<CsScore>
; sine wave.
f 1 0 32768 10 1
; the "twopeaks.aiff" audio file.
f 2 0 256 1 "twopeaks.aiff" 0 0 0

i 1 0 4 6 5
i 1 5 4 .2 10
e
</CsScore>
</CsoundSynthesizer>
```

## voir Aussi

*fmb3*, *fmbell*, *fmpercfl*, *fmrhode*, *fmwurlie*

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
University of Bath, Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.47 de Csound

# fmod

fmod — Calcule le reste d'une division en virgule flottante.

## Description

Calcule le reste de la division de son premier argument par le second.

## Syntaxe

```
iRes[] fmod iarg1[], iarg2[]  
kres[] fmod karg1[], karg2[]  
iRes[] fmod iarg1[], iarg2  
kres[] fmod karg[], karg2
```

## Initialisation

*iarg1*[]/2, *iarg2* -- les opérandes.

## Exécution

*karg1*[]/2, *karg2* -- les opérandes.

## Exemples

Voici un exemple de l'opcode fmod. Il utilise le fichier *fmod.csd* [examples/fmod.csd].

### Exemple 336. Exemple de l'opcode fmod.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
</CsOptions>  
<CsInstruments>  
  
instr 1  
  
iArg1[] fillarray 10,8,22  
iArg2[] fillarray 4,5,6  
iRes[] fmod iArg1,iArg2  
ik init 0  
  
while ik < lenarray(iRes) do  
  print iRes[ik]  
  ik += 1  
od  
  
endin
```

```
</CsInstruments>  
<CsScore>  
i1 0 0  
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
2017

# fmpercfl

fmpercfl — Utilise la synthèse FM pour créer un son de flûte percussive.

## Description

Utilise la synthèse FM pour créer un son de flûte percussive. Il provient d'une famille de sons FM qui utilisent tous 4 oscillateurs élémentaires et diverses architectures, comme dans le synthétiseur TX81Z.

## Syntaxe

```
ares fmpercfl kamp, kfreq, kc1, kc2, kvdepth, kvrate[, ifn1, ifn2, \
    ifn3, ifn4, ivfn]
```

## Initialisation

Tous ces opcodes prennent 5 tables pour l'initialisation. Les 4 premières sont les entrées de base et la dernière est l'oscillateur basse fréquence (LFO) utilisé pour le vibrato. La dernière table contiendra habituellement une onde sinus.

Les formes d'onde initiales seront :

- *ifn1* -- onde sinus
- *ifn2* -- onde sinus
- *ifn3* -- onde sinus
- *ifn4* -- onde sinus

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée.

*kc1*, *kc2* -- Contrôles pour le synthétiseur :

- *kc1* -- Indice de modulation total
- *kc2* -- Fondu des deux modulateurs
- *Algorithme* -- 4

*kvdepth* -- Largeur du vibrato

*kvrate* -- Vitesse du vibrato

## Exemples

Voici un exemple de l'opcode fmpercfl. Il utilise le fichier *fmpercfl.csd* [examples/fmpercfl.csd].

### Exemple 337. Exemple de l'opcode `fmpercfl`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fmpercfl.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kfreq = 220
kc1 = 5
kvdepth = .01
kvrate = 6

kc2 line 5, p3, p4
asig fmpercfl .5, kfreq, kc1, kc2, kvdepth, kvrate, 1, 1, 1, 1
outs asig, asig
endin

</CsInstruments>
<CsScore>
; sine wave.
f 1 0 32768 10 1

i 1 0 4 5
i 1 5 8 .1

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*fmb3, fmbell, fmmetal, fmrhode, fmwurlie*

## Crédits

Auteur : John ffitich (d'après Perry Cook)  
University of Bath, Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.47 de Csound

# fmrhode

fmrhode — Utilise la synthèse FM pour créer un son de piano électrique Fender Rhodes.

## Description

Utilise la synthèse FM pour créer un son de piano électrique Fender Rhodes. Il provient d'une famille de sons FM qui utilisent tous 4 oscillateurs élémentaires et diverses architectures, comme dans le synthétiseur TX81Z.

## Syntaxe

```
ares fmrhode kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, \  
ifn3, ifn4, ivfn
```

## Initialisation

Tous ces opcodes prennent 5 tables pour l'initialisation. Les 4 premières sont les entrées de base et la dernière est l'oscillateur basse fréquence (LFO) utilisé pour le vibrato. La dernière table contiendra habituellement une onde sinus.

Les formes d'onde initiales seront :

- *ifn1* -- onde sinus
- *ifn2* -- onde sinus
- *ifn3* -- onde sinus
- *ifn4* -- *fwavblnk.aiff* [examples/fwavblnk.aiff]



### Note

Le fichier « fwavblnk.aiff » est aussi disponible à <ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/>.

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée.

*kc1*, *kc2* -- Contrôles pour le synthétiseur :

- *kc1* -- Indice de modulation 1
- *kc2* -- Fondu des deux sorties
- *Algorithme* -- 5

*kvdepth* -- Largeur du vibrato



*kvrate* -- Vitesse du vibrato

## Exemples

Voici un exemple de l'opcode *fmrhode*. Il utilise les fichiers *fmrhode.csd* [examples/fmrhode.csd] et *fwavblnk.aiff* [examples/fwavblnk.aiff].

### Exemple 338. Exemple de l'opcode *fmrhode*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fmrhode.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kfreq = 220
kc1 = p4
kc2 = p5
kvdepth = 0.01
kvrate = 3
ifn1 = 1
ifn2 = 1
ifn3 = 1
ifn4 = 2
ivfn = 1

asig fmrhode .5, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn
outs asig, asig

endin
</CsInstruments>
<CsScore>
; sine wave.
f 1 0 32768 10 1
; audio file.
f 2 0 256 1 "fwavblnk.aiff" 0 0 0

i 1 0 3 6 0
i 1 + . 6 3
i 1 + . 20 0
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*fmb3*, *fmbell*, *fmmetal*, *fmpercfl*, *fmwurlie*

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
University of Bath, Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.47 de Csound

# fmvoice

fmvoice — Synthèse FM d'une Voix de Chanteur

## Description

Synthèse FM d'une Voix de Chanteur

## Syntaxe

```
ares fmvoice kamp, kfreq, kvowel, ktilt, kvibamt, kvibrate[, ifn1, \
    ifn2, ifn3, ifn4, ivibfn]
```

## Initialisation

*ifn1, ifn2, ifn3, ifn4* -- Tables, normalement des formes d'onde sinus.

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée.

*kvowel* -- La voyelle chantée, dans l'intervalle 0-64

*ktilt* -- La pente spectrale du son dans l'intervalle 0 à 99

*kvibamt* -- Largeur du vibrato

*kvibrate* -- Vitesse du vibrato

## Exemples

Voici un exemple de l'opcode fmvoice. Il utilise le fichier *fmvoice.csd* [examples/fmvoice.csd].

### Exemple 339. Exemple de l'opcode fmvoice.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fmvoice.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
```

```

0dbfs = 1

instr 1

kfreq = 110
kvowel = p4 ; p4 = vowel (0 - 64)
ktilt = p5
kvibamt = 0.005
kvibrate = 6

asig fmvoice .5, kfreq, kvowel, ktilt, kvibamt, kvibrate, 1, 1, 1, 1, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
; sine wave.
f 1 0 16384 10 1

i 1 0 1 1 0 ; tilt=0
i 1 1 1 > .
i 1 2 1 > .
i 1 3 1 > .
i 1 4 1 > .
i 1 5 1 > .
i 1 6 1 > .
i 1 7 1 12 .

i 1 10 1 1 90 ; tilt=90
i 1 11 1 > .
i 1 12 1 > .
i 1 13 1 > .
i 1 14 1 > .
i 1 15 1 > .
i 1 16 1 > .
i 1 17 1 12 .

e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
 University of Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 3.47 de Csound

# fmwurlie

fmwurlie — Utilise la synthèse FM pour créer un son de piano électrique Wurlitzer.

## Description

Utilise la synthèse FM pour créer un son de piano électrique Wurlitzer. Il provient d'une famille de sons FM qui utilisent tous 4 oscillateurs élémentaires et diverses architectures, comme dans le synthétiseur TX81Z.

## Syntaxe

```
ares fmwurlie kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \  
      ifn4, ivfn
```

## Initialisation

Tous ces opcodes prennent 5 tables pour l'initialisation. Les 4 premières sont les entrées de base et la dernière est l'oscillateur basse fréquence (LFO) utilisé pour le vibrato. La dernière table contiendra habituellement une onde sinus.

Les formes d'onde initiales seront :

- *ifn1* -- onde sinus
- *ifn2* -- onde sinus
- *ifn3* -- onde sinus
- *ifn4* -- *fwavblnk.aiff* [examples/fwavblnk.aiff]



### Note

Le fichier « fwavblnk.aiff » est aussi disponible à <ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/>.

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée.

*kc1*, *kc2* -- Contrôles pour le synthétiseur :

- *kc1* -- Indice de modulation 1
- *kc2* -- Fondu des deux sorties
- *Algorithme* -- 5

*kvdepth* -- Largeur du vibrato

*kvrate* -- Vitesse du vibrato

## Exemples

Voici un exemple de l'opcode *fmwurlie*. Il utilise les fichiers *fmwurlie.csd* [examples/fmwurlie.csd] et *fwavblnk.aiff* [examples/fwavblnk.aiff].

### Exemple 340. Exemple de l'opcode *fmwurlie*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fmwurlie.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kfreq = 440
kc1 = p4
kc2 = 1
kvdepth = 0.05
kvrate = 6
ifn1 = 1
ifn2 = 1
ifn3 = 1
ifn4 = 2
ivfn = 1

asig fmwurlie .5, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn
outs asig, asig

endin
</CsInstruments>
<CsScore>
; sine wave
f 1 0 32768 10 1
; audio file
f 2 0 256 1 "fwavblnk.aiff" 0 0 0

i 1 0 3 6
i 1 + 3 30
i 1 + 2 60
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*fmb3*, *fmbell*, *fmmetal*, *fmpercfl*, *fmrhode*

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
University of Bath, Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.47 de Csound

# fof

fof — Produit des grains FOF (sinusoïde amortie) pour la synthèse par formant et la synthèse granulaire.

## Description

La sortie audio est une succession de grains FOF (fonction d'onde formantique) amorcés à la fréquence *xfund* avec une pointe spectrale à *xform*. Pour *xfund* supérieur à 25 Hz ces grains produisent un formant comme dans la parole avec des caractéristiques spectrales déterminées par les paramètres d'entrée de taux-*k*. Pour des fondamentales plus basses ce générateur fournit une forme spéciale de synthèse granulaire.

## Syntaxe

```
ares fof xamp, xfund, xform, koct, kband, kris, kdur, kdec, iolaps, \  
      ifna, ifnb, itotdur [, iphs] [, ifmode] [, iskip]
```

## Initialisation

*iolaps* -- quantité de mémoire préallouée nécessaire pour contenir les données de chevauchement des grains. Les chevauchements dépendent de la fréquence, et l'espace requis dépend de la valeur maximale de *xfund* \* *kdur*. La surestimation de cet espace n'induit pas de coût de calcul supplémentaire. Chaque *iolap* utilise moins de 50 octets de mémoire.

*ifna*, *ifnb* -- numéro de table de deux fonctions. La première est une table sinus pour la synthèse des grains FOF (une taille d'au moins 4096 est recommandée). La seconde est une forme ascendante, utilisée à l'endroit et à l'envers pour dessiner l'attaque et la chute des grains FOF ; cette forme peut être linéaire (*GEN07*) ou bien sigmoïde (*GEN19*).

*itotdur* -- durée totale durant laquelle ce *fof* sera actif. Fixée normalement à p3. Aucun nouveau grain FOF n'est créé si son *kdur* n'est pas contenu complètement dans le *itotdur* restant.

*iphs* (facultatif, par défaut 0) -- phase initiale du fondamental, exprimée comme une fraction d'une période (0 à 1). La valeur par défaut est 0.

*ifmode* (facultatif, par défaut 0) -- mode fréquentiel du formant. S'il est nul, chaque grain FOF garde la fréquence *xform* avec laquelle il a été amorcé. Sinon, chaque grain FOF est influencé par *xform* en continu. La valeur par défaut est 0.

*iskip* (facultatif, par défaut 0) -- s'il est non nul, l'initialisation est ignorée (ce qui permet l'utilisation du legato).

## Exécution

*xamp* -- amplitude de crête de chaque grain FOF, observée à la toute fin de son attaque. L'attaque pourra dépasser cette valeur si l'on a une grande largeur de bande (disons  $Q < 10$ ) et/ou quand les grains FOF se superposent en partie.

*xfund* -- la fréquence fondamentale (en Hertz) des impulsions qui créent les nouveaux grains FOF.

*xform* -- la fréquence du formant, c'est-à-dire la fréquence du grain FOF induit par chaque impulsion *xfund*. Cette fréquence peut être fixe pour chaque grain ou varier en continu (voir *ifmode*).



*koct* -- indice d'octaviation, normalement zéro. S'il est supérieur à zéro, il abaisse la fréquence *xfund* effective en atténuant les grains FOF de rang impair. Les nombres entiers sont des octaves, les fractions sont transitoires.

*kband* -- la largeur de bande du formant (à -6dB), exprimée en Hz. La largeur de bande détermine la vitesse de décroissance exponentielle du grain FOF, avant l'application de l'enveloppe décrite ci-dessous.

*kris*, *kdur*, *kdec* -- attaque, durée globale et chute (en secondes) du grain FOF. Ces valeurs appliquent une enveloppe à chaque grain, à la manière du générateur de Csound *linen* mais avec des formes d'attaque et de chute dessinées à partir de l'entrée *ifnb*. *kris* détermine en proportion inverse la largeur de jupe (à -40 dB) de la région formantique induite. *kdur* affecte la densité des chevauchements des grains FOF, et par conséquent la vitesse de calcul. Des valeurs typiques pour une imitation vocale sont 0,003, 0,02, 0,007.

Le générateur *fof* de Csound est inspiré du codage en C par Michael Clarke du programme *CHANT* de l'IRCAM (Xavier Rodet et al.). Chaque *fof* produit un seul formant, et les sorties de quatre ou plus de ceux-ci peuvent être additionnées pour produire une riche imitation vocale. La synthèse *fof* est une forme spéciale de la synthèse granulaire, et cette implémentation facilite la transformation entre l'imitation vocale et les textures granulaires. La vitesse de calcul dépend de *kdur*, *xfund*, et de la densité des chevauchements.

## Exemples

Voici un exemple de l'opcode *fof*. Il utilise le fichier *fof.csd* [examples/fof.csd].

### Exemple 341. Exemple de l'opcode *fof*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fof.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

instr 1
; Combine five formants together to create
; a transformation from an alto-"a" sound
; to an alto-"i" sound.
; Values common to all of the formants.
kfund init 261.659
koct init 0
kris init 0.003
kdur init 0.02
kdec init 0.007
iolaps = 100
ifna = 1
ifnb = 2
itotdur = p3

; First formant.
klamp = ampdb(0)
klform line 800, p3, 350
```

```

k1band line 80, p3, 50

; Second formant.
k2amp line ampdb(-4), p3, ampdb(-20)
k2form line 1150, p3, 1700
k2band line 90, p3, 100

; Third formant.
k3amp line ampdb(-20), p3, ampdb(-30)
k3form line 2800, p3, 2700
k3band init 120

; Fourth formant.
k4amp init ampdb(-36)
k4form line 3500, p3, 3700
k4band line 130, p3, 150

; Fifth formant.
k5amp init ampdb(-60)
k5form init 4950
k5band line 140, p3, 200

a1 fof k1amp, kfund, k1form, koct, k1band, kris, \
    kdur, kdec, iolaps, ifna, ifnb, itotdur
a2 fof k2amp, kfund, k2form, koct, k2band, kris, \
    kdur, kdec, iolaps, ifna, ifnb, itotdur
a3 fof k3amp, kfund, k3form, koct, k3band, kris, \
    kdur, kdec, iolaps, ifna, ifnb, itotdur
a4 fof k4amp, kfund, k4form, koct, k4band, kris, \
    kdur, kdec, iolaps, ifna, ifnb, itotdur
a5 fof k5amp, kfund, k5form, koct, k5band, kris, \
    kdur, kdec, iolaps, ifna, ifnb, itotdur

; Combine all of the formants together
asig sum (a1+a2+a3+a4+a5) * 13000
outs asig, asig

endin
</CsInstruments>
<CsScore>
; sine wave
f 1 0 4096 10 1
; sigmoid wave
f 2 0 1024 19 0.5 0.5 270 0.5

i 1 0 1
i 1 2 5 ; same but slower
e
</CsScore>
</CsoundSynthesizer>

```

Les valeurs de formant pour le "a" en voix d'alto proviennent de l'*Appendice Valeurs de Formant*.

## Voir aussi

*fof2, Appendice Valeurs de Formant*

## Crédits

Ajouté dans la version 1 (1990)

# fof2

*fof2* — Produit des grains FOF (sinusoïde amortie) incluant une indexation incrémentielle de taux-k avec chaque grain.

## Description

La sortie audio est une succession de grains FOF (fonction d'onde formantique) amorcés à la fréquence *xfund* avec une pointe spectrale à *xform*. Pour *xfund* supérieur à 25 Hz ces grains produisent un formant comme dans la parole avec des caractéristiques spectrales déterminées par les paramètres d'entrée de taux-k. Pour des fondamentales plus basses ce générateur fournit une forme spéciale de synthèse granulaire.

*fof2* implémente une indexation incrémentielle de taux-k dans la fonction *ifna* avec chaque grain successif.

## Syntaxe

```
ares fof2 xamp, xfund, xform, koct, kband, kris, kdur, kdec, iolaps, \  
      ifna, ifnb, itotdur, kphs, kgliss [, iskip]
```

## Initialisation

*iolaps* -- quantité de mémoire préallouée nécessaire pour contenir les données de chevauchement des grains. Les chevauchements dépendent de la fréquence, et l'espace requis dépend de la valeur maximale de *xfund* \* *kdur*. La surestimation de cet espace n'induit pas de coût de calcul supplémentaire. Chaque *iolap* utilise moins de 50 octets de mémoire.

*ifna*, *ifnb* -- numéro de table de deux fonctions. La première est une table sinus pour la synthèse des grains FOF (une taille d'au moins 4096 est recommandée). La seconde est une forme ascendante, utilisée à l'endroit et à l'envers pour dessiner l'attaque et la chute des grains FOF ; cette forme peut être linéaire (*GEN07*) ou bien sigmoïde (*GEN19*).

*itotdur* -- durée totale durant laquelle ce *fof* sera actif. Fixée normalement à p3. Aucun nouveau grain FOF n'est créé si son *kdur* n'est pas contenu complètement dans le *itotdur* restant.

*iskip* (facultatif, par défaut 0) -- s'il est non nul, l'initialisation est ignorée (ce qui permet l'utilisation du legato).

## Exécution

*xamp* -- amplitude de crête de chaque grain FOF, observée à la toute fin de son attaque. L'attaque pourra dépasser cette valeur si l'on a une grande largeur de bande (disons  $Q < 10$ ) et/ou quand les grains FOF se superposent en partie.

*xfund* -- la fréquence fondamentale (en Hertz) des impulsions qui créent les nouveaux grains FOF.

*xform* -- la fréquence du formant, c'est-à-dire la fréquence du grain FOF induit par chaque impulsion *xfund*. Cette fréquence peut être fixe pour chaque grain ou varier en continu.

*koct* -- indice d'octaviation, normalement zéro. S'il est supérieur à zéro, il abaisse la fréquence *xfund* effective en atténuant les grains FOF de rang impair. Les nombres entiers sont des octaves, les fractions sont transitoires.

*kband* -- la largeur de bande du formant (à -6dB), exprimée en Hz. La largeur de bande détermine la vitesse de décroissance exponentielle du grain FOF, avant l'application de l'enveloppe décrite ci-dessous.

*kris*, *kdur*, *kdec* -- attaque, durée globale et chute (en secondes) du grain FOF. Ces valeurs appliquent une enveloppe à chaque grain, à la manière du générateur de Csound *linen* mais avec des formes d'attaque et de chute dessinées à partir de l'entrée *ifnb*. *kris* détermine en proportion inverse la largeur de jupe (à -40 dB) de la région formantique induite. *kdur* affecte la densité des chevauchements des grains FOF, et par conséquent la vitesse de calcul. Des valeurs typiques pour une imitation vocale sont 0,003, 0,02, 0,007.

*kphs* -- permet d'indexer au taux-k la table de fonction *ifna* avec chaque grain successif, ce qui permet d'appliquer le recalage temporel. Les valeurs de *kphs* sont normalisées entre 0 et 1, 1 étant la fin de la table de fonction *ifna*.

*kgliss* -- fixe la hauteur finale de chaque grain en fonction de sa hauteur initiale, en octaves. Ainsi *kgliss* = 2 signifie que le grain se termine deux octaves plus haut que sa hauteur initiale, tandis qu'avec *kgliss* = -3/4 le grain se termine une sixte majeure plus bas. Chaque 1/12 ajouté à *kgliss* élève la hauteur finale d'un demi-ton. Si vous ne voulez pas de glissando, fixez *kgliss* à 0.

Le générateur *fof* de Csound est inspiré du codage en C par Michael Clarke du programme *CHANT* de l'IRCAM (Xavier Rodet et al.). Chaque *fof* produit un seul formant, et les sorties de quatre ou plus de ceux-ci peuvent être additionnées pour produire une riche imitation vocale. La synthèse *fof* est une forme spéciale de la synthèse granulaire, et cette implémentation facilite la transformation entre l'imitation vocale et les textures granulaires. La vitesse de calcul dépend de *kdur*, *xfund*, et de la densité des chevauchements.



### Note

La fréquence finale de chaque grain étant égale à  $kform * (2 ^ kgliss)$ , des valeurs trop importantes de *kgliss* pourront provoquer un repliement. Par exemple, *kform* = 3000 et *kgliss* = 3 placent la fréquence finale au-delà de la fréquence de Nyquist si *sr* = 44100. Il est prudent de pondérer *kgliss* en conséquence.

## Exemples

Voici un exemple de l'opcode *fof2*. Il utilise le fichier *fof2.csd* [examples/fof2.csd].

### Exemple 342. Exemple de l'opcode *fof2*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fof2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2

;By Andres Cabrera 2007

instr 1                ;table-lookup vocal synthesis

kris init p12
kdur init p13
kdec init p14
```

```

iolaps init p15

ifna init 1 ; Sine wave
ifnb init 2 ; Straight line rise shape

itotdur init p3

kphs init 0 ; No phase modulation (constant kphs)

kfund line p4, p3, p5
kform line p6, p3, p7
koct line p8, p3, p9
kband line p10, p3, p11
kgliss line p16, p3, p17

kenv linen 5000, 0.03, p3, 0.03 ;to avoid clicking

aout fof2 kenv, kfund, kform, koct, kband, kris, kdur, kdec, iolaps, \
      ifna, ifnb, itotdur, kphs, kgliss

      outs aout, aout
      endin

</CsInstruments>
<CsScore>
f1 0 8192 10 1
f2 0 4096 7 0 4096 1

;          kfund1 kfund2 kform1 kform2 koct1 koct2 kband1 kband2 kris kdur kdec iolaps kg
i1 0 4 220 220 510 510 0 0 30 30 0.01 0.03 0.01 20
i1 + . 220 220 510 910 0 0 30 30 0.01 0.03 0.01 20
i1 + . 220 220 510 510 0 0 100 30 0.01 0.03 0.01 20
i1 + . 220 220 510 510 0 1 30 30 0.01 0.03 0.01 20
i1 + . 220 220 510 510 0 0 30 30 0.01 0.03 0.01 20
i1 + . 220 220 510 510 0 0 30 30 0.01 0.03 0.01 20
i1 + . 220 220 510 510 0 0 30 30 0.01 0.05 0.01 100
i1 + . 220 440 510 510 0 0 30 30 0.01 0.05 0.01 100

e

</CsScore>
</CsoundSynthesizer>

```

Voici un autre exemple de l'opcode fof2, qui produit des sons de voyelle en utilisant des formants générés par fof2 avec les valeurs de l'appendice *Valeurs de Formant*. Il utilise le fichier *fof2-2.csd* [examples/fof2-2.csd].

### Exemple 343. Exemple de l'opcode fof2 pour produire des sons de voyelle.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out Audio in
-odac -iadc ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fof2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2

```

*; Example by Chuckk Hubbard 2007*

```

instr 1          ;table-lookup vocal synthesis

iolaps    =    120
ifna      =    1          ;f1 - sine wave
ifnb      =    2          ;f2 - linear rise shape
itotdur   =    p3
iamp      =    p4 * 0dbfs
ifreq1    =    p5          ;starting frequency
ifreq2    =    p6          ;ending frequency

kamp      linseg    0, .003, iamp, itotdur-.007, iamp, .003, 0, .001, 0
kfund     expseg    ifreq1, itotdur, ifreq2
kocf      init     0
kris      init     .003
kdur      init     .02
kdec      init     .007
kphs      init     0
kgliss    init     0

iforma    =    p7          ;starting spectrum
iformb    =    p8          ;ending spectrum

iform1a   tab_i     0, iforma          ;read values of 5 formants for 1st spectrum
iform2a   tab_i     1, iforma
iform3a   tab_i     2, iforma
iform4a   tab_i     3, iforma
iform5a   tab_i     4, iforma
idb1a     tab_i     5, iforma          ;read decibel levels for same 5 formants
idb2a     tab_i     6, iforma
idb3a     tab_i     7, iforma
idb4a     tab_i     8, iforma
idb5a     tab_i     9, iforma
iband1a   tab_i     10, iforma         ;read bandwidths for same 5 formants
iband2a   tab_i     11, iforma
iband3a   tab_i     12, iforma
iband4a   tab_i     13, iforma
iband5a   tab_i     14, iforma
iamp1a    =    ampdb(idb1a)          ;convert db to linear multipliers
iamp2a    =    ampdb(idb2a)
iamp3a    =    ampdb(idb3a)
iamp4a    =    ampdb(idb4a)
iamp5a    =    ampdb(idb5a)

iform1b   tab_i     0, iformb          ;values of 5 formants for 2nd spectrum
iform2b   tab_i     1, iformb
iform3b   tab_i     2, iformb
iform4b   tab_i     3, iformb
iform5b   tab_i     4, iformb
idb1b     tab_i     5, iformb          ;decibel levels for 2nd set of formants
idb2b     tab_i     6, iformb
idb3b     tab_i     7, iformb
idb4b     tab_i     8, iformb
idb5b     tab_i     9, iformb
iband1b   tab_i     10, iformb         ;bandwidths for 2nd set of formants
iband2b   tab_i     11, iformb
iband3b   tab_i     12, iformb
iband4b   tab_i     13, iformb
iband5b   tab_i     14, iformb
iamp1b    =    ampdb(idb1b)          ;convert db to linear multipliers
iamp2b    =    ampdb(idb2b)
iamp3b    =    ampdb(idb3b)
iamp4b    =    ampdb(idb4b)
iamp5b    =    ampdb(idb5b)

```

```

kform1  line   iform1a, itotdur, iform1b   ;transition between formants
kform2  line   iform2a, itotdur, iform2b
kform3  line   iform3a, itotdur, iform3b
kform4  line   iform4a, itotdur, iform4b
kform5  line   iform5a, itotdur, iform5b
kband1  line   iband1a, itotdur, iband1b   ;transition of bandwidths
kband2  line   iband2a, itotdur, iband2b
kband3  line   iband3a, itotdur, iband3b
kband4  line   iband4a, itotdur, iband4b
kband5  line   iband5a, itotdur, iband5b
kamp1   line   iamp1a, itotdur, iamp1b   ;transition of amplitudes of formants
kamp2   line   iamp2a, itotdur, iamp2b
kamp3   line   iamp3a, itotdur, iamp3b
kamp4   line   iamp4a, itotdur, iamp4b
kamp5   line   iamp5a, itotdur, iamp5b

;5 formants for each spectrum
a1      fof2    kamp1, kfund, kform1, koct, kband1, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, kphs,
a2      fof2    kamp2, kfund, kform2, koct, kband2, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, kphs,
a3      fof2    kamp3, kfund, kform3, koct, kband3, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, kphs,
a4      fof2    kamp4, kfund, kform4, koct, kband4, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, kphs,
a5      fof2    kamp5, kfund, kform5, koct, kband5, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, kphs,

aout     =      (a1+a2+a3+a4+a5) * kamp/5   ;sum and scale

aenv  linen 1, 0.05, p3, 0.05   ;to avoid clicking

      outs      aout*aenv, aout*aenv
      endin

</CsInstruments>
<CsScore>
f1 0 8192 10 1
f2 0 4096 7 0 4096 1

;*****
; tables of formant values adapted from MiscFormants.html
; 100's: soprano    200's: alto    300's: countertenor    400's: tenor    500's: bass
; -01: "a" sound   -02: "e" sound   -03: "i" sound   -04: "o" sound   -05: "u" sound
; p-5 through p-9: frequencies of 5 formants
; p-10 through p-14: decibel levels of 5 formants
; p-15 through p-19: bandwidths of 5 formants

;          formant frequencies          decibel levels          bandwidths
;soprano
f101 0 16 -2    800    1150    2900    3900    4950    0.001    -6    -32    -20    -50
f102 0 16 -2    350    2000    2800    3600    4950    0.001    -20    -15    -40    -56    6
f103 0 16 -2    270    2140    2950    3900    4950    0.001    -12    -26    -26    -44    60
f104 0 16 -2    450    800    2830    3800    4950    0.001    -11    -22    -22    -50    40
f105 0 16 -2    325    700    2700    3800    4950    0.001    -16    -35    -40    -60    50
;alto
f201 0 16 -2    800    1150    2800    3500    4950    0.001    -4    -20    -36    -60    8
f202 0 16 -2    400    1600    2700    3300    4950    0.001    -24    -30    -35    -60
f203 0 16 -2    350    1700    2700    3700    4950    0.001    -20    -30    -36    -60
f204 0 16 -2    450    800    2830    3500    4950    0.001    -9    -16    -28    -55
f205 0 16 -2    325    700    2530    3500    4950    0.001    -12    -30    -40    -64
;countertenor
f301 0 16 -2    660    1120    2750    3000    3350    0.001    -6    -23    -24    -38
f302 0 16 -2    440    1800    2700    3000    3300    0.001    -14    -18    -20    -20    70
f303 0 16 -2    270    1850    2900    3350    3590    0.001    -24    -24    -36    -36    40
f304 0 16 -2    430    820    2700    3000    3300    0.001    -10    -26    -22    -34    40
f305 0 16 -2    370    630    2750    3000    3400    0.001    -20    -23    -30    -34    40
;tenor
f401 0 16 -2    650    1080    2650    2900    3250    0.001    -6    -7    -8    -22    8
f402 0 16 -2    400    1700    2600    3200    3580    0.001    -14    -12    -14    -20    70
f403 0 16 -2    290    1870    2800    3250    3540    0.001    -15    -18    -20    -30    40
f404 0 16 -2    400    800    2600    2800    3000    0.001    -10    -12    -12    -26    70

```

```

f405 0 16 -2 350 600 2700 2900 3300 0.001 -20 -17 -14 -26 40
;bass
f501 0 16 -2 600 1040 2250 2450 2750 0.001 -7 -9 -9 -20 6
f502 0 16 -2 400 1620 2400 2800 3100 0.001 -12 -9 -12 -18
f503 0 16 -2 250 1750 2600 3050 3340 0.001 -30 -16 -22 -28
f504 0 16 -2 400 750 2400 2600 2900 0.001 -11 -21 -20 -40
f505 0 16 -2 350 600 2400 2675 2950 0.001 -20 -32 -28 -36
,*****

; start dur amp start freq end freq start formant end formant
i1 0 1 .8 440 412.5 201 203
i1 + . .8 412.5 550 201 204
i1 + . .8 495 330 202 205

i1 + . .8 110 103.125 501 503
i1 + . .8 103.125 137.5 501 504
i1 + . .8 123.75 82.5 502 505

i1 7 . .4 440 412.5 201 203
i1 8 . .4 412.5 550 201 204
i1 9 . .4 495 330 202 205
i1 7 . .4 110 103.125 501 503
i1 8 . .4 103.125 137.5 501 504
i1 9 . .4 123.75 82.5 502 505
i1 + . .4 440 412.5 101 103
i1 + . .4 412.5 550 101 104
i1 + . .4 495 330 102 105
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*fof*

## Crédits

Auteur : Rasmus Ekman

*fof2* est une modification de *fof* par Rasmus Ekman

Nouveau dans Csound 3.45



# fofilter

fofilter — Filtre à formant.

## Description

Lorsqu'il reçoit un train d'impulsions, *fofilter* génère un flux de grains sinusoïdaux se recouvrant. Chaque grain est la réponse impulsionnelle d'une combinaison de deux filtres passe-bande. Les grains sont définis par leur durée d'attaque (qui détermine la largeur de jupe de la région formantique à -60dB) et leur durée de chute (largeur de bande à -6dB). Le recouvrement se produit quand  $1/\text{freq} < \text{decay}$ , mais, à la différence de FOF, il n'y a pas de limite supérieure au nombre de recouvrements. L'idée originale de cet opcode est venue de la classe formlet dans SuperCollider de J McCartney's, mais peut-être est-elle implémentée différemment (?).

## Syntaxe

```
asig fofilter ain, xcf, xris, xdec[, istor]
```

## Initialisation

*istor* -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal d'entrée.

*xcf* -- fréquence centrale du filtre.

*xris* -- durée d'attaque de la réponse impulsionnelle (secs).

*xdec* -- durée de chute de la réponse impulsionnelle (secs).

## Exemples

Voici un exemple de l'opcode fofilter. Il utilise le fichier *fofilter.csd* [examples/fofilter.csd].

### Exemple 344. Exemple de l'opcode fofilter.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fofilter.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kfe expseg 10, p3*0.9, 180, p3*0.1, 175
kenv linen .1, 0.05, p3, 0.05
asig buzz kenv, kfe, sr/(2*kfe), 1
afil fofilter asig, 900, 0.007, 0.04
      outs afil, afil

endin
</CsInstruments>
<CsScore>
; sine wave
f 1 0 16384 10 1

i 1 0 10
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Victor Lazzarini  
Janvier 2005

Nouveau greffon dans la version 5

Paramètres de taux audio introduits dans la version 6.02

Octobre 2013.

# fog

fog — La sortie audio est une succession de grains obtenus à partir des données d'une table de fonction.

## Description

La sortie audio est une succession de grains obtenus à partir des données de la table de fonction *ifna*. L'enveloppe locale de ces grains et leur distribution temporelle sont basées sur le modèle de la synthèse *fof* et permettent un contrôle détaillé de la synthèse granulaire.

## Syntaxe

```
ares fog xamp, xdens, xtrans, aspd, koct, kband, kris, kdur, kdec, \  
      iolaps, ifna, ifnb, itotdur [, iphs] [, itmode] [, iskip]
```

## Initialisation

*iolaps* -- quantité de mémoire préallouée nécessaire pour contenir les données de chevauchement des grains. Les chevauchements dépendent de la densité, et l'espace requis dépend de la valeur maximale de  $xdens * kdur$ . La surestimation de cet espace n'induit pas de coût de calcul supplémentaire. Chaque *iolap* utilise moins de 50 octets de mémoire.

*ifna*, *ifnb* -- numéros de table de deux fonctions. La première contient les données utilisées pour la granulation, provenant habituellement d'un fichier son (*GEN01*). La seconde est une forme ascendante, utilisée à l'endroit et à l'envers pour dessiner l'attaque et la chute des grains ; cette forme est normalement une sigmoïde (*GEN19*) mais elle peut être aussi linéaire (*GEN05*).

*itotdur* -- durée totale durant laquelle ce *fog* sera actif. Fixée normalement à p3. Aucun nouveau grain n'est créé si son *kdur* n'est pas contenu complètement dans le *itotdur* restant.

*iphs* (facultatif) -- phase initiale du fondamental, exprimée comme une fraction d'une période (0 à 1). La valeur par défaut est 0.

*itmode* (facultatif) -- type de transposition. S'il est nul, chaque grain garde la valeur *xtrans* avec laquelle il a été amorcé. Sinon, chaque grain est influencé par *xtrans* de manière continue. La valeur par défaut est 0.

*iskip* (facultatif, par défaut 0) -- s'il est non nul, l'initialisation est ignorée (ce qui permet l'utilisation du legato).

## Exécution

*xamp* -- facteur d'amplitude. L'amplitude dépend également du nombre de grains se chevauchant, de l'interaction de la forme montante (*ifnb*) et de la chute exponentielle (*kband*), et des valeurs de la forme d'onde du grain (*ifna*). L'amplitude réelle peut ainsi dépasser *xamp*.

*xdens* -- densité. Nombre de grains par seconde.

*xtrans* -- facteur de transposition. Le taux de lecture des données de la table de fonction *ifna* dans chaque grain. Il a pour effet de transposer le matériel original. Une valeur de 1 produit la hauteur originale. Les valeurs supérieures à 1 transposent vers le haut tandis que les valeurs inférieures à 1 le font vers le bas. Les valeurs négatives provoquent une lecture à l'envers de la table.

*aspd* -- indice de lecture initial. *aspd* est l'indice de lecture normalisé (0 à 1) dans la table *ifna* qui détermine le mouvement d'un pointeur à partir duquel commence la lecture dans chaque grain. (*xtrans* détermine le taux de lecture des données à partir de ce pointeur.)

*koct* -- indice d'octavation. Ce paramètre fonctionne de manière identique à celui qui est décrit dans *fof*.

*kband*, *kris*, *kdur*, *kdec* -- forme de l'enveloppe du grain. Ces paramètres déterminent les temps de décroissance exponentielle (*kband*), et d'attaque (*kris*), la durée totale (*kdur*), et celle de la chute (*kdec*) de l'enveloppe du grain. Leur mode opératoire est identique à celui des paramètres d'enveloppe locale dans *fof*.

## Exemples

Voici un exemple de l'opcode *fog*. Il utilise le fichier *fog.csd* [examples/fog.csd].

### Exemple 345. Exemple de l'opcode *fog*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fog.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

;p4 = transposition factor
;p5 = speed factor
;p6 = function table for grain data
il = sr/ftlen(1) ;scaling to reflect sample rate and table length
a1 phasor il*p5 ;index for speed
asigl fog .5, 15, p4, a1, 1, 0, .01, .5, .01, 30, 1, 2, p3 ;left channel
asigr fog .4, 25, p4+.2, a1, 1, 0, .01, .5, .01, 30, 1, 2, p3, .5 ;right channel
outs asigl, asigr

endin

</CsInstruments>
<CsScore>
f 1 0 131072 1 "fox.wav" 0 0 0
f 2 0 1024 19 .5 .5 270 .5

i 1 0 10 .7 .1
i 1 + 4 1.2 2
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael Clark

Huddersfield  
Mai 1997

Nouveau dans la version 3.46

Le générateur *fog* de Csound a été écrit par Michael Clarke, comme suite à ses travaux antérieurs basés sur l'algorithme FOF de l'IRCAM.

Notes ajoutées par Rasmus Ekman en septembre 2002.

# fold

fold — Ajoute un repliement artificiel à un signal audio.

## Description

Ajoute un repliement artificiel à un signal audio.

## Syntaxe

```
ares fold asig, kincr
```

## Exécution

*asig* -- signal d'entrée

*kincr* -- importance du repliement exprimée en multiple du taux d'échantillonnage. Doit être  $\geq 1$

*fold* est un opcode qui crée un repliement artificiel. Par exemple, quand *kincr* est égal à 1 avec *sr*=44100, aucun repliement n'est ajouté. Quand *kincr* vaut 2, le repliement est équivalent à un sous-échantillonnage à 22050, quand il vaut 4, à 11025, etc. Il est possible de donner à *kincr* des valeurs fractionnaires ce qui permet une variation continue du taux de repliement. On peut l'utiliser pour une large gamme d'effets spéciaux.

## Exemples

Voici un exemple de l'opcode fold. Il utilise le fichier *fold.csd* [examples/fold.csd].

### Exemple 346. Exemple de l'opcode fold.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fold.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

asig poscil3 .8, 400, 1 ;very clean sine
kincr line p4, p3, p5
asig fold asig, kincr
outs asig, asig
```

```
endin
</CsInstruments>
<CsScore>
; sine wave.
f 1 0 16384 10 1

i 1 0 4 2 2
i 1 5 4 5 5
i 1 10 4 10 10
i 1 15 4 1 100 ; Vary the fold-over amount from 1 to 100

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1999

Nouveau dans la version 3.56 de Csound.

# follow

follow — Générateur unitaire suiveur d'enveloppe.

## Description

Générateur unitaire suiveur d'enveloppe.

## Syntaxe

```
ares follow asig, idt
```

## Initialisation

*idt* -- C'est la période, en secondes, durant laquelle l'amplitude moyenne de *asig* est calculée. Si la fréquence de *asig* est faible, *idt* doit être grande (plus de la moitié de la période de *asig*).

## Exécution

*asig* -- Le signal dont on veut extraire l'enveloppe.

## Exemples

Voici un exemple de l'opcode follow. Il utilise les fichiers *follow.csd* [examples/follow.csd] et *beats.wav* [examples/beats.wav].

### Exemple 347. Exemple de l'opcode follow.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o follow.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

asig soundin "beats.wav"
outs asig, asig

endin

instr 2 ;envelope follower
```



```
as  soundin "beats.wav"
as  = as*.7  ;reduce volume a bit
at  tone    as, 500 ;smooth estimated envelope
af  follow  at, p4
asin poscil3 .5, 440, 1
; "beats.wav" provides amplitude for poscil
asig balance asin, af
    outs    asig, asig

endin
</CsInstruments>
<CsScore>
; sine wave.
f 1 0 32768 10 1

i 1 0 2
i 2 2 2 0.001 ;follow quickly
i 2 5 3 0.2   ;follow slowly
e
</CsScore>
</CsoundSynthesizer>
```

Pour éviter le bruit de transition produit par les discontinuités lors de la détection d'une enveloppe complexe, on peut utiliser un filtre passe-bas qui lissera l'enveloppe estimée.

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

# follow2

follow2 — Un autre extracteur d'enveloppe contrôlable.

## Description

Un extracteur d'enveloppe contrôlable utilisant un algorithme attribué à Jean-Marc Jot.

## Syntaxe

```
ares follow2 asig, katt, krel
```

## Exécution

*asig* -- le signal d'entrée dont l'enveloppe est suivie

*katt* -- la vitesse d'attaque (temps d'attaque à 60dB en secondes)

*krel* -- la vitesse de chute (temps de chute à 60dB en secondes)

La sortie suit l'enveloppe d'amplitude du signal d'entrée. La vitesse à laquelle la sortie augmente pour suivre le signal est contrôlée par *katt*, et la vitesse à laquelle elle diminue en réponse à une amplitude plus faible est contrôlée par *krel*. Cela donne une enveloppe plus lisse qu'avec *follow*.

## Exemples

Voici un exemple de l'opcode follow2. Il utilise les fichiers *follow2.csd* [examples/follow2.csd] et *beats.wav* [examples/beats.wav].

### Exemple 348. Exemple de l'opcode follow2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o follow2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

asig soundin "beats.wav"
outs asig, asig
endin
```

```

instr 2 ;using follow2

as soundin "beats.wav"
af follow2 as, p4, p5
ar rand 44100 ;noise
; "beats.wav" provides amplitude for noise
asig balance ar, af
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 2
i 2 2 2 0.001 0.01 ;quick attack & deacy
i 2 5 2 0.1 0.5 ;slow attack & deacy

e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch  
 L'algorithme de *follow2* est attribué à Jean-Marc Jot.  
 Université de Bath, Codemist Ltd.  
 Bath, UK  
 Février 2000

Nouveau dans la version 4.03 de Csound.

Notes ajoutées par Rasmus Ekman en septembre 2002.

# foscil

foscil — Un oscillateur élémentaire modulé en fréquence.

## Description

Un oscillateur élémentaire modulé en fréquence.

## Syntaxe

```
ares foscil xamp, kcps, xcar, xmod, kndx, ifn [, iphs]
```

## Initialisation

*ifn* -- numéro de la table de fonction. Nécessite un point de garde de lecture cyclique.

*iphs* (facultatif, par défaut 0) -- phase initiale de la forme d'onde dans la table *ifn*, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation de la phase sera ignorée. La valeur par défaut est 0.

## Exécution

*xamp* -- l'amplitude du signal de sortie.

*kcps* -- un dénominateur commun, en cycles par seconde, pour les fréquences porteuse et modulante.

*xcar* -- un facteur qui, lorsqu'il est multiplié par le paramètre *kcps*, donne la fréquence de la porteuse.

*xmod* -- un facteur qui, lorsqu'il est multiplié par le paramètre *kcps*, donne la fréquence de la modulante.

*kndx* -- l'indice de modulation.

*foscil* est une unité composée qui assemble deux opcodes *oscil* dans la configuration familière de synthèse FM de Chowning, où la sortie au taux audio de l'un des générateurs est utilisée pour moduler l'entrée en fréquence de l'autre (la « porteuse »). Fréquence de la porteuse =  $kcps * xcar$  et fréquence modulante =  $kcps * xmod$ . Pour des valeurs entières de *xcar* et de *xmod*, la fondamentale perçue sera la valeur positive minimale de  $kcps * (xcar - n * xmod)$ ,  $n = 0, 1, 2, \dots$  L'entrée *kndx* est l'indice de modulation (habituellement variant dans le temps approximativement dans l'intervalle de 0 à 4) qui détermine la distribution de l'énergie acoustique parmi les positions des partiels données par  $n = 0, 1, 2, \dots$ , etc. *ifn* doit pointer sur une onde sinus stockée. Avant la version 3.50, *xcar* et *xmod* ne pouvaient être que de taux-k.

La formule utilisée pour cette implémentation de la synthèse FM est  $xamp * \cos(2\pi * t * kcps * xcar + kndx * \sin(2\pi * t * kcps * xmod) - \pi)$ , en supposant que la table est une onde sinus.

## Exemples

Voici un exemple de l'opcode foscil. Il utilise le fichier *foscil.csd* [examples/foscil.csd].

### Exemple 349. Exemple de l'opcode foscil.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o foscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kcps = 440
kcar = 1
kmod = p4
kndx line 0, p3, 20 ;intensivy sidebands

asig foscil .5, kcps, kcar, kmod, kn dx, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
; sine
f 1 0 16384 10 1

i 1 0 9 .01 ;vibrato
i 1 10 . 1
i 1 20 . 1.414 ;gong-ish
i 1 30 5 2.05 ;with "beat"
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

Plus d'information au sujet de la modulation de fréquence sur Wikipedia : [http://en.wikipedia.org/wiki/Frequency\\_modulation\\_synthesis](http://en.wikipedia.org/wiki/Frequency_modulation_synthesis)

## Crédits

Exemple écrit par Kevin Conder.

# foscili

foscili — Oscillateur élémentaire modulé en fréquence avec interpolation linéaire.

## Description

Oscillateur élémentaire modulé en fréquence avec interpolation linéaire.

## Syntaxe

```
ares foscili xamp, kcps, xcar, xmod, kndx, ifn [ , iphs]
```

## Initialisation

*ifn* -- numéro de la table de fonction. Nécessite un point de garde de lecture cyclique.

*iphs* (facultatif, par défaut 0) -- phase initiale de la forme d'onde dans la table *ifn*, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation de la phase sera ignorée. La valeur par défaut est 0.

## Exécution

*xamp* -- l'amplitude du signal de sortie.

*kcps* -- un dénominateur commun, en cycles par seconde, pour les fréquences porteuse et modulante.

*xcar* -- un facteur qui, lorsqu'il est multiplié par le paramètre *kcps*, donne la fréquence de la porteuse.

*xmod* -- un facteur qui, lorsqu'il est multiplié par le paramètre *kcps*, donne la fréquence de la modulante.

*kndx* -- l'indice de modulation.

*foscili* diffère de *foscil* en ce que la procédure standard d'utilisation d'une phase tronquée comme index de lecture des échantillons est remplacée ici par une interpolation entre deux lectures successives. Les générateurs avec interpolation produiront un signal de sortie nettement plus propre, mais ils peuvent prendre jusqu'à deux fois plus de temps de calcul. On peut obtenir également ce type de précision sans le surcoût du calcul de l'interpolation en utilisant de grandes tables de fonction stockées de 2K, 4K ou 8K points, si l'on dispose de cet espace mémoire.

## Exemples

Voici un exemple de l'opcode *foscili*. Il utilise le fichier *foscili.csd* [examples/foscili.csd].

### Exemple 350. Exemple de l'opcode foscili.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```

-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o foscili.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kcps = 440
kcar = 1
kmod = p4
kndx line 0, p3, 20 ;intensivy sidebands

asig foscili .5, kcps, kcar, kmod, kndx, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
; sine
f 1 0 16384 10 1

i 1 0 9 .01 ;vibrato
i 1 10 . 1
i 1 20 . 1.414 ;gong-ish
i 1 30 5 2.05 ;with "beat"
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

Plus d'information au sujet de la modulation de fréquence sur Wikipedia : [http://en.wikipedia.org/wiki/Frequency\\_modulation\\_synthesis](http://en.wikipedia.org/wiki/Frequency_modulation_synthesis)

## Crédits

Exemple écrit par Kevin Conder.

# fout

fout — Envoie des signaux de taux-a vers un nombre arbitraire de canaux dans un fichier externe.

## Description

*fout* envoie *N* signaux de taux-a vers un fichier spécifié à *N* canaux.

## Syntaxe

```
fout ifilename, iformat, aout1 [, aout2, aout3,...,aoutN]
```

```
fout ifilename, iformat, array[]
```

## Initialisation

*ifilename* -- le nom du fichier de sortie (entre guillemets)

*iformat* -- un indicateur pour choisir le format du fichier de sortie (note : il se peut que les versions de Csound antérieures à la 5.0 ne supportent que les formats 0, 1 et 2) :

- -1 - formatage identique à celui des options globales de sortie de Csound (-A, -W, --format=..., etc).
- 0 - échantillons en flottants sur 32 bit sans en-tête (fichier PCM binaire multicanaux)
- 1 - entiers sur 16 bit sans en-tête (fichier PCM binaire multicanaux)
- 2 - entiers sur 16 bit avec en-tête. Le type de l'en-tête dépend du format de restitution (-o). Par exemple, si l'utilisateur choisit le format AIFF (en utilisant l'*option* -A), le format de l'en-tête sera de type AIFF.
- 3 - échantillons u-law avec un en-tête (voir iformat=2).
- 4 - entiers sur 16 bit avec un en-tête (voir iformat=2).
- 5 - entiers sur 32 bit avec un en-tête (voir iformat=2).
- 6 - flottants sur 32 bit avec un en-tête (voir iformat=2).
- 7 - entiers non signés sur 8 bit avec un en-tête (voir iformat=2).
- 8 - entiers sur 24 bit avec un en-tête (voir iformat=2).
- 9 - flottants sur 64 bit avec un en-tête (voir iformat=2).
- 50 - format ogg-vorbis.

De plus, les versions de Csound à partir de la 5.0 permettent de choisir explicitement un type d'en-tête particulier en spécifiant le format comme 10 \* typeFichier + formatEchantillon, où typeFichier peut valoir 1 pour WAV, 2 pour AIFF, 3 pour fichiers brut (sans en-tête) et 4 pour IRCAM ; formatEchantillon est l'une des valeurs ci-dessus comprise entre 0 et 9, sauf que le format d'échantillon 0 est déduit de la ligne de commande (-o), le format 1 représente des entiers signés sur 8 bit et le format 2 est a-law. Ainsi, par exemple, *iformat* = 25 signifie des entiers sur 32 bit avec un en-tête AIFF.



## Exécution

*aout1*,... *aoutN* -- signaux à écrire dans le fichier. Dans le cas de fichiers bruts, l'étendue de l'amplitude des signaux audio est déterminée par le format d'échantillon choisi ; pour les fichiers son avec un en-tête comme WAV et AIFF, les signaux audio doivent être dans l'intervalle compris entre -0dbfs et 0dbfs.

*fout* (file output) écrit des échantillons de signaux audio dans un fichier avec n'importe quel nombre de canaux. Le nombre de canaux dépend du nombre de variables *aoutN* (par exemple un signal mono avec un seul argument de *taux-a*, un signal stéréo avec deux arguments de *taux-a*, etc.) Le nombre maximum de canaux est fixé à 64. Plusieurs opcodes *fout* peuvent se trouver dans le même instrument, se référant à différents fichiers.

Noter que, contrairement à *out*, *outs* et *outq*, *fout* ne remet pas à zéro la variable audio, c'est pourquoi l'on doit la remettre à zéro après l'appel. Si l'on travaille en polyphonie, on peut utiliser les opcodes *vincr* et *clear* pour cela.

Noter que *fout* et *foutk* utilisent une chaîne de caractères contenant un nom de chemin de fichier. Alors qu'avec *fouti* et *foutir*, le fichier cible ne peut être spécifié que par un identificateur numérique.



### Note

Si l'on utilise *fout* pour générer un fichier audio depuis la sortie globale de Csound, il peut être désirable d'utiliser l'opcode *monitor* qui peut capter le tampon de sortie, ce qui évite le routage.

## Exemples

Voici un exemple de l'opcode *fout*. Il utilise le fichier *fout.csd* [examples/fout.csd].

### Exemple 351. Exemple de l'opcode *fout*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fout.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 2^10, 10, 1

instr 1

asig poscil3 .5, 880, giSine
;write a raw file: 32 bits with header
fout "fout_880.wav", 15, asig
outs asig, asig
```

```

endin

instr 2

klfo lfo 1, 2, 0
asig poscil3 .5*klfo, 220, giSine
;write an aiff file: 32 bits with header
    fout "fout_aif.aiff", 25, asig
;
    fout "fout_all3.wav", 14, asig
    outs asig, asig

endin

instr 99 ;read the stereo csound output buffer

allL, allR monitor
;write the output of csound to an audio file
;to a wav file: 16 bits with header
    fout "fout_all.wav", 14, allL, allR

endin
</CsInstruments>
<CsScore>

i 1 0 2
i 2 0 3
i 99 0 3
e
</CsScore>
</CsoundSynthesizer>

```

Voici un autre exemple de *fout*, qui l'utilise pour sauvegarder le contenu d'une table dans un fichier audio. Il utilise les fichiers *fout\_ftable.csd* [examples/fout\_ftable.csd] et *beats.wav* [examples/beats.wav].

### Exemple 352. Exemple de l'opcode fout pour sauvegarder le contenu d'une ftable.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc    -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fout_ftable.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
; By: Jonathan Murphy 2007

gilen  = 131072
gicps  = sr/gilen
gitab  ftgen 1, 0, gilen, 10, 1

        instr 1

/***** write file to table *****/

ain disk2 "beats.wav", 1, 0, 1
aphs phasor gicps
andx  = aphis * gilen
tablew ain, andx, gitab

/***** write table to file *****/

aosc table aphis, gitab, 1
out aosc
fout "beats_copy.wav", 6, aosc

```

```
        endin  
  
</CsInstruments>  
<CsScore>  
i1 0 2  
e  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*fiopen, fouti, foutir, foutk, monitor*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1999  
Author: John ffitch  
NUIM, 2013

Nouveau dans la version 3.56 de Csound

Octobre 2002. Note ajoutée par Richard Dobson.

Variante pour tableau ajoutée dans la version 6.01

# fouti

**fouti** — Envoie des signaux de taux-i d'un nombre arbitraire de canaux dans un fichier spécifié.

## Description

*fouti* envoie *N* signaux de taux-i vers un fichier spécifié à *N* canaux.

## Syntaxe

```
fouti ihandle, iformat, iflag, iout1 [, iout2, iout3, ..., ioutN]
```

## Initialisation

*ihandle* -- un nombre qui spécifie ce fichier.

*iformat* -- un indicateur pour choisir le format du fichier de sortie :

- 0 - flottants en format texte
- 1 - flottants sur 32 bit en format binaire

*iflag* -- choisit le mode d'écriture dans le fichier ASCII (n'est valide qu'en mode ASCII ; en mode binaire *iflag* n'a aucune signification, mais il doit quand même être présent). *iflag* peut prendre une des valeurs suivantes :

- 0 - ligne de texte sans préfixe d'instrument
- 1 - ligne de texte avec préfixe d'instrument (voir ci-dessous)
- 2 - remet à zéro le temps des préfixes d'instrument (à n'utiliser que dans certains cas particuliers. Voir ci-dessous)

*iout*, ..., *ioutN* -- valeurs à écrire dans le fichier

## Exécution

*fouti* et *foutir* écrivent des valeurs de taux-i dans un fichier. On utilise ces opcodes principalement pour générer un fichier de partition pendant une session en temps réel. Pour cela, il faut fixer *iformat* à 0 (sortie dans un fichier texte) et *iflag* à 1, ce qui active la sortie d'un préfixe constitué des chaînes de caractères *inum*, *actiontime* et *duration*, avant les valeurs des arguments *iout1*...*ioutN*. Les arguments dans le préfixe font référence au numéro de l'instrument, à la date et à la durée de la note courante.

Noter que *fout* et *foutk* peuvent utiliser soit une chaîne de caractères contenant un nom de chemin de fichier, soit un identificateur numérique généré par *fiopen*. Alors qu'avec *fouti* et *foutir*, le fichier cible ne peut être spécifié que par un identificateur numérique.

## Exemples

Voici un exemple de l'opcode *fouti*. Il utilise le fichier *fouti.csd* [examples/fouti.csd].

### Exemple 353. Exemple de l'opcode fouti.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o fouti.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gihand fiopen "test.txt", 0

instr 1

ires random 0, 10
fouti gihand, 0, 1, ires
ficlose gihand

endin
</CsInstruments>
<CsScore>

i 1 0 1

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*fiopen, fout, foutir, foutk*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1999

Nouveau dans la version 3.56 de Csound

# foutir

*foutir* — Envoie des signaux de taux-*i* d'un nombre arbitraire de canaux dans un fichier spécifié.

## Description

*foutir* envoie *N* signaux de taux-*i* vers un fichier spécifié à *N* canaux.

## Syntaxe

```
foutir ihandle, iformat, iflag, iout1 [, iout2, iout3, ..., ioutN]
```

## Initialisation

*ihandle* -- un nombre qui spécifie ce fichier.

*iformat* -- un indicateur pour choisir le format du fichier de sortie :

- 0 - flottants en format texte
- 1 - flottants sur 32 bit en format binaire

*iflag* -- choisit le mode d'écriture dans le fichier ASCII (n'est valide qu'en mode ASCII ; en mode binaire *iflag* n'a aucune signification, mais il doit quand même être présent). *iflag* peut prendre une des valeurs suivantes :

- 0 - ligne de texte sans préfixe d'instrument
- 1 - ligne de texte avec préfixe d'instrument (voir ci-dessous)
- 2 - remet à zéro le temps des préfixes d'instrument (à n'utiliser que dans certains cas particuliers. Voir ci-dessous)

*iout*..., *ioutN* -- valeurs à écrire dans le fichier

## Exécution

*fouti* et *foutir* écrivent des valeurs de taux-*i* dans un fichier. On utilise ces opcodes principalement pour générer un fichier de partition pendant une session en temps réel. Pour cela, il faut fixer *iformat* à 0 (sortie dans un fichier texte) et *iflag* à 1, ce qui active la sortie d'un préfixe constitué des chaînes de caractères *inum*, *actiontime* et *duration*, avant les valeurs des arguments *iout1*...*ioutN*. Les arguments dans le préfixe font référence au numéro de l'instrument, à la date et à la durée de la note courante.

La différence entre *fouti* et *foutir* est que dans le cas de *fouti*, quand *iflag* vaut 1, la durée du premier opcode est indéfinie (elle est ainsi remplacée par un point). Tandis que *foutir* n'est défini qu'à la fin de la note, si bien que la ligne de texte correspondante n'est écrite qu'à la fin de la note courante (afin de connaître sa durée). Le fichier correspondant est lié par la valeur de *ihandle* générée par l'opcode *fiopen*. On peut ainsi utiliser *fouti* et *foutir* pour générer une partition Csound tout en jouant une session en temps réel.

Noter que *fout* et *foutk* peuvent utiliser soit une chaîne de caractères contenant un nom de chemin de fichier, soit un identificateur numérique généré par *fiopen*. Alors qu'avec *fouti* et *foutir*, le fichier cible ne peut être spécifié que par un identificateur numérique.

## Exemples

Voici un exemple de l'opcode `foutir`. Il utilise le fichier `foutir.csd` [examples/foutir.csd] et il crée la liste "foutir.sco". Celle-ci peut être utilisée comme fichier de partition.

### Exemple 354. Exemple de l'opcode `foutir`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0   ;;realtime audio out and midi in
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o foutir.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gihand fiopen "foutir.sco", 0

instr 1 ; play virtual midi keyboard

inot notnum ;just for priting on screen
icps cpsmidi
iamp ampmidi 1

      foutir gihand, 0, 1, icps, iamp
      prints "WRITING:\n"
      prints "note = %f,velocity = %f\n", icps, iamp ;prints them
      ficlose gihand
asig pluck iamp, icps, 1000, 0, 1
      outs asig, asig

endin
</CsInstruments>
<CsScore>

f 0 10

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*fiopen, fout, fouti, foutk*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1999

Nouveau dans la version 3.56 de Csound

# foutk

**foutk** — Envoie des signaux de taux-k vers un nombre arbitraire de canaux dans un fichier externe, en format brut (sans en-tête).

## Description

*foutk* envoie *N* signaux de taux-k vers un fichier spécifié à *N* canaux.

## Syntaxe

```
foutk ifilename, iformat, kout1 [, kout2, kout3, ..., koutN]
```

## Initialisation

*ifilename* -- le nom du fichier de sortie (entre guillemets)

*iformat* -- un indicateur pour choisir le format du fichier de sortie (note : il se peut que les versions de Csound antérieures à la 5.0 ne supportent que les formats 0 et 1) :

- 0 - échantillons en flottants sur 32 bit sans en-tête (fichier PCM binaire multicanaux)
- 1 - entiers sur 16 bit sans en-tête (fichier PCM binaire multicanaux)
- 2 - entiers sur 16 bit sans en-tête (fichier PCM binaire multicanaux)
- 3 - échantillons u-law sans en-tête
- 4 - entiers sur 16 bit sans en-tête
- 5 - entiers sur 32 bit sans en-tête
- 6 - flottants sur 32 bit sans en-tête
- 7 - entiers non signés sur 8 bit sans en-tête
- 8 - entiers sur 24 bit sans en-tête
- 9 - flottants sur 64 bit sans en-tête

## Exécution

*kout1, ..., koutN* -- signaux au taux de contrôle à écrire dans le fichier. L'étendue de l'amplitude des signaux est déterminée par le format d'échantillon choisi.

*foutk* opère de la même manière que *fout*, mais avec des signaux de taux-k. *iformat* peut prendre une valeur entre 0 et 9, ou 0 et 1 avec une ancienne version de Csound.

Noter que *fout* et *foutk* peuvent utiliser soit une chaîne de caractères contenant un nom de chemin de fichier, soit un identificateur numérique généré par *fiopen*. Alors qu'avec *fouti* et *foutir*, le fichier cible ne peut être spécifié que par un identificateur numérique.



## Voir aussi

*fiopen, fout, fouti, foutir*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1999

Nouveau dans la version 3.56 de Csound

# fprintks

fprintks — Semblable à printks mais imprime dans un fichier.

## Description

Semblable à *printks* mais imprime dans un fichier.

## Syntaxe

```
fprintks "filename", "string", [, kval1] [, kval2] [...]
```

## Initialisation

*"filename"* -- nom du fichier de sortie.

*"string"* -- la chaîne de texte à imprimer. Peut contenir jusqu'à 8192 caractères et doit être entre guillemets.

## Exécution

*kval1*, *kval2*, ... (facultatif) -- Les valeurs de taux-k à imprimer. Elle sont spécifiées dans « *string* » avec les spécificateurs de format du C standard (%f, %d, etc.) dans l'ordre donné.

*fprintks* est semblable à l'opcode *printks* sauf qu'il imprime dans un fichier et qu'il n'a pas de paramètre de taux-i. Pour plus d'information sur le formatage de la sortie, prière de consulter la documentation de *printks*.

## Exemples

Voici un exemple de l'opcode fprintks. Il utilise le fichier *fprintks.csd* [exemples/fprintks.csd].

### Exemple 355. Exemple de l'opcode fprintks.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o fprintks.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Matt Ingalls, edited by Kevin Conder. */
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a score generator example.
instr 1
; K-rate stuff.
```

```

kstart init 0
kdur linrand 10
kpitch linrand 8

; Printing to to a file called "my.sco".
fprintks "my.sco", "i1\\t%2.2f\\t%2.2f\\t%2.2f\\n", kstart, kdur, 4+kpitch

knext linrand 1
kstart = kstart + knext
endin

</CsInstruments>
<CsScore>

/* Written by Matt Ingalls, edited by Kevin Conder. */
; Play Instrument #1.
i 1 0 0.001

</CsScore>
</CsoundSynthesizer>

```

Cet exemple générera un fichier nommé « my.sco ». Il contiendra des lignes comme celles-ci :

```

i1      0.00      3.94      10.26
i1      0.20      3.35      6.22
i1      0.67      3.65      11.33
i1      1.31      1.42      4.13

```

Voici un exemple de l'opcode fprintks, qui convertit un fichier MIDI standard en partition Csound. Il utilise le fichier *fprintks-2.csd* [examples/fprintks-2.csd].

### Exemple 356. Exemple de l'opcode fprintks pour convertir un fichier MIDI en partition Csound.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
; -odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-n -Fmidichn_advanced.mid
;Don't write audio ouput to disk and use the file midichn_advanced.mid as MIDI input
</CsOptions>
<CsInstruments>

sr      = 48000
ksmps   = 16
nchnls  = 2

;Example by Jonathan Murphy 2007

; assign all midi events to instr 1000
massign 0, 1000
pgmassign 0, 1000

instr 1000

ktim timeinsts

kst, kch, kd1, kd2 midiin
if (kst != 0) then
; p4 = MIDI event type  p5 = channel  p6= data1  p7= data2

```

```

        fprintks "MIDI2cs.sco", "i1\\t%f\\t%f\\t%d\\t%d\\t%d\\t%d\\n", ktim, 1/kr, kst, kch, kd1, kd2
    endif

    endin

</CsInstruments>
<CsScore>
i1000 0 10000
e
</CsScore>
</CsoundSynthesizer>

```

Cet exemple générera un fichier nommé « MIDI2cs.sco » contenant des évènements i correspondant au fichier MIDI.

Voici un exemple avancé de l'opcode fprintks, qui génère une partition pour Csound. Il utilise le fichier *scogen-2.csd* [examples/scogen.csd].

### Exemple 357. Exemple de l'opcode fprintks pour créer un générateur de fichier de partition Csound au moyen de Csound.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
; -odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-n
;Don't write audio ouput to disk
</CsOptions>
<CsInstruments>
;=====
;          scogen.csd          by: Matt Ingalls
;
;   a "port" of sorts
;   of the old "mills" score generator (scogen)
;
;   this instrument creates a schottstaedt.sco file
;   to be used with the schottstaedt.orc file
;
;   as long as you dont save schottstaedt.orc as a .csd
;   file, you should be able to keep it open in MacCsound
;   and render each newly generated .sco file.
;
;=====

gScoName = "/Users/matt/Desktop/schottstaedt.sco"      ; the name of the file to be generated

sr      =      100      ; this defines our temporal resolution,
                        ; an sr of 100 means we will generate p2 and p3 values
                        ; to the nearest 1/100th of a second

ksmps   =      1      ; set kr=sr so we can do everything at k-rate

; some print opcodes
opcode PrintInteger, 0, k
    kval    xin
    fprintks gScoName, "%d", kval
endop

opcode PrintFloat, 0, k

```

```

        kval    xin
        fprintfs    gScoName, "%f", kval
    endop

opcode PrintTab, 0, 0
    fprintfs    gScoName, "%n"
endop

opcode PrintReturn, 0, 0
    fprintfs    gScoName, "%r"
endop

; recursively calling opcode to handle all the optional parameters
opcode ProcessAdditionalPfields, 0, ikio
    iPtable, kndx, iNumPfields, iPfield xin

; additional pfields start at 5, we use a default 0 to identify the first call
iPfield = (iPfield == 0 ? 5 : iPfield)

if (iPfield > iNumPfields) goto endloop
; find our tables
iMinTable table    2*iPfield-1, iPtable
iMaxTable table    2*iPfield, iPtable

; get values from our tables
kMin tablei    kndx, iMinTable
kMax tablei    kndx, iMaxTable

; find a random value in the range and write it to the score
fprintfks gScoName, "%t%f", kMin + rnd(kMax-kMin)

; recursively call for any additional pfields.
ProcessAdditionalPfields iPtable, kndx, iNumPfields, iPfield + 1
endloop:

endop

/* =====
Generate a gesture of i-statements

p2 = start of the gesture
p3 = duration of the gesture
p4 = number of a function that contains a list of all
    function table numbers used to define the
    pfield random distribution
p5 = scale generated p4 values according to density (0=off, 1=on) [todo]
p6 = let durations overlap gesture duration (0=off, 1=on) [todo]
p7 = seed for random number generator seed [todo]
=====
*/
instr Gesture

; initialize
iResolution = 1/sr

kNextStart init p2
kCurrentTime init p2

iNumPfields table    0, p4
iInstrMinTable table    1, p4
iInstrMaxTable table    2, p4
iDensityMinTable table    3, p4
iDensityMaxTable table    4, p4
iDurMinTable table    5, p4
iDurMaxTable table    6, p4

```

```

iAmpMinTable table 7, p4
iAmpMaxTable table 8, p4

; check to make sure there is enough data
print iNumPfields
if iNumPfields < 4 then
    prints "%dError: At least 4 p-fields (8 functions) need to be specified.%n", iNumPfields
    turnoff
endif

; initial comment
fprints gScoName, "%!Generated Gesture from %f to %f seconds%n %!%t%twith a p-max of %d%n%n", p2

; k-rate stuff
if (kCurrentTime >= kNextStart) then ; write a new note!

    kndx = (kCurrentTime-p2)/p3

    ; get the required pfield ranges
    kInstMin tablei kndx, iInstrMinTable
    kInstMax tablei kndx, iInstrMaxTable
    kDensMin tablei kndx, iDensityMinTable
    kDensMax tablei kndx, iDensityMaxTable
    kDurMin tablei kndx, iDurMinTable
    kDurMax tablei kndx, iDurMaxTable
    kAmpMin tablei kndx, iAmpMinTable
    kAmpMax tablei kndx, iAmpMaxTable

    ; find random values for all our required parameters and print the i-statement
    fprintks gScoName, "i%d%t%f%t%f%t%f", kInstMin + rnd(kInstMax-kInstMin), kNextStart, kDurMin +

    ; now any additional pfields
    ProcessAdditionalPfields p4, kndx, iNumPfields

    PrintReturn

    ; calculate next starttime
    kDensity = kDensMin + rnd(kDensMax-kDensMin)
    if (kDensity < iResolution) then
        kDensity = iResolution
    endif
    kNextStart = kNextStart + kDensity
endif

kCurrentTime = kCurrentTime + iResolution
endin

</CsInstruments>
<CsScore>
/*
=====
scogen.sco

    this csound module generates a score file
    you specify a gesture of notes by giving
    the "gesture" instrument a number to a
    (negative) gen2 table.

    this table stores numbers to pairs of functions.
    each function-pair represents a range (min-max)
    of randomness for every pfield for the notes to
    be generated.
=====
*/

```

```

; common tables for pfield ranges
f100 0 2 -7 0 2 0 ; static 0
f101 0 2 -7 1 2 1 ; static 1
f102 0 2 -7 0 2 1 ; ramp 0->1
f103 0 2 -7 1 2 0 ; ramp 1->0
f105 0 2 -7 10 2 10 ; static 10
f106 0 2 -7 .1 2 .1 ; static .1

; specific pfield ranges
f10 0 2 -7 .8 2 .01 ; density
f11 0 2 -7 8 2 4 ; pitchmin
f12 0 2 -7 8 2 12 ; pitchmax

;=== table containing the function numbers used for all the p-field distributions
;
; p1 - table number
; p2 - time table is instantiated
; p3 - size of table (must be >= p5!)
; p4 - gen# (should be = -2)
; p5 - number of pfields of each note to be generated
; p6 - table number of the function representing the minimum possible note number (p1) of a generation
; p7 - table number of the function representing the maximum possible note number (p1) of a generation
; p8 - table number of the function representing the minimum possible noteon-to-noteon time (p2)
; p9 - table number of the function representing the maximum possible noteon-to-noteon time (p2)
; p10 - table number of the function representing the minimum possible duration (p3) of a generation
; p11 - table number of the function representing the maximum possible duration (p3) of a generation
; p12 - table number of the function representing the maximum possible amplitude (p4) of a generation
; p13 - table number of the function representing the maximum possible amplitude (p5) of a generation
; p14,p16.. - table number of the function representing the minimum possible value for additional pfields
; p15,p17.. - table number of the function representing the maximum possible value for additional pfields

; siz 2 #pds p1min p1max p2min p2max p3min p3max p4min p4max p5min p5max p6
f1 0 32 -2 6 101 101 10 10 101 105 100 106 11 12 100 101

;gesture definitions
; start dur pTble scale overlap seed
i"Gesture" 0 60 1 ;todo-->0 0 123
</CsScore>

</CsoundSynthesizer>

```

Cet exemple générera un fichier nommé « schottstaedt.sco » que l'on peut utiliser comme partition avec *schottstaedt.orc* [examples/schottstaedt.orc]

## Voir aussi

*printks*

## Crédits

Auteur : Matt Ingalls  
Janvier 2003

# fprints

fprints — Semblable à prints mais imprime dans un fichier.

## Description

Semblable à *prints* mais imprime dans un fichier.

## Syntaxe

```
fprints "filename", "string" [, ival1] [, ival2] [...]
```

## Initialisation

*"filename"* -- nom du fichier de sortie.

*"string"* -- la chaîne de texte à imprimer. Peut contenir jusqu'à 8192 caractères et doit être entre guillemets.

*ival1, ival2, ...* (facultatif) -- Les valeurs de taux-i à imprimer. Elle sont spécifiées dans « *string* » avec les spécificateurs de format du C standard (%f, %d, etc.) dans l'ordre donné.

## Exécution

*fprints* est semblable à l'opcode *prints* sauf qu'il imprime dans un fichier. Pour plus d'information sur le formatage de la sortie, prière de consulter la documentation de *printks*.

## Exemples

Voici un exemple de l'opcode *fprints*. Il utilise le fichier *fprints.csd* [examples/fprints.csd]. Noter que l'exemple doit être exécuté depuis un répertoire dans lequel l'écriture est autorisée.

### Exemple 358. Exemple de l'opcode fprints.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o fprints.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Matt Ingalls, edited by Kevin Conder. */
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```



```
; Instrument #1 - a score generator example.
instr 1
  ; Print to the file "my.sco".
  fprints "my.sco", "%!Generated score by ma++\\n \\n"
endin

</CsInstruments>
<CsScore>

/* Written by Matt Ingalls, edited by Kevin Conder. */
; Play Instrument #1.
i 1 0 0.001

</CsScore>
</CsoundSynthesizer>
```

Cet exemple générera un fichier nommé « my.sco ». Il contiendra cette ligne :

```
!Generated score by ma++
```

## Voir aussi

*prints*

## Crédits

Auteur : Matt Ingalls  
Janvier 2003

# frac

frac — Retourne la partie fractionnaire d'un nombre décimal.

## Description

Retourne la partie fractionnaire de  $x$ .

## Syntaxe

**frac**( $x$ ) (arguments de taux-i ou de taux-k ; fonctionne aussi au taux-a dans Csound5)

**frac**( $k/i[]$ ) ( $k$ - ou  $i$ -tableau)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode frac. Il utilise le fichier *frac.csd* [examples/frac.csd].

### Exemple 359. Exemple de l'opcode frac.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o frac.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = 16 / 5
  i2 = frac(i1)

  print i2
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
```

e

```
</CsScore>  
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme :

```
instr 1:  i2 = 0.200
```

## Voir aussi

*abs, exp, int, log, log10, i, sqrt*

## Crédits

Exemple écrit par Kevin Conder.

# fractalnoise

fractalnoise — Un générateur de bruit fractal.

## Description

Opcodes du greffon fractalnoise.

Un générateur de bruit fractal implémenté comme un bruit blanc filtré par 15 filtres du premier ordre en cascade.

## Syntaxe

```
ares fractalnoise kamp, kbeta
```

## Exécution

*kamp* -- amplitude.

*kbeta* -- paramètre spectral fonction de la dimension fractale

- 0 - blanc
- 1 - rose
- 2 - brun

## Exemples

Voici un exemple de l'opcode fractalnoise. Il utilise le fichier *fractalnoise.csd* [exemples/fractalnoise.csd].

### Exemple 360. Exemple de l'opcode fractalnoise.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o oscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kbeta linseg 0, p3/4, 2, p3/4, 0, p3*0.1, 2, p3*0.15, 0
```

```
seed 20120124
aout fractalnoise 0.05, kbeta
outs aout, aout

endin
</CsInstruments>
<CsScore>
i1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

## Références

1. R. Saletti. A comparison between two methods to generate  $1/(f^\gamma)$  noise. In Proc. IEEE, volume 74, pp. 1595-1596, Novembre 1986.
2. G. Corsini and R. Saletti. A  $1/(f^\gamma)$  power spectrum noise sequence generator. IEEE Trans. on Instrumentation and Measurement, 37(4):615-619, Décembre 1988.
3. The Sounding Object, edited by Davide Rocchesso and Federico Fontana, Edizioni di Mondo Estremo. Chapter 8 by Federico Avanzini, pp. 154-157.

## Crédits

Auteur : Tito Latini  
Janvier 2012

Nouveau dans la version 5.16 de Csound.

# framebuffer

framebuffer — Lit des signaux audio dans des tableaux unidimensionnels de taux-k et vice versa avec une taille de tampon donnée.

## Description

Opcode du greffon framebuffer.

*framebuffer* convertit des signaux audio dans un tableau unidimensionnel de taux-k de taille donnée. La taille du tableau de taux-k doit être supérieure à *ksmps*. Il convertit aussi les tableaux unidimensionnels de taux-k en signaux audio, les tableaux de taux-k ne doivent pas être initialisés et avoir une taille supérieure à *ksmps*. Le tampon est circulaire et peut être utilisé pour du traitement audio à base de trames comme l'analyse/resynthèse spectrale ou comme une simple ligne à retard.

## Syntaxe

```
kout[] framebuffer ain, isize
```

```
aout framebuffer kin, isize
```

## Initialisation

*isize* -- La quantité d'échantillons contenus dans le tampon.

## Exécution

*kout[]* -- Le tableau de taux-k en sortie. *ain* -- Le signal audio en entrée.

*aout* -- Le signal audio en sortie. *kin* -- Le tableau de taux-k en entrée.

## Exemples

Voici un exemple simple de l'opcode *framebuffer*. Il utilise le fichier *framebuffer.csd* [examples/framebuffer.csd].

### Exemple 361. Exemple de l'opcode *framebuffer*.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>
nchnls = 2
Odbfs = 1
ksmps = 128
sr = 44100

instr 1
```

```
    isize init 1024
    ioverlaps init 4

    asig diskin2 "fox.wav", 1, 0, 1
    kframe[] framebuffer asig, isize
    kwindowedFrame[] window kframe, isize

    aout olabuffer kwindowedFrame, ioverlaps
    aout = aout / 2

    outs aout, aout
endin

</CsInstruments>
<CsScore>
i 1 0 400
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*olabuffer shiftin shiftout*

## Crédits

Auteur : Edward Costello;  
NUIM, 2015

Nouveau dans la version 6.06

# freeverb

freeverb — Version opcode de Freeverb de Jezar.

## Description

*freeverb* est une unité de réverbération stéréo basée sur les sources C++ de Jezar du domaine public, composée de huit filtres en peigne en parallèle sur les deux canaux, suivis de quatre unités passe-tout en série. Les filtres du canal de droite sont légèrement déréglés par rapport à ceux du canal de gauche afin de créer un effet stéréo.

## Syntaxe

```
aoutL, aoutR freeverb ainL, ainR, kRoomSize, kHFDamp[, iSRate[, iSkip]]
```

## Initialisation

*iSRate* (facultatif, 44100 par défaut) -- ajuste les paramètres de réverbération pour une utilisation avec le taux d'échantillonnage spécifié (cela affecte la longueur en échantillons des lignes à retard et l'atténuation des hautes fréquences). Seuls des multiples entiers de 44100 reproduiront exactement le caractère original de la réverbération ; il peut ainsi être utile de fixer ce paramètre à 44100 ou à 88200 pour un taux d'échantillonnage de l'orchestre de 48000 ou de 96000 Hz, respectivement. Bien que *iSRate* soit normalement supposé être proche du taux d'échantillonnage de l'orchestre, des valeurs différentes peuvent servir à des effets spéciaux.

*iSkip* (facultatif, 0 par défaut) -- s'il est différent de zéro, l'initialisation de l'opcode sera ignorée, si c'est possible.

## Exécution

*ainL, ainR* -- signaux d'entrée ; habituellement, les deux sont identiques, mais on peut utiliser des entrées différentes pour des effets spéciaux.



### Note

Il est recommandé de traiter les signaux d'entrée avec l'opcode *denorm* afin d'éviter les nombres dénormalisés qui pourraient augmenter la charge CPU de manière significative dans certains cas.

*aoutL, aoutR* -- signaux de sortie pour les canaux gauche et droite.

*kRoomSize* (compris entre 0 et 1) -- contrôle la longueur de la réverbération, une valeur plus importante signifiant une réverbération plus longue. Les valeurs supérieures à 1 peuvent rendre l'opcode instable.

*kHFDamp* (compris entre 0 et 1) -- atténuation des hautes fréquences ; une valeur de zéro signifie que toutes les fréquences décroissent à la même vitesse, tandis que des valeurs supérieures donnent une décroissance plus rapide des hautes fréquences.

## Exemples

Voici un exemple de l'opcode *freeverb*. Il utilise le fichier *freeverb.csd* [examples/freeverb.csd].



### Exemple 362. Un exemple de l'opcode freeverb.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o freeverb.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr      = 44100
ksmps  = 32
nchnls = 2
0dbfs  = 1

instr 1
a1      vco2 0.75, 440, 10
kfrq    port 100, 0.008, 20000
a1      butterlp a1, kfrq
a2      linseg 0, 0.003, 1, 0.01, 0.7, 0.005, 0, 1, 0
a1      = a1 * a2
denorm  a1
aL, aR  freeverb a1, a1, 0.9, 0.35, sr, 0
outs    a1 + aL, a1 + aR
endin

</CsInstruments>
<CsScore>
i 1 0 5
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Istvan Varga  
2005

# ftaudio

ftaudio — Ecrit une table pré-allouée dans un fichier audio.

## Description

Ecrit une table pré-allouée dans un fichier audio sous différents formats.

## Syntaxe

```
ians ftaudio ifn, "filename", iformat
kans ftaudio ktrig, kfn, "filename", \
                                     kformat [, isync, kbegin, kend]
```

## Initialisation

*ifn, kfn* -- Numéro de la table à écrire.

*"filename"* -- Une chaîne de caractères entre guillemets contenant le nom du fichier à sauvegarder.

*iformat, kformat* -- Format du fichier à sauvegarder.

- -1 - le même format que celui fixé par l'indicateur du format de sortie globale de Csound (-A, -W, --format=..., etc).
- 0 - échantillons 32 bit en virgule flottante sans en-tête (fichier binaire PCM multicanal).
- 1 - entiers 16 bit sans en-tête (fichier binaire PCM multicanal).
- 2 - entiers 16 bit avec en-tête. Le type de l'en-tête dépend du format de rendu (-o). Par exemple, si l'utilisateur choisit le format AIFF (en utilisant l'*indicateur* -A), le type du format de l'en-tête sera AIFF.
- 3 - échantillons u-law avec un en-tête (voir iformat=2).
- 4 - entiers 16 bit avec un en-tête (voir iformat=2).
- 5 - entiers 32 bit avec un en-tête (voir iformat=2).
- 6 - nombres en virgule flottante 32 bit avec un en-tête (voir iformat=2).
- 7 - entiers 8 bit non signés avec un en-tête (voir iformat=2).
- 8 - entiers 24 bit avec un en-tête (voir iformat=2).
- 9 - nombre en virgule flottante 64 bit avec un en-tête (voir iformat=2).
- 50 - format ogg-vorbis.

De plus Csound permet de choisir explicitement un type d'en-tête particulier en spécifiant le format par la formule `10 * fileType + sampleFormat`, où `fileType` vaut 1 pour WAV, 2 pour AIFF, 3 pour raw (sans en-tête) et 4 pour IRCAM ; `sampleFormat` prend une des valeurs ci-dessus comprise entre 0 et 9, sauf que le format d'échantillon 0 est celui de la ligne de commande (-o), le format 1 est entiers 8 bit signés et le format 2 est a-law. Ainsi, par exemple, `iformat=25` signifie entiers 32 bit avec un en-tête AIFF.

*isync* -- s'il vaut zéro la version de taux-k attend la fin de l'écriture. S'il est non nul (par défaut) l'écriture des données est déléguée à un fil d'exécution séparé ce qui permet à Csound de continuer l'exécution.

*ibeg, iend, kbeg, kenf* -- donnent le début et la fin de la section de la table où écrire. La valeur par défaut de zéro signifie du début à la fin de la table.

*ians, kans* -- retourne 0 en cas d'échec, 1 sinon. Dans le mode asynchrone c'est mis à jour à la fin de l'écriture, jusqu'à ce qu'il est la valeur -1.

## Exécution

*ktrig* -- la version de taux-k n'est active dans un k-cycle que lorsque ktrig est non nul.

## Exemples

Voici un exemple de l'opcode *ftaudio*. Il utilise le fichier *ftaudio.csd* [examples/ftaudio.csd].

### Exemple 363. Exemple de l'opcode *ftaudio*.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftsave.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 2^10, 10, 1

instr 1
  ktrig init 1
  asig poscil3 .5, 880, giSine
  kans ftaudio ktrig, 100, "k_ftaudio.wav", 15, 1
  ktrig = 0
  outs asig, asig
endin

instr 2
  ians ftaudio 100, "i_ftaudio.wav", 15, p4, p5
  turnoff
endin

</CsInstruments>
<CsScore>
f100 0 0 -1 "beats.wav" 0 0 0
i 1 0 2
i 2 0 1 11025 33075; 0.5 seconds cut
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*ftloadk, ftload, ftsave*

## Crédits

Auteur : John ffitch

Exemple écrit par John ffitch.

Nouveau dans la version 6.12

Limite à un intervalle ajoutée dans la 6.13

# ftchnls

ftchnls — Retourne le nombre de canaux dans un table de fonction en mémoire.

## Description

Retourne le nombre de canaux dans un table de fonction en mémoire.

## Syntaxe

`ftchnls(x)` (arg de taux-i seulement)

## Exécution

Retourne le nombre de canaux d'une table *GEN01*, déterminé par l'en-tête du fichier d'origine. Si le fichier d'origine n'a pas d'en-tête ou si la table n'a pas été créée par *GEN01*, *ftchnls* retourne -1.

## Exemples

Voici un exemple de l'opcode *ftchnls*. Il utilise les fichiers *ftchnls.csd* [exemples/ftchnls.csd], *mary.wav* [exemples/mary.wav] et *kickroll.wav* [exemples/kickroll.wav].

### Exemple 364. Exemple de l'opcode *ftchnls*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
;-odac      ;;realtime audio out
-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ftchnls.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ichnls = ftchnls(p4)
print ichnls

if (ichnls == 1) then
  asigL loscil3 .8, 4, p4
  asigR = asigL
elseif (ichnls == 2) then
  asigL, asigR loscil3 .8, 4, p4
; safety precaution if not mono or stereo
else
  asigL = 0
  asigR = 0
```

```
endif
    outs asigL, asigR

endin
</CsInstruments>
<CsScore>
f 1 0 0 1 "mary.wav" 0 0 0
f 2 0 0 1 "kickroll.wav" 0 0 0

i 1 0 3 1 ;mono file
i 1 + 2 2 ;stereo file
e
</CsScore>
</CsoundSynthesizer>
```

Le fichier audio « mary.wav » étant monophonique et le fichier audio « kickroll.wav » étant stéréophonique, la sortie comprendra des lignes comme celles-ci :

```
instr 1:  ichnls = 1.000
instr 1:  ichnls = 2.000
```

## Voir aussi

*flen, flptim, ftsr, nsamp ftexists*

## Crédits

Auteur : Chris McCormick  
Perth, Australie  
Décembre 2001

# ftconv

ftconv — Convolution multi-canaux à faible latence, utilisant une table de fonction pour la réponse impulsionnelle.

## Description

Convolution multi-canaux à faible latence, utilisant une table de fonction pour la réponse impulsionnelle. L'algorithme divise la réponse impulsionnelle en morceaux dont la longueur est déterminée par le paramètre *iplen*, et retarde et mixe ces morceaux de façon à ce que la réponse impulsionnelle originale soit reconstruite sans lacunes. Le délai de la sortie (latence) est de *iplen* échantillons et ne dépend pas du taux de contrôle, à la différence des autres opcodes de convolution.

## Syntaxe

```
a1[, a2[, a3[, ... a8]]] ftconv ain, ift, iplen[, iskip samples \
[, iirlen[, iskipinit]]]
```

## Initialisation

*ift* -- numéro de la ftable source. La table doit contenir les données audio des différents canaux, entrelacées, avec un nombre de canaux égal au nombre de variables de sortie ((a1, a2, etc.). On peut créer une table entrelacée à partir d'un ensemble de tables mono avec *GEN52*.

*iplen* -- longueur des morceaux de réponse impulsionnelle en trames d'échantillon ; doit être une puissance entière de deux. Avec de faibles valeurs on aura un délai de sortie plus court, mais au prix d'une utilisation accrue du CPU.

*iskipsamples* (facultatif, 0 par défaut) -- nombre de trames d'échantillon à ignorer au début de la table. Utile pour les réponses de réverbération possédant un délai initial. Si ce délai n'est pas inférieur à *iplen* échantillons, en affectant à *iskipsamples* la même valeur que *iplen*, on éliminera toute latence supplémentaire de *ftconv*.

*iirlen* (facultatif) -- longueur totale de la réponse impulsionnelle en trames d'échantillon. Par défaut, on utilise toutes les données de la table (sans le point de garde).

*iskipinit* (facultatif, 0 par défaut) -- s'il a une valeur non nulle, l'initialisation est ignorée lorsque cela est possible sans causer d'erreur.

## Exécution

*ain* -- signal d'entrée

*a1 ... a8* -- signaux de sortie.

## Exemples

Voici un exemple de l'opcode *ftconv*. Il utilise le fichier *ftconv.csd* [examples/ftconv.csd].

### Exemple 365. Exemple de l'opcode *ftconv*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftconv.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr      = 48000
ksmps   = 32
nchnls  = 2
0dbfs   = 1

garvb    init 0
gaW      init 0
gaX      init 0
gaY      init 0

itmp     ftgen 1, 0, 64, -2, 2, 40, -1, -1, -1, 123, \
          1, 13.000, 0.05, 0.85, 20000.0, 0.0, 0.50, 2, \
          1, 2.000, 0.05, 0.85, 20000.0, 0.0, 0.25, 2, \
          1, 16.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2, \
          1, 9.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2, \
          1, 12.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2, \
          1, 8.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2

itmp     ftgen 2, 0, 262144, -2, 0
spat3dt 2, -0.2, 1, 0, 1, 1, 2, 0.005

itmp     ftgen 3, 0, 262144, -52, 3, 2, 0, 4, 2, 1, 4, 2, 2, 4

instr 1

a1       vco2 1, 440, 10
kfrq     port 100, 0.008, 20000
a1       butterlp a1, kfrq
a2       linseg 0, 0.003, 1, 0.01, 0.7, 0.005, 0, 1, 0
a1       = a1 * a2 * 2
denorm a1
vincr garvb, a1
aw, ax, ay, az spat3di a1, p4, p5, p6, 1, 1, 2
vincr gaW, aw
vincr gaX, ax
vincr gaY, ay

endin

instr 2

denorm garvb
; skip as many samples as possible without truncating the IR
arW, arX, arY ftconv garvb, 3, 2048, 2048, (65536 - 2048)
aW       = gaW + arW
aX       = gaX + arX
aY       = gaY + arY
garvb    = 0
gaW      = 0
gaX      = 0
gaY      = 0

aWre, aWim hilbert aW
aXre, aXim hilbert aX
aYre, aYim hilbert aY
aWXr     = 0.0928*aXre + 0.4699*aWre

```



```

aWxiYr = 0.2550*aXim - 0.1710*aWim + 0.3277*aYre
aL      = aWXr + aWxiYr
aR      = aWXr - aWxiYr

    outs aL, aR

    endin

</CsInstruments>
<CsScore>

i 1 0 0.5 0.0 2.0 -0.8
i 1 1 0.5 1.4 1.4 -0.6
i 1 2 0.5 2.0 0.0 -0.4
i 1 3 0.5 1.4 -1.4 -0.2
i 1 4 0.5 0.0 -2.0 0.0
i 1 5 0.5 -1.4 -1.4 0.2
i 1 6 0.5 -2.0 0.0 0.4
i 1 7 0.5 -1.4 1.4 0.6
i 1 8 0.5 0.0 2.0 0.8
i 2 0 10

e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*pconvolve, convolve, dconv.*

## Crédits

Auteur : Istvan Varga  
2005

# ftcps

ftcps — Retourne la fréquence de base d'une table de fonction en Hz.

## Description

Retourne la fréquence de base d'une table de fonction en Hz.

## Syntaxe

**ftcps**(x) (args de taux-i seulement)

## Exécution

Retourne la fréquence de base de la table de fonction en mémoire, numéro *x*. *ftcps* est conçu pour les tables stockant des formes d'onde audio lues depuis des fichiers (voir *GEN01*).

*ftcps* retourne -1 en cas d'erreur (aucune fréquence de base n'est indiquée dans la table ou la table n'existe pas).

## Exemples

Voici un exemple de l'opcode ftcps.

### Exemple 366. Exemple de l'opcode ftcps.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftlen.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Print out the base frequency of Table #1.
; if it has been set in the original file.
icps = ftcps(1)
print icps
endin

</CsInstruments>
```

```
<CsScore>

; Table #1: Use an audio file, Csound will determine its base frequency, if set.
f 1 0 0 1 "sample.wav" 0 0 0

; Play Instrument #1 for 1 second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*fichnls, filptim, ftsr, nsamp*

## Crédits

Auteur : Victor Lazzarini  
2010

Exemple écrit par Victor Lazzarini

# ftexists

ftexists — Cherche si une table donnée existe.

## Description

Opcodes du greffon emugens.

Retourne 1 si l'index donné fait référence à une table existante, 0 si la table n'existe pas. Fonctionne durant l'initialisation si la sortie est une variable de taux-i, sinon au taux-k.

## Syntaxe

```
iexists ftexists ifn  
kexists ftexists kfn / ifn
```

## Arguments

*ifn* / *kfn* - La table à chercher.

## Sortie

*ieuxists* / *kexists* - 1 si la table existe, 0 sinon.

## Exemples

Voici un exemple de l'opcode ftexists. Il utilise le fichier *ftexists.csd* [examples/ftexists.csd].

### Exemple 367. Exemple de l'opcode ftexists.

```
<CsoundSynthesizer>  
<CsOptions>  
  
--nosound  
  
</CsOptions>  
  
<CsInstruments>  
  
; This is the example file for ftexists  
  
/*  
  
  ftexists  
  
  Returns 1 if a given table index refers to an existing  
  ftable  
  
  iexists ftexists ifn  
  kexists ftexists kfn  
  
  Args:  
    ifn / kfn: the table index to query  
*/
```

```

    Returns:
    iexists / kexists: 1 if a table with index ifn exists, 0 otherwise

*/

gifn1 ftgen 0, 0, 8, 2, 0

instr 1
    iexists1 ftexists gifn1
    print iexists1

    iexists2 ftexists 2
    print iexists2

    kexists ftexists 3
    printf "table 3 exists at time %f", kexists, timeinsts()
endin

</CsInstruments>

<CsScore>

f 2 0 8 2 0
f 3 1.5 8 2 0

i 1 0 2

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*filen, nsamp, ftsr, fchnls,*

## Crédits

Par : Eduardo Moguillansky 2019

Nouveau greffon dans la version 6.14

# ftfree

ftfree — Efface une table de fonction.

## Description

Efface une table de fonction.

## Syntaxe

```
ftfree ifno, iwhen
```

## Initialisation

*ifno* -- le numéro de la table à effacer.

*iwhen* -- s'il vaut zéro, la table est effacée pendant la période d'initialisation ; sinon le numéro de table est enregistré pour que celle-ci soit effacée lors de la désactivation de la note.



### Note

Si la table est effacée durant une initialisation, il est possible qu'un opcode antérieur utilisant la table échoue ou se plante pendant un cycle d'exécution. Il est de la responsabilité de l'utilisateur d'éviter cette erreur.

## Exemples

Voici un exemple de l'opcode `ftfree`. Il utilise le fichier `ftfree.csd` [examples/ftfree.csd].

### Exemple 368. Exemple de l'opcode `ftfree`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ftfree.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gitempTable ftgen 0, 0, 65537, 10, 1

instr 1

aout oscili .5, 440, gitempTable
outs aout, aout
```

```

;free temp table at deinit time
ftfree gitempTable, 1
print gitempTable

endin
</CsInstruments>
<CsScore>
f 0 5

i 1 0 .1
i 1 3 1

e
</CsScore>
</CsoundSynthesizer>

instr 1: gitempTable = 101.000
B 0.000 .. 3.000 T 3.000 TT 3.000 M: 0.50000 0.50000
INIT ERROR in instr 1: Invalid ftable no. 101.000000
instr 1: gitempTable = 101.000
Error deleting ftable 101
    B 3.000 - note deleted.  i1 had 1 init errors
B 3.000 .. 5.000 T 5.000 TT 5.000 M: 0.00000 0.00000

```

## Crédits

Auteurs : Steven Yi, Istvan Varga  
2005

# ftgen

ftgen — Génère une table de fonction de partition depuis l'orchestre.

## Description

Génère une table de fonction de partition depuis l'orchestre.

## Syntaxe

```
gir ftgen ifn, itime, isize, igen, iarga [, iargb ] [...]
```

```
gir ftgen ifn, itime, isize, igen, iarray
```

## Initialisation

*gir* -- un numéro de table soit demandé soit assigné automatiquement supérieur à 100.

*ifn* -- numéro de table demandé. Si *ifn* vaut zéro, le numéro est assigné automatiquement et sa valeur est placée dans *gir*. Toute autre valeur est utilisée comme le numéro de la table.

*itime* -- est ignoré, mais il correspond cependant au p2 de l'*instruction f* de partition.

*isize* -- taille de la table. Correspond au p3 de l'*instruction f* de partition.

*igen* -- routine *GEN* de la table de fonction. Correspond au p4 de l'*instruction f* de partition.

*iarga*, *iargb*, ... -- arguments de la table de fonction. Correspondent de p5 à pn de l'*instruction f* de partition.

*iarray* -- un tableau unidimensionnel contenant les arguments de la table de fonction. Correspond de p5 à pn de l'*instruction f* de la partition.

## Exécution

Equivalent à la génération de table dans la partition au moyen de l'*instruction f*.



### Note

A l'origine, Csound était conçu pour ne supporter que les tables dont la taille était une puissance de deux. Bien que ceci ait changé dans les versions récentes (on peut utiliser n'importe quelle taille en donnant un nombre négatif), de nombreux opcodes ne les accepteront pas.



### Avertissement

Bien que Csound ne proteste pas si *ftgen* est utilisé à l'intérieur d'une paire d'instructions *instr-endin*, ce n'est pas une utilisation attendue ni supportée, et celle-ci doit être traitée avec prudence car elle a des effets globaux. En particulier, une taille différente conduit habituellement à une réallocation de la table, ce qui peut causer un plantage ou un comportement erratique.

## Exmples

Voici un exemple de l'opcode ftgen. Il utilise le fichier *ftgen.csd* [examples/ftgen.csd].



### Exemple 369. Exemple de l'opcode ftgen.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ftgen.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisine    ftgen 1, 0, 16384, 10, 1 ;sine wave
gisquare  ftgen 2, 0, 16384, 10, 1, 0, .33, 0, .2, 0, .14, 0, .11, 0, .09 ;odd harmonics
gisaw     ftgen 3, 0, 16384, 10, 0, .2, 0, .4, 0, .6, 0, .8, 0, 1, 0, .8, 0, .6, 0, .4, 0, .2 ;even harmo

instr 1

ifn = p4
asig poscil .6, 200, ifn
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 2 1 ;sine wave
i 1 3 2 2 ;odd harmonics
i 1 6 2 3 ;even harmonics
e
</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode ftgen. Il utilise le fichier *ftgen-2.csd* [examples/ftgen-2.csd].

### Exemple 370. Exemple de l'opcode ftgen.

Cet exemple interroge un fichier sur sa longueur afin de créer une ftable de la taille appropriée.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac            -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ftgen-2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 48000
ksmps   = 16
nchnls  = 2

;Example by Jonathan Murphy 2007
```

```

0dbfs      = 1

    instr 1

    Sfile    = "beats.wav"

    ilen      filelen  Sfile ; Find length
    isr       filesr   Sfile ; Find sample rate

    isamps    = ilen * isr ; Total number of samples
    isize     init     1

loop:
    isize     = isize * 2
; Loop until isize is greater than number of samples
if (isize < isamps) igoto loop

    itab      ftgen     0, 0, isize, 1, Sfile, 0, 0, 0
    print     isize
    print     isamps

    turnoff
    endin

</CsInstruments>
<CsScore>
i1 0 10
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*Routines GEN, ftgentmp*

## Crédits

Auteur : Barry L. Vercoe  
 M.I.T., Cambridge, Mass  
 1997

Avertissement ajouté en Avril 2002 par Rasmus Ekman

# ftgenonce

**ftgenonce** — Génère une table de fonction depuis la définition d'un instrument, sans duplication de données.

## Description

Opcode du greffon `signalflowgraph`.

Permet la création de tables de fonction entièrement à l'intérieur des définitions d'instrument, sans duplication de données.

L'opcode *ftgenonce* est conçu pour simplifier l'écriture des définitions d'instrument qui peuvent être réutilisées dans différents orchestres par un simple *#include* qui les insère dans un instrument. Il n'est pas nécessaire de définir les tables de fonction dans la partition ou dans l'en-tête de l'orchestre.

L'opcode *ftgenonce* est semblable à *ftgentmp* et possède les mêmes arguments. Cependant, les tables de fonctions ne sont ni dupliquées ni effacées. Au lieu de cela, tous les arguments de l'opcode sont assemblés pour former une clé d'accès à une table qui pointe vers le numéro de la table de fonction. Ainsi, chaque invocation de *ftgenonce* avec les mêmes arguments reçoit la même instance des données de la table de fonction. Chaque changement de valeur d'un des arguments de *ftgenonce* provoque la création d'une nouvelle table de fonction.

## Syntaxe

```
ifno ftgenonce ip1dummy, ip2dummy, isize, igen, iarga, iargb, ...
```

## Initialisation

*ifno* -- un numéro de table automatiquement assigné.

*ip1* -- le numéro de la table à générer ou 0 si le numéro doit être assigné.

*ip2dummy* -- ignoré.

*isize* -- taille de la table. Correspond au p3 de l'*instruction f* de partition.

*igen* -- routine *GEN* de la table de fonction. Correspond au p4 de l'*instruction f* de partition.

*iarga, iargb, ...* -- arguments de la table de fonction. Correspondent de p5 à pn de l'*instruction f* de partition.



### Note

A l'origine, Csound était conçu pour ne supporter que les tables dont la taille était une puissance de deux. Bien que ceci ait changé dans les versions récentes (on peut utiliser n'importe quelle taille en donnant un nombre négatif), de nombreux opcodes ne les accepteront pas.

## Exemples

Voici un exemple de l'opcode *ftgenonce*. Il utilise le fichier *ftgenonce.csd* [exemples/*ftgenonce.csd*].

**Exemple 371. Exemple de l'opcode *ftgenonce*.**

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ftgenonce.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
; Use ftgenonce instead of ftgen, ftgentmp, or f statement
iHz = p4
isine ftgenonce 0, 0, 1024, 10, 1
aoscili pluck .7, iHz, 100, isine, 1
aadsr  adsr 0.015, 0.07, 0.6, 0.3
asig = aoscili * aadsr
      outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 1 220
i 1 2 1 261
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Michael Gogins  
2009

# ftgentmp

ftgentmp — Génère une table de fonction de partition depuis l'orchestre, qui est effacée à la fin de la note.

## Description

Génère une table de fonction de partition depuis l'orchestre, qui est facultativement effacée à la fin de la note.

## Syntaxe

```
ifno ftgentmp ip1, ip2dummy, isize, igen, iarga, iargb, ...
```

## Initialisation

*ifno* -- un numéro de table soit demandé soit assigné automatiquement supérieur à 100.

*ip1* -- le numéro de la table à générer ou 0 si le numéro doit être assigné, auquel cas la table est effacée à la fin de la période d'activation de la note.

*ip2dummy* -- ignoré.

*isize* -- taille de la table. Correspond au p3 de l'*instruction f* de partition.

*igen* -- routine *GEN* de la table de fonction. Correspond au p4 de l'*instruction f* de partition.

*iarga, iargb, ...* -- arguments de la table de fonction. Correspondent de p5 à pn de l'*instruction f* de partition.



### Note

A l'origine, Csound était conçu pour ne supporter que les tables dont la taille était une puissance de deux. Bien que ceci ait changé dans les versions récentes (on peut utiliser n'importe quelle taille en donnant un nombre négatif), de nombreux opcodes ne les accepteront pas.

## Exemples

Voici un exemple de l'opcode ftgentmp. Il utilise le fichier *ftgentmp.csd* [examples/ftgentmp.csd].

### Exemple 372. Exemple de l'opcode ftgentmp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2
0dbfs = 1
```

```
instr 1
ifno ftgentmp 0, 0, 512, 10, 1
print ifno
endin
```

```
instr 2
print ftlen(p4)
endin
```

```
</CsInstruments>
<CsScore>
i 1 0 10
i 2 2 1 101
i 1 5 10
i 2 7 1 102
i 2 12 1 101
i 2 17 1 102
e
</CsScore>
</CsoundSynthesizer>
```

La sortie de ce csd montre que les tables sont détruites après la fin des instances d'instrument qui les ont créées, provoquant une erreur d'initialisation si les tables sont demandées.

```
SECTION 1:
new alloc for instr 1:
ftable 101:
instr 1: ifno = 101.000
B 0.000 .. 2.000 T 2.000 TT 2.000 M: 0.00000 0.00000
new alloc for instr 2:
instr 2: #i0 = 512.000
B 2.000 .. 5.000 T 5.001 TT 5.001 M: 0.00000 0.00000
new alloc for instr 1:
ftable 102:
instr 1: ifno = 102.000
B 5.000 .. 7.000 T 7.001 TT 7.001 M: 0.00000 0.00000
instr 2: #i0 = 512.000
B 7.000 .. 12.000 T 11.999 TT 11.999 M: 0.00000 0.00000
INIT ERROR in instr 2: Invalid ftable no. 101.000000
#i0 ftlen.i p4
instr 2: #i0 = -1.000
B 12.000 - note deleted. i2 had 1 init errors
B 12.000 .. 17.000 T 17.000 TT 17.000 M: 0.00000 0.00000
INIT ERROR in instr 2: Invalid ftable no. 102.000000
#i0 ftlen.i p4
instr 2: #i0 = -1.000
B 17.000 - note deleted. i2 had 1 init errors
```

## Crédits

Auteur : Istvan Varga  
2005

# ftlen

ftlen — Retourne la taille d'une table de fonction en mémoire.

## Description

Retourne la taille d'une table de fonction en mémoire.

## Syntaxe

`ftlen(x)` (arg de taux-i seulement)

## Exécution

Retourne la taille (nombre de points, en excluant le point de garde) de la table de fonction numéro *x*. Bien que la plupart des unités faisant référence à une table en mémoire prennent automatiquement en compte sa taille (ce qui permet d'avoir des tables de longueur arbitraire), cette fonction retourne la taille actuelle en cas de besoin. Noter que *ftlen* retourne toujours une puissance de deux, ce qui veut dire que le point de garde de la table de fonction (voir *Instruction f*) n'est pas compris. A partir de Csound 3.53, *ftlen* travaille avec les tables de fonction différées (voir *GEN01*).

*ftlen* diffère de *nsamp* en ce sens que *nsamp* donne le nombre de trames d'échantillon chargées, tandis que *ftlen* donne le nombre total d'échantillons sans le point de garde. Par exemple, avec un fichier son stéréo de 10000 échantillons, *ftlen()* retournera 19999 (c'est-à-dire un total de 20000 échantillons mono, en excluant le point de garde), mais *nsamp()* retournera 10000.

## Exemples

Voici un exemple de l'opcode *ftlen*. Il utilise les fichiers *ftlen.csd* [examples/ftlen.csd], *fox.wav* [examples/fox.wav] et *beats.wav* [examples/beats.wav].

### Exemple 373. Exemple de l'opcode *ftlen*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ftlen.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs  =1

instr 1

ift = ftlen(p4)
```

```
    print ift
aout loscil3 .8, 4, p4
    outs aout, aout

endin
</CsInstruments>
<CsScore>
f 1 0 0 1 "fox.wav" 0 0 0 ;Csound computes tablesiz
f 2 0 0 1 "beats.wav" 0 0 0 ;Csound computes tablesiz

i 1 0 3 1 ;"fox.wav"
i 1 3 3 2 ;"beats.wav"

e
</CsScore>
</CsoundSynthesizer>
```

Le fichier audio « fox.wav » contient 121569 échantillons, « beats.wav » contient 88200 échantillons. L'opcode *filen* retourne des tailles de 121568 et 88199 échantillons car il réserve un point pour le point de garde. Sa sortie comprendra des lignes comme celles-ci :

```
instr 1:  ift = 121568.000
instr 1:  ift = 88199.000
```

## Voir aussi

*ftchnls, filptim, ftsr, nsamp ftexists*

## Crédits

Auteur : Barry L. Vercoe  
MIT  
Cambridge, Massachussetts  
1997



# ftload

ftload — Charge depuis un fichier un ensemble de tables préalablement allouées.

## Description

## Syntaxe

```
ftload Sfilename, iflag, ifn1 [, ifn2] [...]
```

## Initialisation

*Sfilename* -- Une chaîne de caractères contenant le nom du fichier à charger.

*iflag* -- Type du fichier à charger (0 = fichier binaire, différent de 0 = fichier texte).

*ifn1*, *ifn2*, ... -- Numéros des tables à charger.

## Exécution

*ftload* charge une liste de tables depuis un fichier. (Les tables doivent avoir été déjà allouées.) Le format du fichier peut être binaire ou texte.



### Avertissement

Le format du fichier n'est pas compatible avec un fichier WAV et l'ordre des octets (endian-ness) n'est pas sûr.

## Exemples

Voir l'exemple pour *ftsave*.

## Voir aussi

*ftloadk*, *ftsavek*, *ftsave*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.21

# ftloadk

ftloadk — Charge depuis un fichier un ensemble de tables préalablement allouées.

## Description

Charge depuis un fichier un ensemble de tables préalablement allouées.

## Syntaxe

```
ftloadk Sfilename, ktrig, iflag, ifn1 [, ifn2] [...]
```

## Initialisation

*Sfilename* -- Une chaîne de caractères contenant le nom du fichier à charger.

*iflag* -- Type du fichier à charger (0 = fichier binaire, différent de 0 = fichier texte).

*ifn1*, *ifn2*, ... -- Numéros des tables à charger.

## Exécution

*ktrig* -- Le signal de déclenchement. Le fichier est chargé chaque fois que ce signal est différent de zéro.

*ftloadk* charge une liste de tables depuis un fichier. (Les tables doivent avoir été déjà allouées.) Le format du fichier peut être binaire ou texte. A la différence de *ftload*, l'opération de chargement peut-être répétée de nombreuses fois pendant la même note en utilisant un signal de déclenchement.



### Avertissement

Le format du fichier n'est pas compatible avec un fichier WAV et l'ordre des octets (endian-ness) n'est pas sûr.

## Voir aussi

*ftload*, *ftsavek*, *ftsave*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.21

# ftlptim

ftlptim — Retourne la date du début de boucle d'une table de fonction en mémoire.

## Description

Retourne la date du début de boucle d'une table de fonction en mémoire.

## Syntaxe

`ftlptim(x)` (arg de taux-i seulement)

## Exécution

Retourne la date du début de boucle (en secondes) de la table de fonction numéro *x*. La valeur retournée est la durée de l'attaque et du decay directement enregistrés avant le segment de boucle. Retourne zéro (et un message d'avertissement) si l'échantillon ne contient pas de points de boucle.

## Exemples

Voici un exemple de l'opcode `ftlptim`. Il utilise les fichiers *ftlptim.csd* [examples/ftlptim.csd] et *Church.wav* [examples/Church.wav].

### Exemple 374. Exemple de l'opcode `ftlptim`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ftlptim.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs  =1

instr 1

itim = ftlptim(1)
print itim
aout loscil3 .8, 40, 1
outs aout, aout

endin
</CsInstruments>
<CsScore>
f 1 0 0 1 "Church.wav" 0 0 0 ;Csound computes tablesizesize
```

```
i 1 0 5  
e  
</CsScore>  
</CsoundSynthesizer>
```

Le fichier audio « Church.wav » contenant une boucle, la sortie comprendra des lignes comme celles-ci :

```
Base Note : 60 Detune      : 0  
Low  Note : 0 High Note : 127  
Low  Vel. : 0 High Vel.  : 127  
Gain    : 1 Count       : 1  
mode     : 801  
start    : 4529 end      : 4912 count :0  
mode     : 0  
start    : 0 end        : 0 count  :0
```

## Voir aussi

*ftchnls, flen, ftsr, nsamp*

## Crédits

Auteur : Barry L. Vercoe  
MIT  
Cambridge, Massachussetts  
1997

# ftmorf

ftmorf — Fondu enchaîné entre plusieurs ftables données dans une liste.

## Description

Utilise un index dans une table de numéros de ftable pour faire un fondu enchaîné entre les tables voisines dans la liste. La fonction résultante est écrite dans la table référencée par *iresfn* à chaque cycle-k.

## Syntaxe

```
ftmorf kftndx, iftn, iresfn
```

## Initialisation

*iftn* -- la table contenant les numéros des tables existantes qui sont utilisées pour le fondu enchaîné.

*iresfn* -- numéro de table de la fonction résultante.

Toutes les tables référencées dans *iftn* doivent avoir la même longueur que *iresfn*.

## Exécution

*kftndx* -- l'index dans la table *iftn*.

Si *iftn* contient (6, 4, 6, 8, 7, 4):

- *kftndx*=4 écrira le contenu de f7 dans *iresfn*.
- *kftndx*=4.5 écrira la moyenne des contenus de f7 et de f4 dans *iresfn*.



### Note

*iresfn* n'est mise à jour que si l'indice du fondu enchaîné change de valeur. Si *kftndx* est statique, il n'y a pas d'écriture dans *iresfn*.

## Exemples

Voici un exemple de l'opcode ftmorf. Il utilise le fichier *ftmorf.csd* [examples/ftmorf.csd].

### Exemple 375. Exemple de l'opcode ftmorf.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ftmorf.wav -W ;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

  sr = 44100
  ksmps = 32
  nchnls = 2
  0dbfs = 1

  instr 1

  kndx line 0, p3, 7
      ftmorf kndx, 1, 2
  asig oscili .8, 440, 2
      outs asig, asig

  endin
</CsInstruments>
<CsScore>

f1 0 8 -2 3 4 5 6 7 8 9 10
f2 0 1024 10 1 /*contents of f2 dont matter */
f3 0 1024 10 1
f4 0 1024 10 0 1
f5 0 1024 10 0 0 1
f6 0 1024 10 0 0 0 1
f7 0 1024 10 0 0 0 0 1
f8 0 1024 10 0 0 0 0 0 1
f9 0 1024 10 0 0 0 0 0 0 1
f10 0 1024 10 1 1 1 1 1 1 1

i1 0 15
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Author : William « Pete » Moss  
 Université du Texas à Austin  
 Austin, Texas USA  
 Janvier 2002

Nouveau dans la version 4.18

# ftom

ftom — Convertit une fréquence en MIDI.

## Description

Opcode du greffon emugens.

Convertit une fréquence en numéro de note MIDI, en tenant compte de la valeur globale du *la3* (A4).

## Syntaxe

```
imidi ftom ifreq [,irnd]
kmidi ftom kfreq [,irnd]
imidis[] ftom ifreqs[] [,irnd]
kmidis[] ftom kfreqs[] [,irnd]
```

## Exécution

*kfreq* / *ifreq* -- Fréquence.

*irnd* -- Facultatif, s'il est différent de zéro le résultat est arrondi à l'entier le plus proche (zéro par défaut).

*kmidi* / *imidi* -- Numéro de note MIDI correspondant.



### Note

Fixer la valeur globale du *la3* (A4) dans l'en-tête pour modifier l'accordage.

## Exemples

Voici un exemple de l'opcode ftom. Il utilise le fichier *mtof-ftom.csd* [examples/mtof-ftom.csd].

### Exemple 376. Exemple de l'opcode ftom.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2
0dbfs = 1.0
A4 = 440

instr 1
  kfreq = mtof(69)
  printks2 "midi 69    -> %f\n", kfreq

  kmidi = ftom(442)
```

```

printks2 "freq 442 -> %f\n", kmidi

kmidi = ftom(442,1)
printks2 "freq 442 -> %f rounded\n", kmidi

kfreq = mtof(kmidi)
printks "midi %f -> %f\n", 1, kmidi, kfreq

imidi = ftom:i(440)
print imidi

ifreq = mtof:i(60)
print ifreq

turnoff
endin

instr 2
  imidis0[] fillarray 60, 62, 64, 69
  ifreqs0[] mtof imidis0
  printarray ifreqs0, "", "ifreqs0"

  kfreqs[] fillarray 220, 440, 880
  kmidis[] ftom kfreqs
  puts "kfreqs", 1
  printarray kmidis, 1, "%.2f", "kmidis"
  turnoff
endin

</CsInstruments>
<CsScore>
i 1 0 1
i 2 0 1
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*cpsmidinn, mtof*

## Crédits

Par : Eduardo Moguillansky 2017

Nouveau dans la version 6.09

Option d'arrondi ajoutée dans la 6.12

Version tableau ajoutée dans la 6.13



# ftprint

ftprint — Affiche le contenu d'une table de fonction (pour débogage).

## Description

Opcodes du greffon emugens.

Affiche le contenu d'une table de fonction (pour débogage). Fonctionne au taux-k, un déclencheur contrôlant quand afficher (utiliser ktrig=1, qui est la valeur par défaut, pour n'afficher qu'à l'initialisation).

## Syntaxe

```
ftprint ifn [, ktrig, kstart, kend, kstep, inumcols ]
```

## Initialisation

*ifn* -- La table à afficher.

*inumcols* -- Le nombre d'éléments à afficher dans une colonne (10 par défaut).

## Exécution

*ktrig* -- La table sera affichée chaque fois que cette valeur change de 0 à un nombre positif. On peut l'utiliser avec *metro* pour afficher à intervalle donné. Une valeur de -1 indique d'afficher à chaque cycle-k (vaut 1 par défaut).

*kstart* -- La première position à afficher (0 par défaut).

*kend* -- La dernière position à afficher (non incluse). (Par défaut = la longueur de la table).

*kstep* -- Le nombre d'éléments à ignorer (1 par défaut)

## Exemples

Voici un exemple de l'opcode ftprint. Il utilise le fichier *ftprint.csd* [examples/ftprint.csd].

### Exemple 377. Exemple de l'opcode ftprint.

```
<CsoundSynthesizer>
<CsOptions>

--nosound

</CsOptions>
<CsInstruments>

; This is the example file for ftprint

/*

    ftprint
```

```

Print the contents of an f-table
(mostly for debuggin purposes)

ftprint ifn, ktrig=1, kstart=0, kend=0, kstep=1, inumcols=0

ifn: the table to print
ktrig: table will be printed whenever this changes
      from non-positive to positive
kstart: start index
kend: end index (non inclusive)
kstep: number of elements to skip
inumcols: number of elements to print per line

See also: printarray

*/

instr 1
  ifn  ftgentmp 0, 0, -13, -2, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

  ; print all elements
  ftprint ifn

  ; print all elements in columns of 4 elements
  ftprint ifn, 1, 0, 0, 1, 4

  ; assume that a table holds a 2D matrix, print the matrix
  imatrix ftgentmp 0, 0, 0, -2, \
    00, 01, 02, 03, 04, \
    10, 11, 12, 13, 14, \
    20, 21, 22, 23, 24, \
    30, 31, 32, 33, 34

  ; print the whole matrix, 5 columns per line
  ftprint imatrix, 1, 0, 0, 1, 5

  ; print one row
  irow = 2
  inumcols = 5
  ftprint imatrix, 1, 2*inumcols, 3*inumcols

  ; print one column
  ftprint imatrix, 1, 3, 0, inumcols, 1

  turnoff
endin

</CsInstruments>

<CsScore>
i 1 0 0.01
</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

*printarray, printf*

## Crédits

Auteur : Eduardo Moguillansky 2018

Nouveau greffon dans la version 6.12

# ftsamplebank

ftsamplebank — Lit les fichiers son d'un répertoire.

## Description

Opcode du greffon ftsamplebank.

Lit les fichiers son d'un répertoire et les charge dans une série de table de fonction GEN01.

## Syntaxe

```
iNumberOfFile ftsamplebank SDirectory, iFirstTableNumber, iSkipTime, iFormat, iChannel,  
kNumberOfFile ftsamplebank SDirectory, kFirstTableNumber, kTrigger, kSkipTime, kFormat, kChannel,
```

## Initialisation

*SDirectory* -- une chaîne de caractères identifiant le répertoire à parcourir.

*FirstTableNumber* -- fixe le numéro de la première table dans laquelle un fichier son sera chargé.

*kTrigger* -- les tables sont mises à jour lorsqu'il vaut 1, ne s'applique qu'à la version de taux-k.

*SkipTime* -- la lecture commence à *skiptime* secondes dans le fichier.

*Format* -- indique le format des données du fichier audio :

1 - caractères signés sur 8 bit	4 - entiers courts sur 16 bit
2 - octets A-law	5 - entiers longs sur 32 bit
3 - octets U-law	6 - flottants sur 32 bit

*Channel* -- numéro du canal à lire. 0 signifie tous les canaux.

Si *Format* = 0 le format des échantillons est déduit de l'en-tête du fichier ou, par défaut, de l'option de ligne de commande *-o* de Csound.

## Exécution

*iNumberOfFile* -- le nombre de tables créées.

*kNumberOfFile* -- le nombre de tables créées.



### Note

Le chargement au taux-k d'un grand nombre de fichiers dans des tables de fonction peut causer des interruptions dans le flux audio.

## Exemple

Cet exemple montre ftsamplebank cherchant des échantillons à un endroit donné. Il charge tous les échantillons qu'il trouve dans des tables de fonction GEN01 et les joue ensuite en séquence, un par seconde. Il utilise le fichier *ftsamplebank.csd* [examples/ftsamplebank.csd].

**Exemple 378. Exemple de l'opcode ftsamplebank.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime output leave only the line below:
; -o diskin.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
Odbfs = 1

;load all samples in a given directory into function tables and play them using instrument 1000
instr 1
  iFirstTableNumber = 60;
  iFileCount init 1
  iNumberOfFiles ftsamplbank "../examples/", iFirstTableNumber, 0, 4, 1

  until iFileCount>=iNumberOfFiles do
event_i "i", 1000, iFileCount, 1, iFirstTableNumber+iFileCount
    iFileCount = iFileCount+1
  enduntil

endin

instr 1000
  iTable = p4
  aOut loscil3 1, 1, iTable, 1, 0;
  outs aOut, aOut
endin

</CsInstruments>
<CsScore>
i1 0 20
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Rory Walsh  
2015

# ftsave

ftsave — Sauvegarde dans un fichier un ensemble de tables préalablement allouées.

## Description

Sauvegarde dans un fichier un ensemble de tables préalablement allouées.

## Syntaxe

```
ftsave "filename", iflag, ifn1 [, ifn2] [...]
```

## Initialisation

*"filename"* -- Une chaîne de caractères entre guillemets contenant le nom du fichier à sauvegarder.

*iflag* -- Type du fichier à sauvegarder (0 = binaire, différent de 0 = fichier texte).

*ifn1*, *ifn2*, ... -- Numéros des tables à sauvegarder.

## Exécution

*ftsave* sauvegarde une liste de tables dans un fichier. Le format du fichier peut être binaire ou texte.



### Avertissement

Le format du fichier n'est pas compatible avec un fichier WAV et l'ordre des octets (endian-ness) n'est pas sûr.

## Exemples

Voici un exemple de l'opcode *ftsave*. Il utilise le fichier *ftsave.csd* [examples/ftsave.csd].

### Exemple 379. Exemple de l'opcode *ftsave*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o ftsave.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Table #1, make a sine wave using the GEN10 routine.
gitmp1 ftgen 1, 0, 32768, 10, 1
; Table #2, create an empty table.
gitmp2 ftgen 2, 0, 32768, 7, 0, 32768, 0

; Instrument #1 - a basic oscillator.
instr 1
  kamp = 20000
  kcps = 440
  ; Use Table #1.
  ifn = 1

  a1 oscil kamp, kcps, ifn
  out a1
endin

; Instrument #2 - Load Table #1 into Table #2.
instr 2
  ; Save Table #1 to a file called "table1.ftsave".
  ftsave "table1.ftsave", 0, 1

  ; Load the "table1.ftsave" file into Table #2.
  ftload "table1.ftsave", 0, 2

  kamp = 20000
  kcps = 440
  ; Use Table #2, it should contain Table #1's sine wave now.
  ifn = 2

  a1 oscil kamp, kcps, ifn
  out a1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 1 second.
i 1 0 1
; Play Instrument #2 for 1 second.
i 2 2 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*ftloadk, ftload, ftsavek*

## Crédits

Auteur : Gabriel Maldonado

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.21

# ftsavek

ftsavek — Sauvegarde dans un fichier un ensemble de tables préalablement allouées.

## Description

Sauvegarde dans un fichier un ensemble de tables préalablement allouées.

## Syntaxe

```
ftsavek "filename", ktrig, iflag, ifn1 [, ifn2] [...]
```

## Initialisation

*"filename"* -- Une chaîne de caractères entre guillemets contenant le nom du fichier à sauvegarder.

*iflag* -- Type du fichier à sauvegarder (0 = binaire, différent de 0 = fichier texte).

*ifn1*, *ifn2*, ... -- Numéros des tables à sauvegarder.

## Exécution

*ktrig* -- Le signal de déclenchement. Le fichier est sauvegardé chaque fois que ce signal est différent de zéro.

*ftsavek* sauvegarde une liste de tables dans un fichier. Le format du fichier peut être binaire ou texte. A la différence de *ftsav*, l'opération de sauvegarde peut-être répétée de nombreuses fois pendant la même note en utilisant un signal de déclenchement.



### Avertissement

Le format du fichier n'est pas compatible avec un fichier WAV et l'ordre des octets (endian-ness) n'est pas sûr.

## Voir aussi

*ftloadk*, *ftload*, *ftsav*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.21

# ftslice

ftslice — Copie un bout d'une ftable dans une autre.

## Description

Opcode du greffon emugens.

L'opcode *ftslice* prend une ftable et en copie un bout dans une seconde ftable (comme *tab2array*, mais entre tables de fonction).

## Syntaxe

```
ftslice ifnsource, ifndest [, kstart, kend, kstep ]
```

## Initialisation

*ifnsource* -- Le numéro de la table source.

*ifndest* -- Le numéro de la table destination.

## Exécution

*kstart* -- La position à partir de laquelle copier. 0 par défaut.

*kend* -- La position à partir de laquelle la copie est stoppée. Elle n'est PAS inclusive. 0 indique de copier jusqu'à la fin de la table. Par défaut = la fin de la table.

*kstep* -- Le nombre d'éléments à ignorer. 1 par défaut.

## Exemples

Voici une exemple de l'opcode ftslice. Il utilise le fichier *ftslice.csd* [examples/ftslice.csd].

### Exemple 380. Exemple de l'opcode ftslice.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

; Example file ftslice

/*

ftslice:

Copy slice from source table to destination table

Syntax:

ftslice ifnsource, ifndest, kstart=0, kend=0, kstep=1
```



```

ifnsource: source table
ifndest: destination table
kstart: the index to start copying from
kend: the end index to stop copying. This is NOT inclusive. 0=end of table
kstep: how many elements to skip

See also: tablecopy, tableicopy, tab2array
*/

instr 1
  ifn  ftgentmp 0, 0, -13, -2, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
  idest ftgentmp 0, 0, -11, -2, 0 ; empty table of size 11

  ; copy only even elements
  ftslice ifn, idest, 0, 0, 2
  ftprint idest

  ; copy too many elements - only the elements which fit in the dest table
  ; are copied

  ftslice ifn, idest
  ftprint idest

  turnoff
endin

</CsInstruments>
<CsScore>
i 1 0 0.1

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*slicearray, copyf2array, tab2array, tablecopy, tableicopy, ftprint*

## Crédits

Auteur : Eduardo Moguillansky 2018

Nouveau greffon dans la version 6.12

# ftsr

ftsr — Retourne le taux d'échantillonnage d'une table de fonction en mémoire.

## Description

Retourne le taux d'échantillonnage d'une table de fonction en mémoire.

## Syntaxe

**ftsr**(x) (arg de taux-i seulement)

## Exécution

Retourne le taux d'échantillonnage d'une table générée par *GEN01*. Le taux d'échantillonnage est déterminé à partir de l'en-tête du fichier original. Si ce dernier n'a pas d'en-tête ou si la table n'a pas été créée avec *GEN01*, *ftsr* retourne 0. Nouveau dans la version 3.49 de Csound.

## Exemples

Voici un exemple de l'opcode ftsr. Il utilise le fichier *ftsr.csd* [exemples/ftsr.csd].

### Exemple 381. Exemple de l'opcode ftsr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ftsr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

itab = p4
isr = ftsr(itab)
prints "sampling-rate of table number %d = %d\n", itab, isr

endin
</CsInstruments>
<CsScore>
f 1 0 0 1 "kickroll.wav" 0 0 0 ;stereo file
f 2 0 0 1 "ahh.aiff" 0 0 0 ;& different formats
f 3 0 0 1 "beats.mp3" 0 0 0
f 4 0 0 1 "beats.ogg" 0 0 0
```

```
i 1 0 1 1
i 1 + 1 2
i 1 + 1 3
i 1 + 1 4
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
+sampling-rate of table number 1 = 44100
+sampling-rate of table number 2 = 22050
+sampling-rate of table number 3 = 44100
+sampling-rate of table number 4 = 44100
```

## Voir aussi

*fichnls, flen, flptim, nsamp*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Octobre 1998

# gain

gain — Ajuste l'amplitude d'un signal audio en fonction d'une valeur efficace.

## Description

Ajuste l'amplitude d'un signal audio en fonction d'une valeur efficace.

## Syntaxe

```
ares gain asig, krms [, ihp] [, iskip]
```

## Initialisation

*ihp* (facultatif, 10 par défaut) -- point à mi-puissance (en Hz) d'un d'un filtre passe-bas interne spécial. La valeur par défaut est 10.

*iskip* (facultatif, 0 par défaut) -- disposition initiale de l'espace de données interne (voir *reson*). La valeur par défaut est 0.

## Exécution

*asig* -- signal audio en entrée

*gain* effectue une modification d'amplitude de *asig* de sorte que la sortie *ares* ait pour valeur effice *krms*. *rms* et *gain* utilisés conjointement (et avec des valeurs de *ihp* correspondantes) produiront le même effet que *balance*.

## Exemples

Voici un exemple de l'opcode gain. Il utilise le fichier *gain.csd* [examples/gain.csd].

### Exemple 382. Exemple de l'opcode gain.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gain.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
```

```

asrc buzz .8, 440, sr/440, 1 ; band-limited pulse train.
a1 reson asrc, 1000, 100 ; Sent through
a2 reson a1, 3000, 500 ; 2 filters
krms rms asrc ; then balanced
afin gain a2, krms ; with source
outs afin, afin
endin

</CsInstruments>
<CsScore>
; sine wave.
f 1 0 16384 10 1

i 1 0 2
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*balance, rms*

# gainslider

**gainslider** — Une implémentation de courbe de gain logarithmique qui est semblable à l'objet `gainslider~` de Cycling 74 Max / MSP.

## Description

Cet opcode sert à être multiplié par un signal audio pour donner la même impression qu'avec une console de mixage. Il n'y a pas de limites dans le code source si bien que l'on peut par exemple donner des valeurs supérieures à 127 pour obtenir un signal audio plus fort mais avec un risque d'écèlement.

## Syntaxe

`kout gainslider kindex`

## Exécution

*kindex* -- Valeur d'indice. Intervalle nominal de 0 à 127. Par exemple un intervalle de 0 à 152 donnera un intervalle de  $-\infty$  à +18,0 dB.

*kout* -- Sortie pondérée.

## Exemples

Voici un exemple de l'opcode `gainslider`. Il utilise le fichier `gainslider.csd` [exemples/gainslider.csd].

### Exemple 383. Exemple de l'opcode gainslider.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  Silent
-odac          -iadc    -d      ;;realtime output
</CsOptions>
<CsInstruments>

sr = 48000
ksmps = 100
nchnls = 2

instr 1 ; gainslider test

; uncomment for realtime midi
;kmod ctrl7 1, 1, 0, 127

; uncomment for non realtime
kmod phasor 1/10
kmod scale kmod, 127, 0

kout gainslider kmod

printks "kmod = %f kout = %f\n", 0.1, kmod, kout
```

```

aout diskin2 "fox.wav", 1, 0, 1

aout = aout*kout

outs aout, aout

    endin

</CsInstruments>
<CsScore>
i1 0 30
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*scale, logcurve, expcurve*

## Crédits

Auteur : David Akbari  
Octobre  
2006

# gtf

gtf — Applique un filtre gammatone à un signal audio.

## Description

Applique un filtre gammatone de différents ordres à un signal audio.

## Syntaxe

```
aout gtf ain, kfreq, idecay[, iorder, iphase]
```

## Initialisation

*idecay* -- taux de chute

*iorder* -- (facultatif) Ordre du filtre compris entre 1 et 10 (valeurs entières). Vaut 4 par défaut.

*iphase* -- (facultatif) Phase en sortie, zéro par défaut.

## Exécution

*asig* -- Signal audio à filtrer.

*kfreq* -- Fréquence centrale du filtre en Hz.

Le filtre gammatone est habituellement utilisé dans des modèles du système auditif. L'algorithme est basé sur les travaux présentés dans la thèse de doctorat de Martin Cooke (Cooke, 1993) qui utilisent la transformation invariante d'impulsion en bande de base. Voir <http://staffwww.dcs.shef.ac.uk/people/N.Ma/re-sources/gammatone/>

## Crédits

Par : John ffitich 2019



# gauss

gauss — Générateur de nombres aléatoires de distribution gaussienne.

## Description

Générateur de nombres aléatoires de distribution gaussienne. C'est un générateur de bruit de classe x.

## Syntaxe

```
ares gauss krange  
ires gauss krange  
kres gauss krange
```

## Exécution

*krange* -- l'intervalle des nombres aléatoires (*-krange* à *+krange*). Produit des nombres positifs et négatifs.

*gauss* retourne des nombres aléatoires suivant une distribution normale centrée sur 0 ( $\mu = 0.0$ ) avec une variance (sigma) de *krange* / 3.83. Ainsi plus de 99.99% des valeurs aléatoires générées sont comprises entre *-krange* et *+krange*. Si l'on veut une valeur moyenne différente de 0.0, il faut ajouter cette valeur moyenne à chaque nombre généré (voir l'exemple ci-dessous).

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode gauss. Il utilise le fichier *gauss.csd* [examples/gauss.csd].

### Exemple 384. Exemple de l'opcode gauss.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
  
<CsOptions>  
-d -o dac  
</CsOptions>  
  
<CsInstruments>  
instr 1  
irange = p4  
imu    = p5  
isamples = p6  
indx    = 0  
icount  = 1
```

```

ix      = 0.0
ix2     = 0.0

loop:
i1      gauss   irange
i1      =       i1 + imu
ix      =       ix + i1
ix2     =       ix2 + i1*i1
if i1 >= -(irange+imu) && i1 <= (irange+imu) then
    icount = icount+1
endif
        loop_lt indx, 1, isamples, loop

imean   =       ix / isamples                ;mean value
istd    =       sqrt(ix2/isamples - imean*imean) ;standard deviation
prints  "mean = %3.3f, std = %3.3f, ", imean, istd
prints  "samples inside the given range: %3.3f\\n", icount*100.0/isamples

endin
</CsInstruments>
<CsScore>
i 1 0 0.1 1.0 0 100000 ; range = 1, mu = 0.0, sigma = 1/3.83 = 0.261
i 1 0.1 0.1 3.83 0 100000 ; range = 3.83, mu = 0.0, sigma = 1
i 1 0.2 0.1 5.745 2.7 100000 ; range = 5.745, mu = 2.7, sigma = 5.745/3.83 = 1.5
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des ligne comme celles-ci :

```

mean = 0.000, std = 0.260, samples inside the given range: 99.993%
mean = 0.005, std = 0.999, samples inside the given range: 99.998%
mean = 2.700, std = 1.497, samples inside the given range: 100.000%

```

## Voir aussi

*seed, betarand, bexprnd, cauchy, exprand, linrand, pcauchy, poisson, trirand, unirand, weibull*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Précisions sur mu et sigma ajoutées par François Pinot après une discussion avec Joachim Heintz sur la liste de Csound, Décembre 2010.

Exemple écrit par François Pinot, adapté d'un fichier csd de Joachim Heintz, Décembre 2010.

Existait dans la version 3.30

# gaussi

gaussi — Générateur de nombres aléatoires à distribution gaussienne avec interpolation.

## Description

Générateur de nombres aléatoires à distribution gaussienne avec interpolation contrôlée entre les valeurs. C'est un générateur de bruit de classe x.

## Syntaxe

ares **gaussi** krange, xamp, xcps

ires **gaussi** krange, xamp, xcps

kres **gaussi** krange, xamp, xcps

## Exécution

*krange* -- l'intervalle des nombres aléatoires (*-krange* à *+krange*). Produit des nombres positifs et négatifs.

*gaussi* retourne des nombres aléatoires suivant une distribution normale centrée sur 0 ( $\mu = 0.0$ ) avec une variance (sigma) de *krange* / 3.83. Ainsi plus de 99.99% des valeurs aléatoires générées sont comprises entre *-krange* et *+krange*. Si l'on veut une valeur moyenne différente de 0.0, il faut ajouter cette valeur moyenne à chaque nombre généré (voir l'exemple ci-dessous).

Pour des explications plus détaillées, voir :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

*xamp* -- intervalle de distribution des nombres aléatoires.

*xcps* -- fréquence à laquelle de nouveaux nombres sont générés.

## Exemples

Voici un exemple de l'opcode *gaussi*. Il utilise le fichier *gaussi.csd* [examples/gaussi.csd].

### Exemple 385. Exemple de l'opcode *gaussi*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o exprand.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
klambda gaussi 100, 1, 3
  printk2 klambda ; look
aout oscili 0.8, 440+klambda, 1 ; & listen
  outs aout, aout
endin

</CsInstruments>
<CsScore>
; sine wave
f 1 0 16384 10 1

i 1 0 4
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*gauss*

## Crédits

Auteur : John ffitch  
 Bath  
 Mai 2011  
 Nouveau dans la version 5.14

# gausstrig

gausstrig — Impulsions aléatoires autour d'une fréquence donnée.

## Description

Génère des impulsions aléatoires autour d'une fréquence donnée.

## Syntaxe

```
ares gausstrig kamp, kcps, kdev [, imode] [, ifrst1]
kres gausstrig kamp, kcps, kdev [, imode] [, ifrst1]
```

## Initialisation

*imode* (facultatif, 0 par défaut) -- *imode* > 0 signifie une meilleure modulation de fréquence. Si la fréquence change, le délai avant l'impulsion suivante est recalculé. Avec le mode par défaut, nous avons le comportement classique du générateur unitaire GaussTrig dans SuperCollider, où la modulation de fréquence est ignorée durant le délai précédant l'impulsion suivante.

*ifrst1* (facultatif, 0 par défaut) -- *ifrst1* > 0 change le comportement original du générateur unitaire Gauss-Trig. Par défaut, celui-ci génère toujours une impulsion au tout début. Ici, l'apparition de la première impulsion est aléatoire et dépend des paramètres *kcps* et *kdev* parameters.

## Exécution

*kamp* -- amplitude.

*kcps* -- la fréquence moyenne autour de laquelle sont distribuées les impulsions aléatoires.

*kdev* -- déviation aléatoire autour de la moyenne ( $0 \leq dev < 1$ ).

## Exemples

Voici un exemple de l'opcode gausstrig. Il utilise le fichier *gausstrig.csd* [examples/gausstrig.csd].

### Exemple 386. Exemple de l'opcode gausstrig.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o oscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kdev line 0, p3, 0.9
seed 20120125
aimp gausstrig 0.5, 10, kdev
aenv filter2 aimp, 1, 1, 0.993, 0.993
anoi fractalnoise 0.2, 1.7
al = anoi*aenv
ar delay al, 0.02
outs al, ar

endin
</CsInstruments>
<CsScore>
i1 0 10
e
</CsScore>
</CsoundSynthesizer>

```

Voici un exemple de l'opcode `gausstrig` avec `imode = 1`. Il utilise le fichier `gausstrig-2.csd` [exemples/ gausstrig-2.csd].

### Exemple 387. Exemple de l'opcode `gausstrig` avec `imode = 1`.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o oscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kfrq0 oscil 8, 0.25, 1
ktrig metro 1
kfrq samphold kfrq0+8, ktrig
seed 20120125
aimp gausstrig 0.5, kfrq, 0.5, 1
aenv filter2 aimp, 1, 1, 0.993, 0.993
anoi fractalnoise 0.2, 1.7
al = anoi*aenv
ar delay al, 0.02
outs al, ar

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1
i1 0 16
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*dust dust2 mpulse*

## Crédits

Basé sur le générateur unitaire GaussTrig de Bhob Rainey (SuperCollider)

Auteur : Tito Latini

Janvier 2012

Modification : Gleb Rogozinsky

Mars 2015

Nouveau dans la version 5.16 de Csound.

# gbuzz

*gbuzz* — La sortie est un ensemble de partiels cosinus en relation harmonique.

## Description

La sortie est un ensemble de partiels cosinus en relation harmonique.

## Syntaxe

```
ares gbuzz xamp, xcps, knh, klh, kmul, ifn [, iphs]
```

## Initialisation

*ifn* -- numéro de table d'une fonction stockée contenant une onde cosinus. Une grande table d'au moins 8192 points est recommandée.

*iphs* (facultatif, par défaut 0) -- phase initiale de la fréquence fondamentale, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative l'initialisation de la phase sera ignorée. La valeur par défaut est zéro.

## Exécution

Les unités *buzz* génèrent un ensemble additif de partiels cosinus en relation harmonique de fréquence fondamentale *xcps*, et dont les amplitudes sont pondérées de telle façon que la crête de leur somme égale *xamp*. Le choix et l'importance des partiels sont déterminés par les paramètres de contrôle suivants :

*knh* -- nombre total d'harmoniques demandés. Si *knh* est négatif, sa valeur absolue est utilisée. Si *knh* vaut zéro, une valeur de 1 est utilisée.

*klh* -- harmonique présent le plus bas. Peut être positif, nul ou négatif. Dans *gbuzz* l'ensemble de partiels peut commencer à n'importe quel numéro de partiel et se complète vers le haut ; si *klh* est négatif, tous les partiels en-dessous de zéro seront repliés comme des partiels positifs sans changement de phase (car le cosinus est une fonction paire), et s'ajouteront de façon constructive aux partiels positifs de l'ensemble.

*kmul* -- spécifie la raison de la série des coefficients d'amplitude. C'est une série entière : si le *klh*ème partiel a pour coefficient A, le (*klh* + n)ème partiel aura pour coefficient  $A * (kmul ** n)$ , c'est-à-dire que les valeurs d'intensité dessinent une courbe exponentielle. *kmul* peut être positif, nul ou négatif, et n'est pas restreint aux valeurs entières.

*buzz* et *gbuzz* sont utiles comme sources de son complexe dans la synthèse soustractive. *buzz* est un cas particulier du plus général *gbuzz* dans lequel *klh* = *kmul* = 1 ; il produit ainsi un ensemble de *knh* harmoniques de même importance, commençant avec le fondamental. (C'est un train d'impulsions à bande de fréquence limitée ; si les partiels vont jusqu'à la fréquence de Nyquist, c'est-à-dire  $knh = \text{int}(sr / 2 / \text{fréq. fondamentale})$ , le résultat est un train d'impulsions réelles d'amplitude *xamp*.)

Bien que l'on puisse faire varier *knh* et *klh* durant l'exécution, leurs valeurs internes sont nécessairement entières ce qui peut provoquer des « pops » dûs à des discontinuités dans la sortie. Cependant, la variation de *kmul* durant l'exécution produit un bon effet. *gbuzz* peut être modulé en amplitude et/ou en fréquence soit par des signaux de contrôle soit par des signaux audio.

Nota Bene : cette unité a son pendant avec *GENII*, dans lequel le même ensemble de cosinus peut être stocké dans une table de fonction qui sera lue par un oscillateur. Bien que plus efficace en termes de calcul,



le train d'impulsions stocké a un contenu spectral fixe, non variable dans le temps comme celui décrit ci-dessus.

## Exemples

Voici un exemple de l'opcode `gbuzz`. Il utilise le fichier `gbuzz.csd` [examples/gbuzz.csd].

### Exemple 388. Exemple de l'opcode `gbuzz`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gbuzz.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kcps = 220
knh = p4 ;total no. of harmonics
klh = p5 ;lowest harmonic
kmul line 0, p3, 1 ;increase amplitude of
;higher partials
asig gbuzz .6, kcps, knh, klh, kmul, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
; a cosine wave
f 1 0 16384 11 1

i 1 0 3 3 1 ;3 harmonics, lowest harmonic=1
i 1 + 3 30 1 ;30 harmonics, lowest harmonic=1
i 1 + 3 3 2 ;3 harmonics, lowest harmonic=3
i 1 + 3 30 2 ;30 harmonics, lowest harmonic=3
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

`buzz`

## Crédits

Septembre 2003. Merci à Kanata Motohashi pour avoir corrigé les mentions du paramètre `kmul`.

# genarray

genarray — Génère un vecteur à partir d'une suite arithmétique.

## Description

Génère un vecteur (tableau unidimensionnel de taux-k ou de taux-i) à partir d'une suite arithmétique.

## Syntaxe

```
karray genarray kstart, kend[, inc]
```

```
iarray genarray istart, iens[, inc]
```

## Initialisation

*istart* -- indice dans le tableau où placer le premier élément.

*iend* -- indice dans le tableau où placer le dernier élément.

*inc* -- incrément entre deux valeurs (1 par défaut).

## Exemples

Voici un exemple de l'opcode genarray. Il utilise le fichier *genarray.csd* [examples/genarray.csd].

### Exemple 389. Exemple de l'opcode genarray.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

;create and fill two arrays
iArr1[] genarray 1, 6
iArr2[] genarray 1, 6, 2/3

;print the content of iArr1
printf "%s", 1, "iArr1: start=1, end=6, step=default\n"
kndx = 0
until kndx == lenarray(iArr1) do
  printf "iArr[%d] = %f\n", kndx+1, kndx, iArr1[kndx]
  kndx += 1
od
```

```

;print the content of iArr2
    printf "%s", 1, "iArr2: start=1, end=6, step=2/3\n"
kndx = 0
    until kndx == lenarray(iArr2) do
        printf "iArr[%d] = %f\n", kndx+1, kndx, iArr2[kndx]
        kndx += 1
    od

    turnoff
endin

</CsInstruments>
<CsScore>
i 1 0 1
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

Opcodes vectoriels

## Crédits

Auteur : John ffitch  
 Codemist Ltd  
 2013

Nouveau dans la version 6.00

# genarray\_i

genarray\_i — Génère un vecteur à partir d'une suite arithmétique.

## Description

Génère un vecteur (tableau unidimensionnel de taux-k) à partir d'une suite arithmétique durant l'initialisation.

## Syntaxe

```
karray genarray_i istart, iend[, inc]
```

## Initialisation

*istart* -- indice dans le tableau où placer le premier élément.

*iend* -- indice dans le tableau où placer le dernier élément.

*inc* -- incrément entre deux valeurs (1 par défaut).

## Exemples

Voici un exemple de l'opcode genarray\_i. Il utilise le fichier *genarray\_i.csd* [exemples/genarray\_i.csd].

### Exemple 390. Exemple de l'opcode genarray\_i.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

;create and fill two arrays
kArr1[] genarray_i 1, 6
kArr2[] genarray_i 1, 6, 2/3

;print the content of kArr1
printf "%s", 1, "kArr1: start=1, end=6, step=default\n"
kndx = 0
until kndx == lenarray(kArr1) do
printf "kArr[%d] = %f\n", kndx+1, kndx, kArr1[kndx]
kndx += 1
od
```

```

;print the content of kArr2
    printf "%s", 1, "kArr2: start=1, end=6, step=2/3\n"
kndx = 0
    until kndx == lenarray(kArr2) do
        printf "kArr[%d] = %f\n", kndx+1, kndx, kArr2[kndx]
        kndx += 1
    od

    turnoff
endin

</CsInstruments>
<CsScore>
i 1 0 1
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

Opcodes vectoriels, *fillarray*

## Crédits

Auteur : John ffitch  
 Codemist Ltd  
 2013

Nouveau dans la version 6.00

# gendy

gendy — Approche dynamique stochastique de la synthèse de forme d'onde conçue par Iannis Xenakis.

## Description

Implémentation de la *Génération Dynamique Stochastique* (GENDYN), une approche dynamique stochastique de la synthèse de forme d'onde conçue par Iannis Xenakis.

## Syntaxe

```
ares gendy kamp, kampdist, kdurdist, kadpar, kddpar, kminfreq, kmaxfreq, \  
          kampscl, kdurscl [, initcps] [, knum]  
  
kres gendy kamp, kampdist, kdurdist, kadpar, kddpar, kminfreq, kmaxfreq, \  
          kampscl, kdurscl [, initcps] [, knum]
```

## Initialisation

*initcps* (facultatif, 12 par défaut) -- nombre maximum de points de contrôle.

## Exécution

*kamp* -- amplitude.

*kampdist* -- choix de la distribution de probabilité pour la perturbation d'amplitude suivante d'un point de contrôle. Les distributions valides sont :

- 0 - LINEAIRE
- 1 - CAUCHY
- 2 - LOGISTIQUE
- 3 - COSINUS HYPERBCOLIQUE
- 4 - ARCSINUS
- 5 - EXPONENTIELLE
- 6 - SINUS (signal externe de taux-k)

Si *kampdist*=6, on peut utiliser un signal externe de taux-k via *kadpar*.

*kdurdist* -- choix de la distribution de probabilité pour la perturbation de la durée courante entre points de contrôle. Voir *kampdist* pour les distributions valides. Si *kdurdist*=6, on peut utiliser un signal externe de taux-k via *kddpar*.

*kadpar* -- paramètre pour la distribution *kampdist*. Doit être compris entre 0.0001 et 1.

*kddpar* -- paramètre pour la distribution *kdurdist*. Doit être compris entre 0.0001 et 1.

*kminfreq* -- fréquence d'oscillation minimale autorisée.

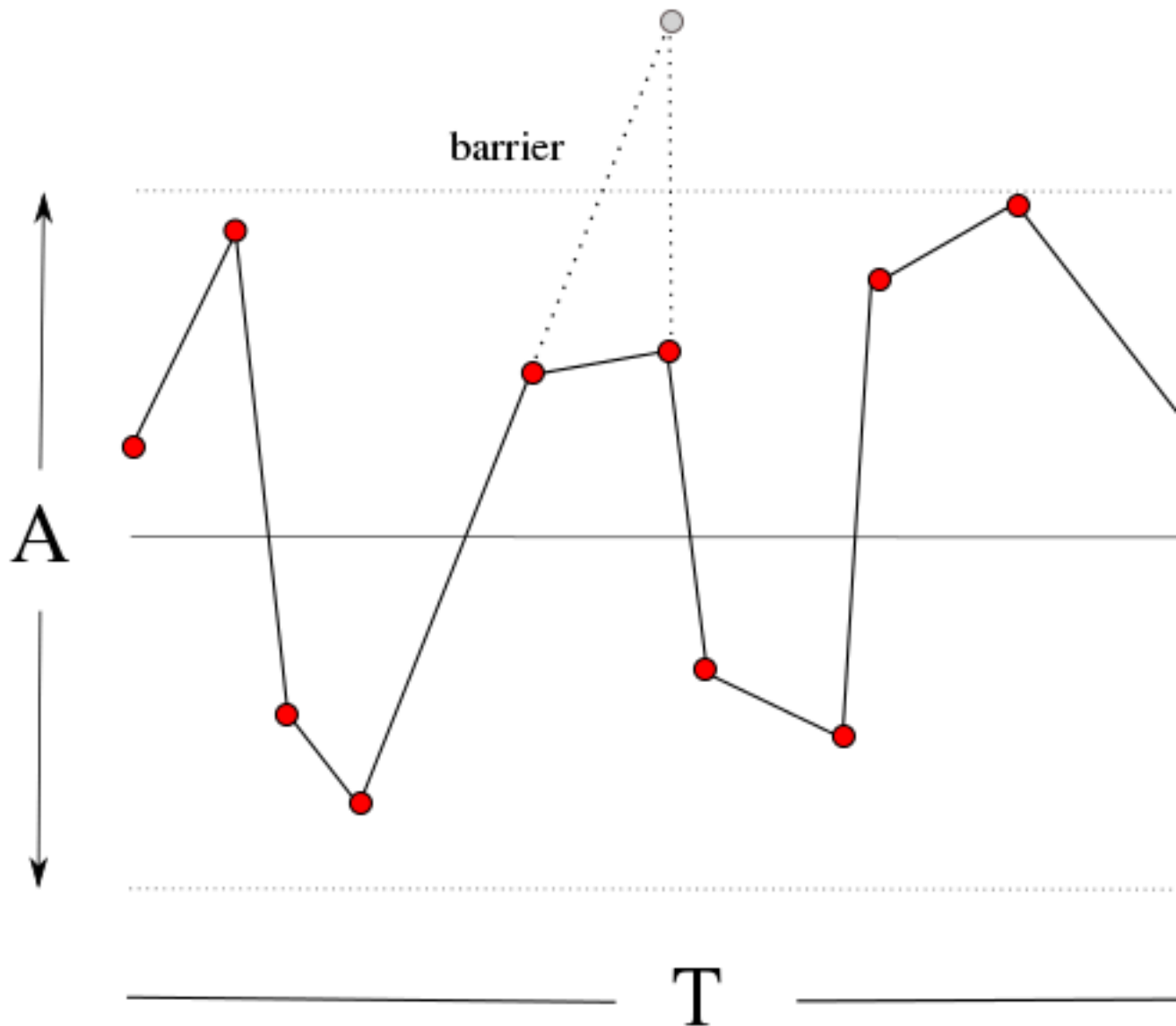
*kmaxfreq* -- fréquence d'oscillation maximale autorisée.

*kampscl* -- multiplicateur pour la valeur du delta de la distribution pour l'amplitude (1.0 pour tout l'intervalle).

*kdurscl* -- multiplicateur pour la valeur du delta de la distribution pour la durée.

*knum* (facultatif, *initcps* par défaut) -- nombre courant de points de contrôle utilisés.

La forme d'onde est générée par *knum* - 1 segments et se répète dans le temps. Les sommets (points de contrôle) bougent par une action stochastique dans les limites de leur réflexion sur un miroir formé par une barrière d'amplitude et une barrière temporelle.



Une répétition de la forme d'onde générée avec *knum*=12.

## Exemples

Voici un exemple de l'opcode *gendy*. Il utilise le fichier *gendy.csd* [examples/gendy.csd].

### Exemple 391. Exemple de l'opcode gendy.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o oscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

aout gendy 0.7, 1, 1, 1, 1, 20, 1000, 0.5, 0.5
outs aout, aout

endin
</CsInstruments>
<CsScore>
i1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

Voici un exemple de l'opcode gendy avec des modulations. Il utilise le fichier *gendy-2.csd* [examples/gendy-2.csd].

### Exemple 392. Exemple de l'opcode gendy avec des modulations.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o oscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kenv expseg 0.01, p3*0.1, 1, p3*0.8, 1, p3*0.1, 0.001
kosc oscil 0.1, 3/p3, 1
seed 20120123
kdis bexprnd kosc
knum linseg 3, p3*0.75, 10, p3*0.20, 12, p3*0.05, 5
```



```
asig gendy 0.2, kosc*60, 6, 0.7, kdis, 500*kenv, 4800, 0.23, 0.3, 12, knum
aflt resonz asig, 1400, 400
aout comb kenv*aflt*0.1, 0.9, 0.1
outs aout, aout

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1 0 .8 0 0 .3 0 0 0 .1

i1 0 20
e
</CsScore>
</CsoundSynthesizer>
```

## Références

1. I. Xenakis. Formalized Music (1992, Stuyvesant, NY: Pendragon Press), pp. 246 - 254, 289 - 322.

## Voir aussi

*gendyc gendyx*

## Crédits

Basé sur le générateur unitaire Gendy1 de Nick Collins (SuperCollider)

Auteur : Tito Latini

Janvier 2012

Nouveau dans la version 5.16 de Csound.

# gendyc

gendyc — Approche dynamique stochastique de la synthèse de forme d'onde avec interpolation cubique.

## Description

Implémentation avec interpolation cubique de la *Génération Dynamique Stochastique* (GENDYN), une approche dynamique stochastique de la synthèse de forme d'onde conçue par Iannis Xenakis.

## Syntaxe

```
ares gendyc kamp, kampdist, kdurdist, kadpar, kddpar, kminfreq, kmaxfreq, \  
      kampscl, kdurscl [, initcps] [, knum]  
  
kres gendyc kamp, kampdist, kdurdist, kadpar, kddpar, kminfreq, kmaxfreq, \  
      kampscl, kdurscl [, initcps] [, knum]
```

## Initialisation

*initcps* (facultatif, 12 par défaut) -- nombre maximum de points de contrôle.

## Exécution

*kamp* -- amplitude.

*kampdist* -- choix de la distribution de probabilité pour la perturbation d'amplitude suivante d'un point de contrôle. Les distributions valides sont :

- 0 - LINEAIRE
- 1 - CAUCHY
- 2 - LOGISTIQUE
- 3 - COSINUS HYPERBOLIQUE
- 4 - ARCSINUS
- 5 - EXPONENTIELLE
- 6 - SINUS (signal externe de taux-k)

Si *kampdist*=6, on peut utiliser un signal externe de taux-k via *kadpar*.

*kdurdist* -- choix de la distribution de probabilité pour la perturbation de la durée courante entre points de contrôle. Voir *kampdist* pour les distributions valides. Si *kdurdist*=6, on peut utiliser un signal externe de taux-k via *kddpar*.

*kadpar* -- paramètre pour la distribution *kampdist*. Doit être compris entre 0.0001 et 1.

*kddpar* -- paramètre pour la distribution *kdurdist*. Doit être compris entre 0.0001 et 1.

*kminfreq* -- fréquence d'oscillation minimale autorisée.

*kmaxfreq* -- fréquence d'oscillation maximale autorisée.

*kampscl* -- multiplicateur pour la valeur du delta de la distribution pour l'amplitude (1.0 pour tout l'intervalle).

*kdurscl* -- multiplicateur pour la valeur du delta de la distribution pour la durée.

*knum* (facultatif, *initcps* par défaut) -- nombre courant de points de contrôle utilisés.

La forme d'onde est générée par *knum* - 1 segments et se répète dans le temps. Les sommets (points de contrôle) bougent par une action stochastique dans les limites de leur réflexion sur un miroir formé par une barrière d'amplitude et une barrière temporelle.

## Exemples

Voici un exemple de l'opcode *gendyc*. Il utilise le fichier *gendyc.csd* [examples/gendyc.csd].

### Exemple 393. Exemple de l'opcode *gendyc*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o oscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

aout gendyc 0.5, 1, 1, 1, 1, 220, 440, 0.5, 0.5
outs aout, aout

endin
</CsInstruments>
<CsScore>
i1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

## Références

1. I. Xenakis. *Formalized Music* (1992, Stuyvesant, NY: Pendragon Press), pp. 246 - 254, 289 - 322.

## Voir aussi

*gendy gendyx*

## Crédits

Basé sur *Gendy1* de Nick Collins et *Gendy4* de Bhub Rainey (SuperCollider)

Auteur : Tito Latini  
Janvier 2012

Nouveau dans la version 5.16 de Csound.

# gendyx

gendyx — Variation de l'approche dynamique stochastique de la synthèse de forme d'onde conçue par Iannis Xenakis.

## Description

*gendyx* (gendy eXtended) est une implémentation de la *Génération Dynamique Stochastique* (GENDYN), une approche dynamique stochastique de la synthèse de forme d'onde conçue par Iannis Xenakis, en utilisant des courbes au lieu de segments.

## Syntaxe

```
ares gendyx kamp, kampdist, kdurdist, kadpar, kddpar, kminfreq, kmaxfreq, \  
          kampscl, kdurscl, kcurveup, kcurvedown [, initcps] [, knum]  
  
kres gendyx kamp, kampdist, kdurdist, kadpar, kddpar, kminfreq, kmaxfreq, \  
          kampscl, kdurscl, kcurveup, kcurvedown [, initcps] [, knum]
```

## Initialisation

*initcps* (facultatif, 12 par défaut) -- nombre maximum de points de contrôle.

## Exécution

*kamp* -- amplitude.

*kampdist* -- choix de la distribution de probabilité pour la perturbation d'amplitude suivante d'un point de contrôle. Les distributions valides sont :

- 0 - LINEAIRE
- 1 - CAUCHY
- 2 - LOGISTIQUE
- 3 - COSINUS HYPERBCOLIQUE
- 4 - ARCSINUS
- 5 - EXPONENTIELLE
- 6 - SINUS (signal externe de taux-k)

Si *kampdist*=6, on peut utiliser un signal externe de taux-k via *kadpar*.

*kdurdist* -- choix de la distribution de probabilité pour la perturbation de la durée courante entre points de contrôle. Voir *kampdist* pour les distributions valides. Si *kdurdist*=6, on peut utiliser un signal externe de taux-k via *kddpar*.

*kadpar* -- paramètre pour la distribution *kampdist*. Doit être compris entre 0.0001 et 1.

*kddpar* -- paramètre pour la distribution *kdurdist*. Doit être compris entre 0.0001 et 1.

*kminfreq* -- fréquence d'oscillation minimale autorisée.

*kmaxfreq* -- fréquence d'oscillation maximale autorisée.

*kampscl* -- multiplicateur pour la valeur du delta de la distribution pour l'amplitude (1.0 pour tout l'intervalle).

*kdurscl* -- multiplicateur pour la valeur du delta de la distribution pour la durée.

*kcurveup* -- contrôle la courbe de croissance des amplitudes entre deux points ; doit être non négatif.

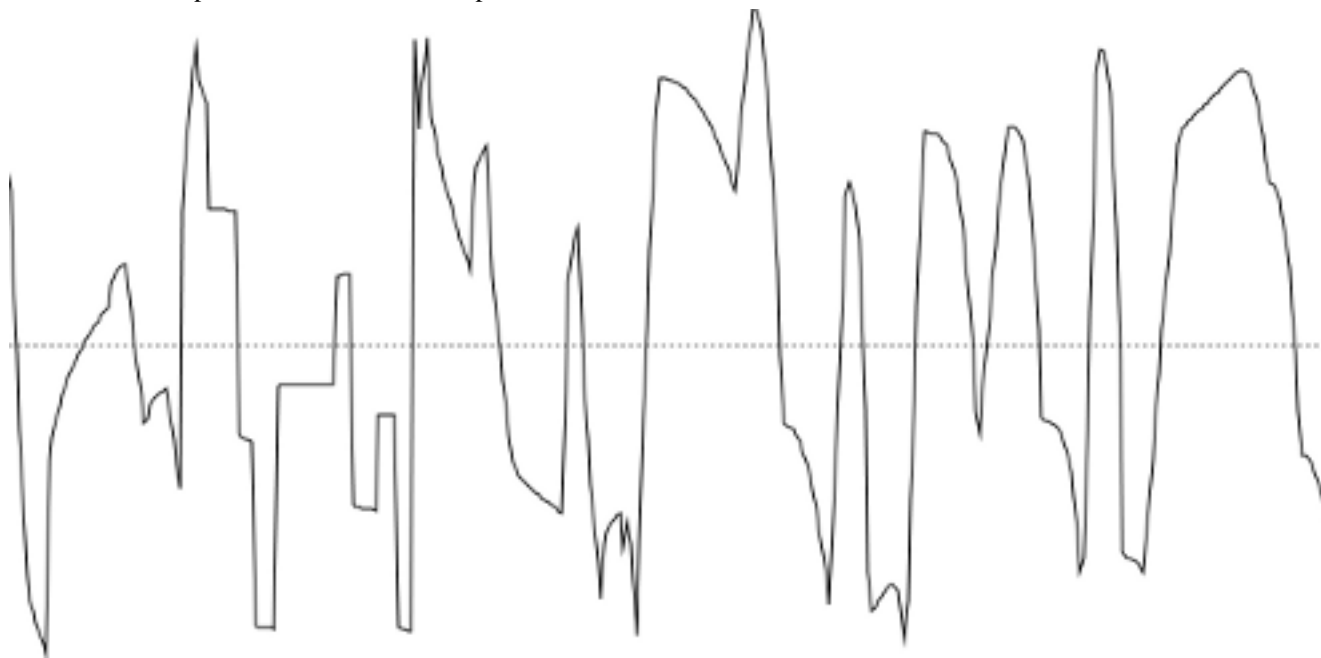
- 0 : fonction en escalier (comme dans échantillonneur-bloqueur)
- <1 : concave
- 1 : linéaire (comme *gendy*)
- >1 : convexe

*kcurvedown* -- contrôle la courbe de décroissance des amplitudes entre deux points ; doit être non négatif.

- 0 : fonction en escalier
- <1 : convexe
- 1 : linéaire
- >1 : concave

*knum* (facultatif, *initcps* par défaut) -- nombre courant de points de contrôle utilisés.

La forme d'onde est générée par *knum* - 1 segments et se répète dans le temps. Les sommets (points de contrôle) bougent par une action stochastique dans les limites de leur réflexion sur un miroir formé par une barrière d'amplitude et une barrière temporelle.



Extrait d'une forme d'onde générée avec *gendyx*.

## Exemples

Voici un exemple de l'opcode gendyx. Il utilise le fichier *gendyx.csd* [examples/gendyx.csd].

### Exemple 394. Example of the gendyx opcode.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o oscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

aout gendyx 0.7, 1, 1, 1, 1, 20, 1000, 0.5, 0.5, 4, 0.13
outs aout, aout

endin
</CsInstruments>
<CsScore>
i1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

Voici un exemple de l'opcode gendyx avec des modulations. Il utilise le fichier *gendyx-2.csd* [examples/gendyx-2.csd].

### Exemple 395. Exemple de l'opcode gendyx avec des modulations.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o oscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
```

```
kenv expseg 0.001, p3*0.05, 0.5, p3*0.9, 0.5, p3*0.05, 0.001
kc1 linseg 1, p3/2, 12, p3/2, 3
kc2 random 0, 4
seed 20120124
asig gendyx kenv, 1, 3, 0.7, 0.8, 120, 4300, 0.2, 0.7, kc1, kc2, 12, kc1
aout dcblock asig
outs aout, aout
endin
</CsInstruments>
<CsScore>
i1 0 20
e
</CsScore>
</CsoundSynthesizer>
```

## Références

1. I. Xenakis. Formalized Music (1992, Stuyvesant, NY: Pendragon Press), pp. 246 - 254, 289 - 322.

## Voir aussi

*gendy gendyc*

## Crédits

Variation du générateur unitaire Gendy1 de Nick Collins (SuperCollider)

Auteur : Tito Latini

Janvier 2012

Nouveau dans la version 5.16 de Csound.



# getcfg

getcfg — Retourne les réglages de Csound.

## Description

Retourne différents réglages de configuration dans *Svalue* sous forme de chaîne de caractères, pendant l'initialisation.

## Syntaxe

*Svalue* **getcfg** *iopt*

## Initialisation

*iopt* -- le paramètre à retourner ; peut être un de ceux-ci :

- 1 : la longueur maximale des variables chaîne, en caractères ; vaut au moins la valeur de l'option -+max\_str\_len de la ligne de commande - 1.



### Note

Dans Csound6 il n'y a pas de longueur maximale pour les chaînes de caractères, ce qui ôte toute signification à la valeur retournée.

- 2 : le nom du fichier son en entrée (-i), ou une chaîne vide s'il n'y a pas de fichier en entrée
- 3 : le nom du fichier son en sortie (-o), ou une chaîne vide s'il n'y a pas de fichier en sortie
- 4 : retourne "1" si une entrée ou une sortie audio en temps réel est utilisée, "0" sinon
- 5 : retourne "1" si l'exécution est en mode pulsation (option -t de la ligne de commande), "0" sinon
- 6 : le nom du système d'exploitation hôte
- 7 : retourne "1" si une fonction de rappel a été installée pour les opcodes *chnrecv* et *chnsend*, "0" sinon (ce qui veut dire que ces opcodes ne font rien)

## Exemples

Voici un exemple de l'opcode getcfg. Il utilise le fichier *getcfg.csd* [examples/getcfg.csd].

### Exemple 396. Exemple de l'opcode getcfg.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
-iadc      ;;uncomment -iadc if realtime audio input is needed too
</CsOptions>
```

```

<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
odbfs = 1

instr 1

S1 getcfg 1 ; --+max_str_len
S2 getcfg 2 ; -i
S3 getcfg 3 ; -o
S4 getcfg 4 ; RTaudio
S5 getcfg 5 ; -t
S6 getcfg 6 ; os system host
S7 getcfg 7 ; callback

prints "-----"
prints "\nMax string len : "
prints S1
prints "\nInput file name (-i) : "
prints S2
prints "\nOutput file name (-o) : "
prints S3
prints "\nRTaudio (-odac) : "
prints S4
prints "\nBeat mode (-t)? : "
prints S5
prints "\nHost Op. Sys. : "
prints S6
prints "\nCallback ? : "
prints S7
prints "\n"
prints "-----\n"

endin

</CsInstruments>
<CsScore>
i 1 0 0
e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie comprendra des lignes comme celles-ci :

```

-----
Max string len : 255
Input file name (-i) : adc
Output file name (-o) : dac
RTaudio (-odac) : 1
Beat mode (-t)? : 0
Host Op. Sys. : Linux
Callback ? : 0
-----

```

## Crédits

Auteur : Istvan Varga  
2006

Nouveau dans la version 5.02

# getcol

getcol — Retourne une colonne donnée d'un tableau bidimensionnel comme un vecteur.

## Description

Retourne une colonne donnée d'un tableau bidimensionnel. La sortie est un tableau unidimensionnel contenant la colonne requise.

## Syntaxe

```
i/kout[] getcoli/kin[], i/kcol
```

## Initialisation

*iout[]* -- tableau de sortie contenant la colonne extraite. Créé s'il n'existe pas.

*iin[]* -- tableau bidimensionnel en entrée.

*icol* -- colonne à extraire.

## Exécution

*kout[]* -- tableau de sortie contenant la colonne extraite. Créé s'il n'existe pas.

*kin[]* -- tableau bidimensionnel en entrée.

*kcol* -- colonne à extraire.

## Exemples

Voici un exemple de l'opcode getcol. Il utilise le fichier *getcol.csd* [examples/getcol.csd].

### Exemple 397. Exemple de l'opcode getcol.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
instr 1
  kcnt init 0
  kArr[] init 3,3
  kArr[] fillarray 0,1,2,0,1,2,0,1,2
  while kcnt < 3 do
    kVec[] getcol kArr,kcnt
    printf "column %d: %d %d %d\n",kcnt+1,kcnt,kVec[0],kVec[1],kVec[2]
    kcnt += 1
  od
endin
</CsInstruments>
```

```
<CsScore>  
i1 0 0.1  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux.

## Crédits

Auteur : Victor Lazzarini  
NUI Maynooth  
2014

Nouveau dans la version 6.04

# getftargs

getftargs — Remplit une variable chaîne de caractères avec les arguments donnés à la création d'une table de fonction au taux-k.

## Description

Opcode du greffon getftargs.

*getftargs* écrit les arguments donnés à la création d'une table de fonction dans une variable chaîne de caractères. *getftargs* fonctionne durant l'initialisation ainsi que pendant l'exécution.

## Syntaxe

Sdst **getftargs** iftno, ktrig

## Initialisation

*iftno* -- Numéro de la table dont les arguments vont être utilisés.

## Exécution

*Sdst* -- variable chaîne de caractères en sortie.

*ktrig* -- signal de déclenchement, doit être valide au temps-i. La variable chaîne de caractères en sortie est remplie à l'initialisation si *ktrig* est positif, et pendant l'exécution, chaque fois que *ktrig* est positif et différent de la valeur précédente. Utiliser la constante 1 pour n'imprimer qu'une fois à l'initialisation de la note.

## Exemples

Voici un exemple de l'opcode getftargs. Il utilise le fichier *getftargs.csd* [examples/getftargs.csd].

### Exemple 398. Exemple de l'opcode getftargs.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o getftargs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
```

```
instr 1
Sargs getftargs 1, 1
puts Sargs, 1
endin

</CsInstruments>
<CsScore>
f 1 0 1024 "quadbezier" 0 0 0.5 200 0.8 450 0.33 600 0.1 800 0.4 1024 0
i 1 0 1
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie devrait contenir cette ligne :

```
0 0 0.5 200 0.8 450 0.33 600 0.1 800 0.4 1024 0
```

## Voir aussi

*ftgen*, *ftlen* et *sprintf*

## Crédits

Ecrit par Guillermo Senna

2016

# getrow

getrow — Retourne une ligne donnée d'un tableau bidimensionnel comme un vecteur.

## Description

Retourne une ligne donnée d'un tableau bidimensionnel. La sortie est un tableau unidimensionnel contenant la ligne requise.

## Syntaxe

```
i/kout[] getrowi/kin[],i/krow
```

## Initialisation

*iout[]* -- tableau de sortie contenant la ligne extraite. Créé s'il n'existe pas.

*iin[]* -- tableau bidimensionnel en entrée.

*irow* -- ligne à extraire.

## Exécution

*kout[]* -- tableau de sortie contenant la ligne extraite. Créé s'il n'existe pas.

*kin[]* -- tableau bidimensionnel en entrée.

*krow* -- ligne à extraire.

## Exemples

Voici un exemple de l'opcode getrow. Il utilise le fichier *rfft.csd* [examples/rfft.csd].

### Exemple 399. Exemple de l'opcode getrow.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>

<CsOptions>
-d -o dac
</CsOptions>

<CsInstruments>
;ksmps needs to be an integer div of hopsize
ksmps = 64
0dbfs=1

instr 1

    ihopsize = 256    ; hopsize
    ifftsize = 1024  ; FFT size
```

```

iolaps = ifftsize/ihopsize ; overlaps
ibw = sr/ifftsize ; bin bandwidth
kcnt init 0 ; counting vars
krow init 0

kOla[] init ifftsize ; overlap-add buffer
kIn[] init ifftsize ; input buffer
kSw[] init ifftsize
kOut[][] init iolaps, ifftsize ; output buffers

a1 diskin2 "fox.wav",1,0,1 ; audio input
ks expon 100, p3, 1000
asw vco2 0.15, ks

/* every hopsize samples */
if kcmt == ihopsize then
  /* window and take FFT */
  kWin[] window kIn,krow*ihopsize
  kSpec[] rfft kWin
  kWin window kSw,krow*ihopsize
  kSpec2[] rfft kWin
  kProd[] cmplxprod kSpec, kSpec2

  /* IFFT + window */
  kRow[] rfft kProd + kSpec
  kWin window kRow, krow*ihopsize
  /* place it on out buffer */
  kOut setrow kWin, krow

  /* zero the ola buffer */
  kOla = 0
  /* overlap-add */
  ki = 0
  until ki == iolaps do
    kRow getrow kOut, ki
    kOla = kOla + kRow
    ki += 1
  od

  /* update counters */
  krow = (krow+1)%iolaps
  kcmt = 0
endif

/* shift audio in/out of buffers */
kIn shiftin a1
kSw shiftin asw
a2 shiftout kOla
out a2/iolaps

/* increment counter */
kcmt += ksmpts

endin

</CsInstruments>

<CsScore>
i1 0 10
</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

Vectorial opcodes, Opcodes de tableaux.



## Crédits

Auteur : Victor Lazzarini  
NUI Maynooth  
2014

Nouveau dans la version 6.04

# getrowlin

getrowlin — Copie une ligne d'un tableau 2D dans un ftable avec interpolation entre les lignes.

## Description

Opcode du greffon beosc.

Etant donné un tableau 2D (tableau-i ou -k), ou une ftable représentant une matrice 2D, prend une ligne de cette matrice (ça peut être un bout de ligne). Si *krow* n'est pas un entier, les valeurs sont interpolées entre les deux lignes adjacentes. En supposant qu'une telle matrice 2D contient plusieurs lignes de flots échantillonnés (par exemple les amplitudes d'un ensemble d'oscillateurs échantillonnées à intervalles réguliers), cet opcode extrait une ligne de ces données avec interpolation linéaire entre lignes adjacentes (si la ligne n'est pas un nombre rond) et place le résultat dans un tableau 1D.

## Syntaxe

```
kOut[] getrowlin kMtx[], krow [, kstart, kend, kstep ]  
kOut[] getrowlin krow, ifn, inumcols [, iskip, start, iend, istep ]
```

## Initialisation / Exécution

*kMtrx[]* -- Un tableau 2D.

*krow* -- La ligne à lire (peut être un nombre fractionnaire, auquel cas il y aura interpolation avec la ligne suivante).

*kstart* -- Position à partir de laquelle commence la lecture dans la ligne (0 par défaut).

*kend* -- Position (non inclusive) à laquelle finit la lecture dans la ligne.

*kstep* -- Pas de lecture dans la ligne (1 par défaut).

*iskip* -- Si une ftable est utilisée en entrée, *iskip* indique le début des données échantillonnées (ignorant un possible en-tête dans les données). Vaut 0 par défaut.

*inumcols* -- Si une ftable est utilisée en entrée, *inumcols* indique le nombre de colonnes de la matrice 2D.

## Exemples

Voici un exemple de l'opcode getrowlin. Il utilise le fichier *getrowlin.csd* [examples/getrowlin.csd]. See also the example for beadsynt: *beadsynt.csd* [examples/beadsynt.csd]

### Exemple 400. Exemple de l'opcode getrowlin.

```
<CsoundSynthesizer>  
<CsOptions>  
-odac      ;;;realtime audio out  
</CsOptions>  
<CsInstruments>  
  
/*  
  
This is the example file for getrowlin
```

*getrowlin*  
 =====

Given a 2D array (i- or k- array), or a table representing a 2D matrix, get a row of this matrix (possibly a slice). If krow is not an integer, the values are the result of the interpolation between the two adjacent rows.

Assuming such a 2D matrix containing multiple rows of sampled streams (for instance, the amplitudes of a set of oscillators, sampled at a regular interval), this opcode extracts one row of that data with linear interpolation between adjacent rows and places the result in a 1D array

NB: The destination array (the left hand term) does not need to be previously initialized

NB2: if the destination array is too small to fit the data, it will be enlarged

*Syntax*  
 =====

kOut[] *getrowlin* kMtrx[], krow, kstart=0, kend=0, kstep=1  
 kOut[] *getrowlin* krow, ifn, inumcols, iskip=0, istart=0, iend=0, istep=1

kMtrx[] : a 2D array  
 krow : the row to read (can be a fractional number, in which case interpolation with the next row is performed)  
 kstart : start index to read from the row  
 kend : end index to read from the row (not inclusive)  
 kstep : step used to read the along the row  
 iskip : in the case of using a table as input, iskip indicates the start of the sampled data (skipping a possible header in the data)  
 inumcols : in the case of using a table as input, inumcols indicates the number of columns of the 2D matrix.

\*/

**sr** = 44100  
**ksmps** = 128  
**nchnls** = 1  
**0dbfs** = 1

; just a simple test of the bare functionality  
 instr 1

; make a 4x3 array  
 kMtx[] **init** 3, 4  
 kMtx **fillarray** 0, 1, 2, 3, \  
                   10, 11, 12, 13, \  
                   20, 21, 22, 23

krow **linseg** 0, **p3**, 2  
**printk2** krow, 20  
 kOut[] **getrowlin** kMtx, krow  
**printarray** kOut, -1, "", "kOut"

; the same, but use cosine interpolation  
 krow0 = **floor**(krow)  
 krow1 = krow0 + 1  
 krowcos = **lincos**(krow, krow0, krow1, krow0, krow1)  
 kOut2[] **getrowlin** kMtx, krowcos  
**printarray** kOut2, -1, "", "kOut2"

endin

; simplified example taken from beadsynt, to show a practical use case  
 ; this uses the file fox.mtx.wav, which is not a real soundfile, but

```
; a 2D matrix saved as a soundfile, generated with loristrck_pack
; (see https://github.com/gesellkammer/loristrck)
instr 2
  ifn ftgentmp 0,0,0,-1, "fox.mtx.wav", 0, 0, 0
  ; ifn is organized as follow:
  ; header: iskip, idt, inumcols, inumrows, itimestart
  ; multiple rows of the form f0, a0, bw0, f1, a1, bw1, ...
  iskip      tab_i 0, ifn
  idt        tab_i 1, ifn
  inumcols   tab_i 2, ifn
  inumrows   tab_i 3, ifn
  itimestart tab_i 4, ifn
  inumpartials = inumcols / 3
  imaxrow = inumrows - 2
  idur = inumrows * idt
  kfreqscale cosseg 1, idur, 2
  kplayhead phasor (1/idur)*idur
  krow = limit(kplayhead / idt, 0, imaxrow)
  ;
  ; start end step
  kFreqs[] getrowlin krow, ifn, inumcols, iskip, 0, 0, 3
  kAmps[] getrowlin krow, ifn, inumcols, iskip, 1, 0, 3
  kBws[] getrowlin krow, ifn, inumcols, iskip, 2, 0, 3
  aout beadsynt kFreqs, kAmps, kBws, -1, 0, kfreqscale
  outch 1, aout
  if(timeinstds() > idur) then
    event "e", 0, 0, 0
  endif
endin

</CsInstruments>
<CsScore>
i 1 0 2
i 2 2 3600
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*tabrowlin, beadsynt, getrow, slicearray, copyf2array, tab2array, ftslice,*

## Crédits

Auteur : Eduardo Moguillansky 2018

Nouveau greffon dans la version 6.12

# getseed

getseed — Lit la valeur globale de la graine.

## Description

Retourne la valeur globale de la graine utilisée pour tous les *générateurs de bruit de classe x*.

## Syntaxe

`ians getseed`

`kans getseed`

## Exécution

Lit la valeur globale de la graine du générateur interne de nombres pseudo aléatoires.

## Crédits

Auteur : John ffitch  
Université de Bath/NUIM/Codemist Ltd.  
Bath, UK  
Août 2015

Nouveau dans Csound6.06

# gogobel

*gogobel* — La sortie audio est un son tel que celui produit lorsque l'on frappe une cloche à vache.

## Description

La sortie audio est un son tel que celui produit lorsque l'on frappe une cloche à vache. Il s'agit d'un modèle physique développé par Perry Cook, mais recodé pour Csound.

## Syntaxe

```
ares gogobel kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivfn
```

## Initialisation

*ihrd* -- la dureté de la baguette utilisée pour la frappe. On utilise un intervalle allant de 0 à 1. 0,5 est une valeur adéquate.

*ipos* -- le point d'impact sur le bloc, compris entre 0 et 1.

*imp* -- une table des impulsions de la frappe. Le fichier *marmstk1.wav* [examples/marmstk1.wav] contient une fonction adéquate créée à partir de mesures et l'on peut le charger dans une table *GEN01*. Il est aussi disponible à <ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/>.

*ivfn* -- forme du vibrato, habituellement une table sinus, créée par une fonction.

## Exécution

Une note est jouée sur une instrument de type cloche à vache, avec les arguments suivants.

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note.

*kvibf* -- Fréquence du vibrato en Hertz. L'intervalle conseillé va de 0 à 12.

*kvamp* -- Amplitude du vibrato.

## Exemples

Voici un exemple de l'opcode *gogobel*. Il utilise les fichiers *gogobel.csd* [examples/gogobel.csd] et *marmstk1.wav* [examples/marmstk1.wav]

### Exemple 401. Exemple de l'opcode *gogobel*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform
```

```

-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gogobel.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

    kfreq = 200
    ihrd = 0.5
    ipos = p4
    kvibf = 6.0
    kvamp = 0.3

    asig gogobel .9, kfreq, ihrd, ipos, 1, 6.0, 0.3, 2
        outs asig, asig
    endin
</CsInstruments>
<CsScore>
;audio file
f 1 0 256 1 "marmstkl.wav" 0 0 0
;sine wave for the vibrato
f 2 0 128 10 1

i 1 0.5 0.5 0.01
i 1 + 0.5 0.561
i 1 + 0.5 0.9
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 3.47 de Csound

# goto

goto — Transfère le contrôle à chaque passage.

## Description

Transfère le contrôle vers *label* à chaque passage. Combinaison de *igoto* et de *kgoto*

## Syntaxe

`goto label`

où *label* se trouve dans le même bloc d'instrument et n'est pas une expression.



### Note

Si l'on utilise *goto* en dehors d'une instruction *if* (comme dans : goto end), l'initialisation sera ignorée pour tout le code sauté par le *goto*. Si dans une exécution quelques opcodes ne sont pas initialisés, cela provoquera l'effacement de la note ou de l'évènement. Dans ce cas, il peut être préférable d'utiliser *kgoto* (comme dans : kgoto end).

## Exemples

Voici un exemple de l'opcode goto. Il utilise le fichier *goto.csd* [examples/goto.csd].

### Exemple 402. Exemple de l'opcode goto.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o goto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  a1 oscil 10000, 440, 1
  goto playit

  ; The goto will go to the playit label.
  ; It will skip any code in between like this comment.

playit:
  out a1
endin
```



```
</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*cggoto, cigoto, ckgoto, if, igoto, kgoto, tigoto, timeout*

## Crédits

Exemple écrit par Kevin Conder.

Note ajoutée par Jim Aikin.

# grain

grain — Génère des textures de synthèse granulaire.

## Description

Génère des textures de synthèse granulaire.

## Syntaxe

```
ares grain xamp, xpitch, xdens, kampoff, kpitchoff, kgdur, igfn, \  
      iwfn, imgdur [, igrnd]
```

## Initialisation

*igfn* -- numéro de la ftable de la forme d'onde du grain. Peut être une onde sinus ou un son échantillonné.

*iwfn* -- numéro de la ftable de l'enveloppe d'amplitude utilisée pour les grains (voir aussi *GEN20*).

*imgdur* -- durée maximum du grain en secondes. C'est la plus grande valeur que l'on peut affecter à *kgdur*.

*igrnd* (facultatif) -- s'il est non nul, le décalage aléatoire du grain est désactivé. Cela signifie que tous les grains commenceront à lire la table *igfn* depuis son début. S'il vaut zéro (par défaut), les grains commenceront leur lecture dans la table *igfn* à partir de positions aléatoires.

## Exécution

*xamp* -- amplitude de chaque grain.

*xpitch* -- hauteur du grain. Pour utiliser la fréquence originale du son en entrée, on se sert de la formule :

$\text{sndsr} / \text{ftlen}(\text{igfn})$

où *sndsr* est le taux d'échantillonnage original du son *igfn*.

*xdens* -- densité des grains mesurée en grains par seconde. Si elle est constante la sortie sera une synthèse granulaire synchrone, très semblable à *fof*. Si *xdens* a une composante aléatoire (comme du bruit ajouté), alors le résultat ressemblera plus à une synthèse granulaire asynchrone.

*kampoff* -- déviation d'amplitude maximale par rapport à *xamp*. Cela signifie que l'amplitude maximale possible pour un grain est *xamp* + *kampoff* et l'amplitude minimale est *xamp*. Si *kampoff* est nul alors il n'y a pas d'amplitude aléatoire pour chaque grain.

*kpitchoff* -- déviation de hauteur maximale par rapport à *xpitch* en Hz. Semblable à *kampoff*.

*kgdur* -- durée du grain en secondes. Sa valeur maximale doit être déclarée dans *imgdur*. Si *kgdur* dépasse *imgdur* en un point, sa valeur sera tronquée à celle de *imgdur*.

Le générateur *grain* est principalement basé sur les travaux et les écrits de Barry Truax et de Curtis Roads.

## Exemples

Cet exemple génère une texture avec des grains de plus en plus courts, une amplitude de plus en plus large et une dispersion de hauteur. Il utilise les fichiers *grain.csd* [examples/grain.csd] et *beats.wav* [examples/beats.wav].

### Exemple 403. Exemple de l'opcode grain.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o grain.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 32
nchnls  = 2
0dbfs   = 1

instr 1

insnd    = 10
ibasfrq  = 44100 / ftlen(insnd) ; Use original sample rate of insnd file

kamp     expseg .001, p3/2, .1, p3/2, .01 ;a swell in amplitude
kpitch   line ibasfrq, p3, ibasfrq * .8
kdens    line 600, p3, 100
kaoff    line 0, p3, .1
kpoff    line 0, p3, ibasfrq * .5
kgdur    line .4, p3, .01
imaxgdur = .5

asigL    grain kamp, kpitch, kdens, kaoff, kpoff, kgdur, insnd, 5, imaxgdur, 0.0
asigR    grain kamp, kpitch, kdens, kaoff, kpoff, kgdur, insnd, 5, imaxgdur, 0.0
outs     asigL, asigR

endin
</CsInstruments>
<CsScore>
f5 0 512 20 2 ; Hanning window
f10 0 16384 1 "beats.wav" 0 0 0

i1 0 15
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Paris Smaragdis  
MIT  
Mai 1997

# grain2

grain2 — Générateur de textures par synthèse granulaire facile à utiliser.

## Description

Génère des textures par synthèse granulaire. *grain2* est plus simple à utiliser, mais *grain3* offre plus de contrôle.

## Syntaxe

```
ares grain2 kcps, kfmd, kgdur, iovrlp, kfn, iwfn [, irpow] \  
      [, iseed] [, imode]
```

## Initialisation

*iovrlp* -- (constant) nombre de grains se chevauchant.

*iwfn* -- table de fonction contenant la forme d'onde d'une fenêtre (utiliser GEN20 pour calculer *iwfn*).

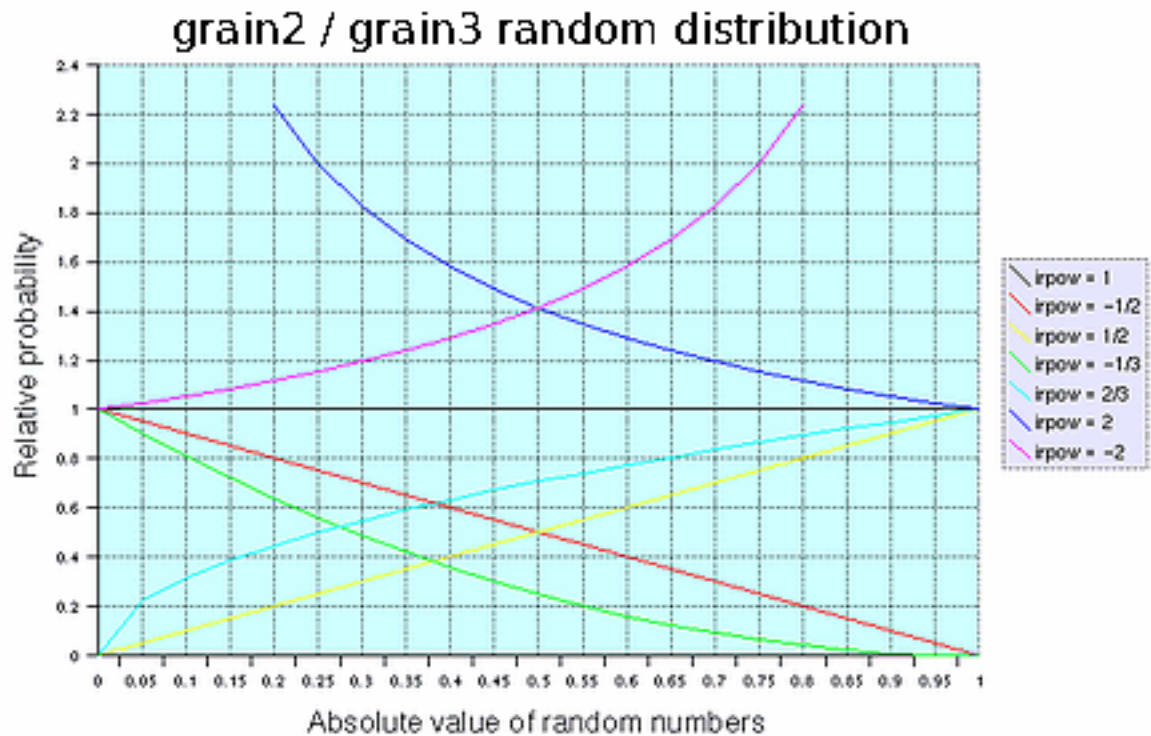
*irpow* (facultatif, par défaut 0) -- cette valeur contrôle la variation de la distribution de la fréquence du grain. Si *irpow* est positif, la distribution aléatoire ( $x$  est compris entre -1 et 1) est

$\text{abs}(x) ^ ((1 / \text{irpow}) - 1) ;$

pour des valeurs négatives de *irpow*, elle est

$(1 - \text{abs}(x)) ^ ((-1 / \text{irpow}) - 1)$

En fixant *irpow* à -1, 0, ou 1 on obtiendra une distribution uniforme (dont le calcul est plus rapide). L'image ci-dessous montre quelques exemples pour *irpow*. La valeur par défaut de *irpow* est 0.



Un graphique des distributions pour différentes valeurs de *irpow*.

*iseed* (facultatif, par défaut 0) -- valeur de la graine du générateur de nombres aléatoires (entier positif compris entre 1 et 2147483646 ( $2^{31} - 2$ )). Une valeur nulle ou négative force la graine à prendre la valeur de l'horloge de l'ordinateur (c'est le comportement par défaut).

*imode* (facultatif, par défaut 0) -- somme de valeurs prises parmi les suivantes :

- 8 : forme d'onde de la fenêtre avec interpolation (plus lent).
- 4 : pas d'interpolation pour la forme d'onde des grains (rapide, mais de moindre qualité).
- 2 : la fréquence des grains est modifiée continuellement par *kcps* et *kfmd* (par défaut, chaque grain garde la fréquence avec laquelle il a démarré). Avec des taux de contrôle élevés, ceci peut ralentir le processus.
- 1 : ignorer l'initialisation.

## Exécution

*ares* -- signal de sortie.

*kcps* -- fréquence du grain en Hz.

*kfmd* -- variation aléatoire (bipolaire) de la fréquence du grain en Hz.

*kgdur* -- durée du grain en secondes. *kgdur* contrôle aussi la durée des grains déjà actifs (en fait la vitesse à laquelle la fonction fenêtre est lue). Ce comportement ne dépend pas des indicateurs positionnés dans *imode*.

*kfn* -- table de fonction contenant la forme d'onde du grain. Le numéro de table peut changer au taux-k (on peut ainsi choisir parmi un ensemble de tables à bande limitée générées par GEN30, afin d'éviter le repliement).



## Note

*grain2* utilise en interne le même générateur aléatoire que *rnd31*. Il est ainsi recommandé de lire également sa *documentation*.

## Exemples

Voici un exemple de l'opcode *grain2*. Il utilise le fichier *grain2.csd* [examples/grain2.csd].

### Exemple 404. Exemple de l'opcode *grain2*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc    -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o grain2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 48000
kr = 750
ksmps = 64
nchnls = 2

/* square wave */
i_ ftgen 1, 0, 4096, 7, 1, 2048, 1, 0, -1, 2048, -1
/* window */
i_ ftgen 2, 0, 16384, 7, 0, 4096, 1, 4096, 0.3333, 8192, 0
/* sine wave */
i_ ftgen 3, 0, 1024, 10, 1
/* room parameters */
i_ ftgen 7, 0, 64, -2, 4, 50, -1, -1, -1, 11, \
    1, 26.833, 0.05, 0.85, 10000, 0.8, 0.5, 2, \
    1, 1.753, 0.05, 0.85, 5000, 0.8, 0.5, 2, \
    1, 39.451, 0.05, 0.85, 7000, 0.8, 0.5, 2, \
    1, 33.503, 0.05, 0.85, 7000, 0.8, 0.5, 2, \
    1, 36.151, 0.05, 0.85, 7000, 0.8, 0.5, 2, \
    1, 29.633, 0.05, 0.85, 7000, 0.8, 0.5, 2

ga01 init 0

/* generate bandlimited square waves */

i0 = 0
loop1:
imaxh = sr / (2 * 440.0 * exp(log(2.0) * (i0 - 69) / 12))
i_ ftgen i0 + 256, 0, 4096, -30, 1, 1, imaxh
i0 = i0 + 1
if (i0 < 127.5) igoto loop1

instr 1
```

```

p3 = p3 + 0.2

/* note velocity */
iamp = 0.0039 + p5 * p5 / 16192
/* vibrato */
kcps oscili 1, 8, 3
kenv linseg 0, 0.05, 0, 0.1, 1, 1, 1
/* frequency */
kcps = (kcps * kenv * 0.01 + 1) * 440 * exp(log(2) * (p4 - 69) / 12)
/* grain ftable */
kfn = int(256 + 69 + 0.5 + 12 * log(kcps / 440) / log(2))
/* grain duration */
kgdur port 100, 0.1, 20
kgdur = kgdur / kcps

a1 grain2 kcps, kcps * 0.02, kgdur, 50, kfn, 2, -0.5, 22, 2
a1 butterlp a1, 3000
a2 grain2 kcps, kcps * 0.02, 4 / kcps, 50, kfn, 2, -0.5, 23, 2
a2 butterbp a2, 12000, 8000
a2 butterbp a2, 12000, 8000
aenv1 linseg 0, 0.01, 1, 1, 1
aenv2 linseg 3, 0.05, 1, 1, 1
aenv3 linseg 1, p3 - 0.2, 1, 0.07, 0, 1, 0

a1 = aenv1 * aenv3 * (a1 + a2 * 0.7 * aenv2)

ga01 = ga01 + a1 * 10000 * iamp

endin

/* output instr */

instr 81

i1 = 0.000001
aLl, aLh, aRl, aRh spat3di ga01 + i1*i1*i1*i1, 3.0, 4.0, 0.0, 0.5, 7, 4
ga01 = 0
aLl butterlp aLl, 800.0
aRl butterlp aRl, 800.0

outs aLl + aLh, aRl + aRh

endin

</CsInstruments>
<CsScore>

t 0 60

i 1 0.0 1.3 60 127
i 1 2.0 1.3 67 127
i 1 4.0 1.3 64 112
i 1 4.0 1.3 72 112

i 81 0 6.4

e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*grain3*

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.15

Mise à jour en avril 2002 par Istvan Varga



# grain3

grain3 — Générateur de textures par synthèse granulaire avec plus de contrôle.

## Description

Génère des textures par synthèse granulaire. *grain2* est plus simple à utiliser mais *grain3* offre plus de contrôle.

## Syntaxe

```
ares grain3 kcps, kphs, kfmd, kpmd, kgdur, kdens, imaxovr, kfn, iwfn, \  
      kfrpow, krpow [, iseed] [, imode]
```

## Initialisation

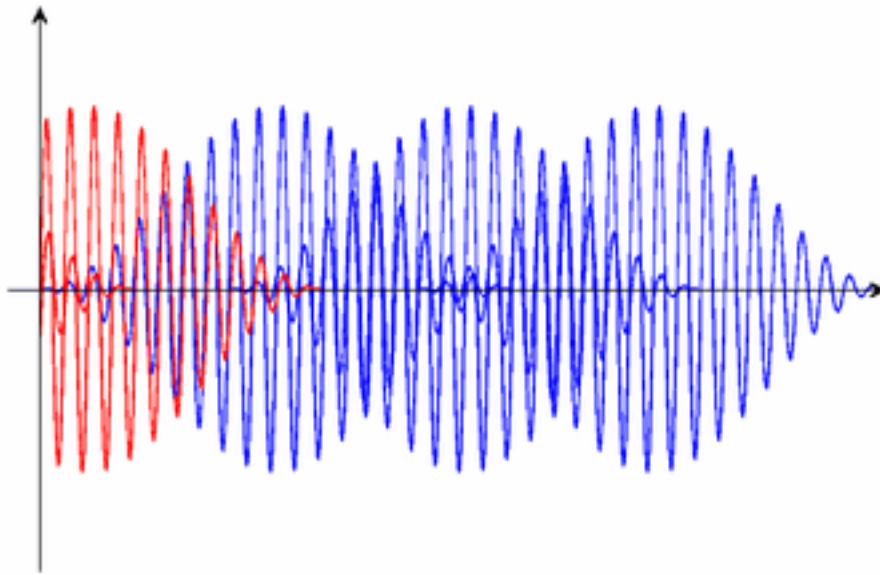
*imaxovr* -- nombre maximum de grains se chevauchant. Le nombre de chevauchements peut être calculé par (*kdens* \* *kgdur*) ; cependant, il peut être surestimé sans coût supplémentaire lors de l'exécution, et un simple chevauchement utilise (selon le système) de 16 à 32 octets en mémoire.

*iwfn* -- table de fonction contenant la forme d'onde d'une fenêtre (utiliser GEN20 pour calculer *iwfn*).

*iseed* (facultatif, par défaut 0) -- valeur de la graine du générateur de nombres aléatoires (entier positif compris entre 1 et 2147483646 ( $2^{31} - 2$ )). Une valeur nulle ou négative force la graine à prendre la valeur de l'horloge de l'ordinateur (c'est le comportement par défaut).

*imode* (facultatif, par défaut 0) -- somme de valeurs prises parmi les suivantes :

- 64 : synchronise la phase au démarrage des grains sur *kcps*.
- 32 : démarre tous les grains sur une position d'échantillon entière. Ceci peut être plus rapide dans certains cas, tout en rendant moins précis le déroulement temporel des enveloppes de grain.
- 16 : ne pas générer de grains ayant une date de démarrage inférieure à zéro. (Voir la figure ci-dessous ; cette option désactive les grains marqués en rouge sur l'image).
- 8 : forme d'onde de la fenêtre avec interpolation (plus lent).
- 4 : pas d'interpolation pour la forme d'onde des grains (rapide, mais de moindre qualité).
- 2 : la fréquence des grains est modifiée continuellement par *kcps* et *kfmd* (par défaut, chaque grain garde la fréquence avec laquelle il a démarré). Avec des taux de contrôle élevés, ceci peut ralentir le processus. Contrôle aussi la modulation de phase (*kphs*)
- 1 : ignorer l'initialisation.



Graphique montrant des grains avec une date de démarrage inférieure à zéro en rouge.

## Exécution

*ares* -- signal de sortie.

*kcps* -- fréquence du grain en Hz.

*kphs* -- phase du grain. C'est une position dans la forme d'onde du grain, exprimée comme une fraction (entre 0 et 1) de la longueur de la table.

*kfmd* -- variation aléatoire (bipolaire) de la fréquence du grain en Hz.

*kpmf* -- variation aléatoire (bipolaire) de la phase au démarrage.

*kgdur* -- durée du grain en secondes. *kgdur* contrôle aussi la durée des grains déjà actifs (en fait la vitesse à laquelle la fonction fenêtre est lue). Ce comportement ne dépend pas des indicateurs positionnés dans *imode*.

*kdens* -- nombre de grains par seconde.

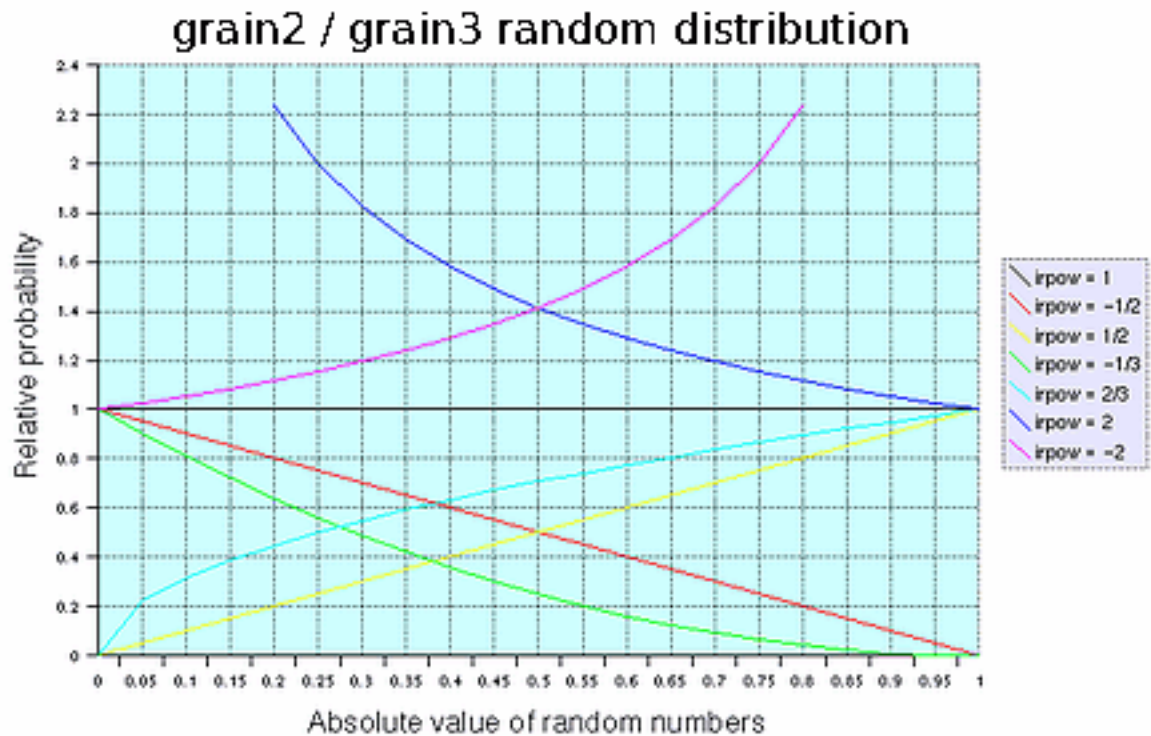
*kfrpow* -- cette valeur contrôle la variation de la distribution de la fréquence du grain. Si *kfrpow* est positif, la distribution aléatoire (*x* est compris entre -1 et 1) est

$$\text{abs}(x)^{((1 / \text{kfrpow}) - 1)} ;$$

pour des valeurs négatives de *kfrpow*, elle est

$$(1 - \text{abs}(x))^{((-1 / \text{kfrpow}) - 1)}$$

En fixant *kfrpow* à -1, 0, ou 1 on obtiendra une distribution uniforme (dont le calcul est plus rapide). L'image ci-dessous montre quelques exemples pour *kfrpow*. La valeur par défaut de *kfrpow* est 0.



Un graphique des distributions pour différentes valeurs de *kfrpow*.

*kprpow* -- variation de la distribution de phase aléatoire (voir *kfrpow*). En fixant *kphs* et *kpmid* à 0.5, et *kprpow* à 0 on émule *grain2*.

*kfn* -- table de fonction contenant la forme d'onde du grain. Le numéro de table peut changer au taux-k (on peut ainsi choisir parmi un ensemble de tables à bande limitée générées par GEN30, afin d'éviter le repliement).



### Note

*grain3* utilise en interne le même générateur aléatoire que *rnd31*. Il est ainsi recommandé de lire également sa *documentation*.

## Exemples

Voici un exemple de l'opcode *grain3*. Il utilise le fichier *grain3.csd* [examples/grain3.csd].

### Exemple 405. Exemple de l'opcode *grain3*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o grain3.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

sr = 48000
kr = 1000
ksmps = 48
nchnls = 1

/* Bartlett window */
itmp ftgen 1, 0, 16384, 20, 3, 1
/* sawtooth wave */
itmp ftgen 2, 0, 16384, 7, 1, 16384, -1
/* sine */
itmp ftgen 4, 0, 1024, 10, 1
/* window for "soft sync" with 1/32 overlap */
itmp ftgen 5, 0, 16384, 7, 0, 256, 1, 7936, 1, 256, 0, 7936, 0
/* generate bandlimited sawtooth waves */
itmp ftgen 3, 0, 4096, -30, 2, 1, 2048
icnt = 0
loop01:
; 100 tables for 8 octaves from 30 Hz
ifrq = 30 * exp(log(2) * 8 * icnt / 100)
itmp ftgen icnt + 100, 0, 4096, -30, 3, 1, sr / (2 * ifrq)
icnt = icnt + 1
if (icnt < 99.5) igoto loop01
/* convert frequency to table number */
#define FRQ2FNUM(xout'xcps'xbsfn) #

$xout = int(($xbsfn) + 0.5 + (100 / 8) * log(($xcps) / 30) / log(2))
$xout limit $xout, $xbsfn, $xbsfn + 99

#

/* instr 1: pulse width modulated grains */

instr 1

kfrq = 523.25 ; frequency
$FRQ2FNUM(kfnum'kfrq'100) ; table number
kfmd = kfrq * 0.02 ; random variation in frequency
kgdur = 0.2 ; grain duration
kdens = 200 ; density
iseed = 1 ; random seed

kphs oscili 0.45, 1, 4 ; phase

a1 grain3 kfrq, 0, kfmd, 0.5, kgdur, kdens, 100, \
kfnum, 1, -0.5, 0, iseed, 2
a2 grain3 kfrq, 0.5 + kphs, kfmd, 0.5, kgdur, kdens, 100, \
kfnum, 1, -0.5, 0, iseed, 2

; de-click
aenv linseg 0, 0.01, 1, p3 - 0.05, 1, 0.04, 0, 1, 0

out aenv * 2250 * (a1 - a2)

endin

/* instr 2: phase variation */

instr 2

kfrq = 220 ; frequency
$FRQ2FNUM(kfnum'kfrq'100) ; table number
kgdur = 0.2 ; grain duration
kdens = 200 ; density
iseed = 2 ; random seed

```

```

kprdst expon 0.5, p3, 0.02 ; distribution

a1 grain3 kfrq, 0.5, 0, 0.5, kgdur, kdens, 100, \
    kfnum, 1, 0, -kprdst, izeed, 64

; de-click
aenv linseg 0, 0.01, 1, p3 - 0.05, 1, 0.04, 0, 1, 0

out aenv * 1500 * a1

endin

/* instr 3: "soft sync" */

instr 3

kdens = 130.8 ; base frequency
kgdur = 2 / kdens ; grain duration

kfrq expon 880, p3, 220 ; oscillator frequency
$FRQ2FNUM(kfnum'kfrq'100) ; table number

a1 grain3 kfrq, 0, 0, 0, kgdur, kdens, 3, kfnum, 5, 0, 0, 0, 2
a2 grain3 kfrq, 0.667, 0, 0, kgdur, kdens, 3, kfnum, 5, 0, 0, 0, 2

; de-click
aenv linseg 0, 0.01, 1, p3 - 0.05, 1, 0.04, 0, 1, 0

out aenv * 10000 * (a1 - a2)

endin

</CsInstruments>
<CsScore>

t 0 60
i 1 0 3
i 2 4 3
i 3 8 3
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*grain2*

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.15

Mise à jour en avril 2002 par Istvan Varga

# granule

granule — Un générateur de texture par synthèse granulaire plus complexe.

## Description

Le générateur unitaire *granule* est plus complexe que *grain*, mais il ajoute de nouvelles possibilités.

*granule* est un générateur unitaire de Csound qui emploie une table d'onde en entrée pour produire une sortie audio par synthèse granulaire. Les données de la table d'onde peuvent être générées par n'importe laquelle des routines GEN telle que *GEN01* qui lit un fichier audio. On peut ainsi utiliser un son échantillonné comme source pour les grains. L'implémentation interne comprend jusqu'à 128 voix. Le nombre maximum de voix peut être augmenté en redéfinissant la variable MAXVOICE dans le fichier grain4.h. *granule* possède son propre générateur de nombres aléatoires pour produire toutes les fluctuations aléatoires des paramètres. Il comprend aussi une fonction de seuil pour scanner la table de fonction source lors de la phase d'initialisation. On peut ainsi facilement ignorer les passages de silence entre les phrases.

Les caractéristiques de la synthèse sont contrôlées par 22 paramètres. *xamp* est l'amplitude de la sortie et elle peut varier aussi bien au taux audio qu'au taux de contrôle.

## Syntaxe

```
ares granule xamp, ivoice, iratio, imode, ithd, ifn, ipshift, igskip, \  
    igskip_os, ilength, kgap, igap_os, kgsz, igsz_os, iatt, idec \  
    [, iseed] [, ipitch1] [, ipitch2] [, ipitch3] [, ipitch4] [, ifnenv]
```

## Initialisation

*ivoice* -- nombre de voix.

*iratio* -- rapport entre la vitesse du pointeur de lecture et le taux d'échantillonnage de la sortie, par exemple 0,5 donnera une vitesse de lecture moitié de la vitesse originale.

*imode* -- +1, le pointeur de lecture progresse en avant (direction naturelle du fichier source), -1, en arrière (direction opposée à la direction naturelle du fichier source), ou 0, pour une direction aléatoire.

*ithd* -- seuil ; lorsque le signal échantillonné dans la table est plus petit que *ithd*, il est ignoré.

*ifn* -- numéro de la table de fonction de la source sonore.

*ipshift* -- contrôle de la transposition. Si *ipshift* vaut 0, la hauteur sera fixée aléatoirement dans un ambitus d'une octave de part et d'autre de la hauteur de chaque grain. Si *ipshift* vaut 1, 2, 3 ou 4, on peut fixer jusqu'à quatre hauteurs différentes pour le nombre de voix défini dans *ivoice*. Les paramètres facultatifs *ipitch1*, *ipitch2*, *ipitch3* et *ipitch4* servent à quantifier les transpositions.

*igskip* -- décalage initial depuis le début de la table de fonction en sec.

*igskip\_os* -- fluctuation aléatoire du pointeur de lecture en sec, 0 signifiant pas de décalage.

*ilength* -- longueur de la partie de la table à utiliser à partir de *igskip* en sec.

*igap\_os* -- fluctuation aléatoire de l'écart en % de la taille de l'écart, 0 signifiant pas de décalage.

*igsz\_os* -- fluctuation aléatoire de la taille du grain en % de la taille du grain, 0 signifiant pas de décalage.

*iatt* -- attaque de l'enveloppe du grain en % de la taille du grain.

*idec* -- chute de l'enveloppe du grain en % de la taille du grain.

*iseed* (facultatif, par défaut 0,5) -- graine pour le générateur de nombre aléatoire.

*ipitch1*, *ipitch2*, *ipitch3*, *ipitch4* (facultatif, par défaut 1) -- paramètre de transposition, utilisé lorsque *ipshift* vaut 1, 2, 3 ou 4. La transposition est réalisée par une technique de pondération temporelle avec interpolation linéaire entre les points. La valeur par défaut de 1 signifie la hauteur originale.

*ifnenv* (facultatif, par défaut 0) -- numéro de la table de fonction utilisée pour générer la forme de l'enveloppe.

## Exécution

*xamp* -- amplitude.

*kgap* -- écart entre les grains en sec.

*kgsiz*e -- taille du grain en sec.

## Exemples

Voici un exemple de l'opcode *granule*. Il utilise les fichiers *granule.csd* [examples/granule.csd], et *mary.wav* [examples/mary.wav].

### Exemple 406. Exemple de l'opcode *granule*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o granule.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
instr 1
;
k1      linseg 0,0.5,1,(p3-p2-1),1,0.5,0
a1      granule p4*k1,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15,\
        p16,p17,p18,p19,p20,p21,p22,p23,p24
a2      granule p4*k1,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15,\
        p16,p17,p18,p19, p20+0.17,p21,p22,p23,p24
outs a1,a2
endin

</CsInstruments>
<CsScore>
```

```

; f statement read sound file mary.wav in the SFDIR
; directory into f-table 1
f1      0 262144 1 "mary.wav" 0 0 0
i1      0 10 2000 64 0.5 0 0 1 4 0 0.005 5 0.01 50 0.02 50 30 30 0.39 \
      1 1.42 0.29 2
e

</CsScore>
</CsoundSynthesizer>

```

L'exemple ci-dessus lit un fichier son nommé *mary.wav* dans la table de fonction numéro 1 en gardant 262 144 échantillons. Il génère 10 secondes de sortie stéréo à partir de la table de fonction. Dans le fichier orchestre, tous les paramètres nécessaires au contrôle de la synthèse proviennent du fichier partition. Un générateur de fonction *linseg* est utilisé pour produire une enveloppe avec une attaque et une chute linéaires de 0,5 secondes. On obtient un effet stéréo par l'utilisation de différentes graines pour les deux appels de la fonction *granule*. Dans l'exemple, on ajoute 0,17 à p20 avant de le passer au second appel de *granule* pour s'assurer que toutes les fluctuations aléatoires seront différentes de celles du premier appel.

Voici la signification des paramètres dans le fichier partition :

Parameter	Interpreted As
p5 ( <i>ivoice</i> )	le nombre de voix est fixé à 64
p6 ( <i>iratio</i> )	fixé à 0,5, on lit la table d'onde deux fois moins vite que le taux de la sortie audio
p7 ( <i>imode</i> )	fixé à 0, le pointeur du grain ne se déplace qu'en avant
p8 ( <i>ithd</i> )	fixé à 0, pas de détection de seuil
p9 ( <i>ifn</i> )	fixé à 1, on utilise la table de fonction numéro 1
p10 ( <i>ipshift</i> )	fixé à 4, quatre hauteurs différentes seront générées
p11 ( <i>igskip</i> )	fixé à 0 et p12 ( <i>igskip_os</i> ) est fixé à 0,005, pas de décalage par rapport au début de table d'onde et on utilise une fluctuation aléatoire de 5 ms
p13 ( <i>ilength</i> )	fixé à 5, on n'utilise que 5 secondes de la table d'onde
p14 ( <i>kgap</i> )	fixé à 0,01 et p15 ( <i>igap_os</i> ) est fixé à 50, on utilise un écart de 10 ms avec une fluctuation aléatoire de 50%
p16 ( <i>kgsz</i> )	fixé à 0,02 et p17 ( <i>igsize_os</i> ) est fixé à 50, la durée du grain est de 20 ms avec une fluctuation aléatoire de 50%
p18 ( <i>iatt</i> ) et p19 ( <i>idec</i> )	fixés à 30, on applique une attaque et une chute linéaires de 30% au grain
p20 ( <i>iseed</i> )	la graine pour le générateur de nombre aléatoire est fixée à 0,39
p21 - p24	les hauteurs sont fixées à 1, soit la hauteur originale, 1,42 soit une quinte plus haut, 0,29 soit une septième plus bas et enfin 2 soit une octave plus haut.

## Crédits

Auteur : Allan Lee



Belfast

1996

Nouveau dans la version 3.35

# guiro

guiro — Modèle semi-physique d'un son de guiro.

## Description

*guiro* est un modèle semi-physique d'un son de guiro. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntaxe

```
ares guiro kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] [, ifreq1]
```

## Initialisation

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (optional) -- (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 128.

*idamp* (facultatif) -- le facteur d'amortissement de l'instrument. *Inutilisé.*

*imaxshake* (facultatif, 0 par défaut) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

*ifreq* (facultatif) -- la fréquence de résonance principale. La valeur par défaut est 2500 Hz.

*ifreq1* (facultatif) -- la première fréquence de résonance. La valeur par défaut est 4000 Hz.

## Exécution

*kamp* -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

## Exemples

Voici un exemple de l'opcode guiro. Il utilise le fichier *guiro.csd* [examples/guiro.csd].

### Exemple 407. Exemple de l'opcode guiro.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o guiro.wav -W ;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
odbfs = 1

instr 1

a1 guiro .8, p4
outs a1, a1

endin
</CsInstruments>
<CsScore>

i1 0 1 1
i1 + 1 .01
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*bamboo, dripwater, sleighbells, tambourine*

## Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)

Adapté par John ffitch

Université de Bath, Codemist Ltd.

Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

# harmon

harmon — Analyse une entrée audio et génère des voix harmoniques synchrones.

## Description

Analyse une entrée audio et génère des voix harmoniques synchrones.

## Syntaxe

```
ares harmon asig, kestfrq, kmaxvar, kgenfreq1, kgenfreq2, imode, \  
    iminfrq, iprd
```

## Initialisation

*imode* -- mode d'interprétation des entrées de génération de fréquence *kgenfreq1*, *kgenfreq2*. 0 : les valeurs entrées sont des rapports de la fréquence analysée du signal audio. 1 : les valeurs entrées sont les fréquences demandées en Hz.

*iminfrq* -- la fréquence la plus basse en Hz attendue dans l'entrée audio. Ce paramètre détermine la quantité de signal en entrée qui est enregistrée pour l'analyse courante et fixe une limite inférieure au détecteur de hauteur interne.

*iprd* -- période d'analyse (en secondes). Comme l'analyse de hauteur interne peut prendre du temps, l'entrée est typiquement analysée seulement toutes les 20 à 50 ms.

## Exécution

*kestfrq* -- fréquence estimée de l'entrée.

*kmaxvar* -- variance maximale (valeur attendue entre 0 et 1).

*kgenfreq1* -- la première fréquence générée.

*kgenfreq2* -- la seconde fréquence générée.

Cette unité est un harmoniseur, capable d'ajouter jusqu'à deux voix supplémentaires avec la même amplitude et le même spectre que l'entrée. L'analyse de l'entrée est facilitée par deux éléments : une estimation de la fréquence de l'entrée *kestfrq* (en Hz) et une variance fractionnaire maximale *kmaxvar* autour de cette estimation, qui sert à limiter la taille de la recherche. Une fois la fréquence réelle de l'entrée déterminée, la forme de pulsation la plus récente est utilisée pour générer les autres voix aux fréquences demandées.

Les trois entrées de fréquence peuvent être dérivées de diverses manières depuis un fichier de partition ou depuis une source MIDI. La première est la hauteur attendue, avec un paramètre de variance permettant les inflexions ou les approximations ; si la hauteur attendue vaut zéro l'harmoniseur sera silencieux. Les seconde et troisième hauteurs contrôlent les fréquences de sortie ; si l'une d'elles vaut zéro, l'harmoniseur ne générera que la fréquence demandée différente de zéro ; si les deux sont nulles, l'harmoniseur sera silencieux. Lorsque la fréquence demandée est plus haute que l'entrée, le procédé demande plus de calculs à cause de la superposition des pulsations en sortie. Pour des raisons d'efficacité, ceci est actuellement limité, ce qui a pour résultat de ne permettre à tout moment qu'une seule voix plus haute que l'entrée.

Cette unité est utile pour fournir à la demande un effet de chœur en fond, ou bien pour corriger la hauteur d'une voix un peu faussée en entrée. Il n'y a pratiquement pas de délai entre l'entrée et la sortie. La sortie ne comprend que les parties générées sans l'entrée.

## Exemples

Voici un exemple de l'opcode *harmon*. Il utilise le fichier *harmon.csd* [examples/harmon.csd].

### Exemple 408. Exemple de l'opcode *harmon*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime output leave only the line below:
; -o harmon.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

aout diskin2 "sing.wav", 1

kestfrq = p4 ;different estimated frequency
kmaxvar = 0.1
imode = 1
iminfrq = 100
iprd = 0.02

asig harmon aout, kestfrq, kmaxvar, kestfrq*.5, kestfrq*4, \
      imode, iminfrq, iprd
outs (asig + aout)*.6, (asig + aout)*.6 ;mix dry&wet signal

endin
</CsInstruments>
<CsScore>

i 1 0 2.7 100
i 1 + . 200
i 1 + . 500
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Barry L. Vercoe  
M.I.T., Cambridge, Mass  
1997

Nouveau dans la version 3.47

# harmon2

**harmon2** — Analyse une entrée audio et génère des voix harmoniques synchrones avec préservation des formants.

## Description

Génère des voix harmoniques avec préservation des formants.

## Syntaxe

```
ares harmon2 asig, koct, kfrq1, kfrq2, icpsmode, ilowest[, ipolarity]

ares harmon3 asig, koct, kfrq1, \
    kfrq2, kfrq3, icpsmode, ilowest[, ipolarity]

ares harmon4 asig, koct, kfrq1, \
    kfrq2, kfrq3, kfrq4, icpsmode, ilowest[, ipolarity]
```

## Initialisation

*icpsmode* -- mode d'interprétation des entrées de génération de fréquence *kfrq1*, *kfrq2*, *kfrq3* et *kfrq4* : 0 : les valeurs entrées sont des rapports de l'équivalent en fréquence (Hz) de *koct*. 1 : les valeurs entrées sont les fréquences demandées en Hz.

*ilowest* -- valeur la plus basse de *koct* pour laquelle des voix harmoniques seront générées.

*ipolarity* -- polarité de l'entrée *asig*, 1 = pulsations glottales positives, 0 = négatives. La valeur par défaut est 1.

## Exécution

**Harmon2**, **harmon3** et **harmon4** sont des harmoniseurs très performants, capables de générer jusqu'à quatre copies transposées de l'entrée *asig* avec préservation des formants. L'algorithme de transposition nécessite une estimation précise (*koct*, en unités décimales d'oct) de la hauteur de *asig*, normalement obtenue par un détecteur de hauteur indépendant comme *specptrk*. L'algorithme isole ensuite la pulsation pleine la plus récente dans *asig* et l'utilise pour générer les autres voix avec les taux de pulsation requis.

Si la fréquence (ou le rapport) présenté pour *kfrq1*, *kfrq2*, *kfrq3* ou *kfrq4* vaut zéro, aucun signal n'est généré pour cette voix. S'il y en a qui sont différents de zéro, mais que l'entrée *koct* est inférieure à la valeur *ilowest*, alors cette voix sortira une copie directe de l'entrée *asig*. En conséquence, les données arrivant sur les entrées de taux-k peuvent au choix activer ou désactiver les voix générées, passer une copie directe d'une source fricative non voisée ou harmoniser la source en fonction d'un algorithme construit. La transition d'un mode à l'autre est progressive, ce qui donne une alternance continue entre les sons voisés (harmonisés) et les fricatives non-voisées d'une entrée parlée ou chantée.

*harmon2*, *harmon3*, *harmon4* sont spécialement adaptés à la sortie de *specptrk*. Ce dernier génère des données de hauteur en format décimal d'octave ; il retourne également sa valeur de base si aucune hauteur n'est identifiée (comme dans un bruit de fricative) et émet zéro si l'énergie tombe en-dessous du seuil, si bien que *harmon2*, *harmon3*, *harmon4* peuvent être réglés pour passer le signal direct dans les deux cas. Naturellement, on pourrait utiliser n'importe quelle autre forme d'estimation de la hauteur. Comme les détecteurs de hauteur subissent habituellement un léger retard lors d'une estimation précise (pour *specptrk*

le retard est imposé par l'unité *spectrum*), il est normal de retarder le signal audio de la même durée pour que *harmon2*, *harmon3*, *harmon4* puissent travailler à partir d'une estimation synchrone.

## Exemples

Voici un exemple de l'opcode *harmon2*. Il utilise le fichier *harmon.csd* [exemples/harmon.csd].

### Exemple 409. Exemple de l'opcode *harmon2*.

```

a1,a2      ins                                ; récupère l'entrée mic
w1         spectrum a1, .02, 7, 24, 12, 1, 3    ; et l'examine
koct,kamp  specptrk w1, 1, 6.5, 9.5, 7.5, 10, 7, .7, 0, 3, 1
a3         delay a1, .065                      ; retarde ptrk
a4         harmon2 a3, koct, 1.25, 0.75, 0, 6.9 ; sort une harmonie fixe 6-4
          outs a3, a4                          ; ainsi que l'original

```

Voici un exemple complet de l'opcode *harmon3*. Il utilise le fichier *harmon3.csd* [exemples/harmon3.csd].

### Exemple 410. Exemple de l'opcode *harmon3*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o harmon3.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ilow = p4      ;lowest value to harmonize
aout diskin2 "sing.wav", 1, 0, 1
koct, kamp pitch aout, .01, 6, 10, 10 ;track pitch
asig harmon3 aout, koct, .9, 1.5, 0.7, 0, ilow
      outs (asig + aout)*.4, (asig + aout)*.4 ;mix dry&wet signal

endin
</CsInstruments>
<CsScore>

i1 0 2.2 8.8
i1 3 2.2 8.2
i1 6 2.2 7.0

e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Barry L. Vercoe

M.I.T., Cambridge, Mass  
2006

Nouveau dans la version 5.04



# hdf5read

hdf5read — Lit des signaux et des tableaux depuis un fichier hdf5.

## Description

Opcodes du greffon hdf5ops.

*hdf5read* lit *N* signaux et tableaux depuis un fichier hdf5 spécifié.

## Syntaxe

```
xout1[, xout2, xout3, ..., xoutN] hdf5read ifilename, ivariablename1[, ivariablename2, ivariablename3,
```

## Initialisation

*ifilename* -- le nom du fichier hdf5 ( entre guillemets).

*ivariablename1[, ivariablename2, ivariablename3, ..., ivariablenameN]* -- les noms des ensembles de données (entre guillemets) à lire depuis le fichier hdf5 ; si le nom d'un ensemble de données est suffixé avec un astérisque, par exemple "monEnsembleDeDonnées\*", tout l'ensemble de données est copié dans le tableau quelque soit le type du tableau.

## Exécution

*xout1, ..., xoutN* -- les variables typées dans lesquelles les ensembles de données hdf5 sont lus. Les ensembles de données de rang supérieur à 1 doivent être lus dans des tableaux, les signaux de taux-i doivent être lus également dans des signaux de taux-i. En dehors de ces restrictions, les ensembles de données peuvent être lus dans n'importe quel type de tableau ou de signal. Lorsque la lecture atteint le fin d'un ensemble de données, celui-ci cesse de fournir de nouvelles valeurs.

## Exemples

Voici un exemple de l'opcode hdf5read. Il utilise le fichier *hdf5read.csd* [examples/hdf5read.csd].

### Exemple 411. Exemple de l'opcode hdf5read.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>
nchnls = 2
0dbfs = 1
ksmps = 8
sr = 44100

instr hdf5read
```

```
aArray[], aVar, kVar hdf5read "example.h5", "aArray", "aVar", "kVar" ; Open hdf5 file and read variables

aLeft = (aArray[0][0] + aArray[0][1] + aVar) / 3 ; Add audio signals together for stereo out
aRight = (aArray[1][0] + aArray[1][1] + aVar) / 3

outs aLeft * kVar, aRight * kVar ; Multiply audio signals by k-rate signal
endin

</CsInstruments>
<CsScore>

i "hdf5read" 0 1

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*hdf5write*

## Crédits

Auteur : Edward Costello;  
NUIM, 2014

# hdf5write

hdf5write — Écrit des signaux et des tableaux dans un fichier hdf5.

## Description

Opcodes du greffon hdf5ops.

*hdf5write* écrit *N* signaux et tableaux dans un fichier hdf5 spécifié.

## Syntaxe

```
hdf5write ifilename, xout1[, xout2, xout3, ..., xoutN]
```

## Initialisation

*ifilename* -- le nom du fichier hdf5 ( entre guillemets). Si le fichier n'existe pas, il est créé.

## Exécution

*xout1*, ..., *xoutN* -- signaux ou tableaux à écrire dans le fichier hdf5. Cet opcode accepte des signaux de taux-i, taux-k, taux-a ou des tableaux de taux-i, taux-k, taux-a, de n'importe quelle dimension. Ces signaux ou ces tableaux sont écrits dans un ensemble de données du fichier hdf5 utilisant le même nom de variable que dans Csound. Par exemple, si la variable Csound s'appelle 'ksignal', le nom de l'ensemble de données hdf5 sera 'ksignal'. N'importe quel nombre d'ensemble de données de différents types peuvent être écrits à chaque appel de cet opcode.

## Exemples

Voici un exemple de l'opcode hdf5write. Il utilise le fichier *hdf5write.csd* [examples/hdf5write.csd].

### Exemple 412. Exemple de l'opcode hdf5write.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>
nchnls = 2
0dbfs = 1
ksmps = 8
sr = 44100

instr hdf5write

  aArray[] init 2,2 ; Initialise a 2 X 2 a-rate array

  aArray[0][0] vco2 0.2, 100 ; Fill array with vco2 signals
  aArray[0][1] vco2 0.4, 200
  aArray[1][0] vco2 0.8, 300
```

```
aArray[1][1] vco2 1, 400

aVar vco2 0.2, 100 ; Initialise an a-rate variable with a vco2 signal

kVar phasor 1 ; Initialise a k-rate variable with a phasor signal

hdf5write "example.h5", aArray, aVar, kVar ; Write variables to an hdf5 file

endin

</CsInstruments>
<CsScore>

i "hdf5write" 0 1

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*hdf5read*

## Crédits

Auteur : Edward Costello;  
NUIM, 2014

# hilbert

hilbert — Une transformée de Hilbert.

## Description

Une implémentation RII de la transformée de Hilbert.

## Syntaxe

```
ar1, ar2 hilbert asig
```

## Exécution

*asig* -- signal d'entrée.

*ar1* -- sortie sinus de *asig*.

*ar2* -- sortie cosinus de *asig*.

*hilbert* est un filtre RII basé sur l'implémentation d'un réseau déphaseur de 90 degrés à large bande. L'entrée de *hilbert* est un signal audio dont la fréquence peut aller de 15 Hz à 15 kHz. Les sorties de *hilbert* ont la même réponse en fréquence que l'entrée (même sonorité), mais les deux sorties ont un déphasage constant de 90 degrés, plus ou moins un petit delta d'erreur, sur toute la gamme de fréquence. Les sorties sont en quadrature de phase.

*hilbert* est utile dans l'implémentation de plusieurs techniques de traitement numérique du signal en quadrature de phase. *ar1* correspond à la sortie cosinus de *hilbert*, tandis que *ar2* correspond à la sortie sinus. Les deux sorties ont un déphasage constant sur tout l'intervalle audio correspondant à la relation de phase entre une onde cosinus et une onde sinus.

En interne, *hilbert* est basé sur deux filtres passe-tout du sixième ordre en parallèle. Chaque filtre passe-tout implémente un retard qui augmente avec la fréquence ; la différence entre les retards de phase des filtres passe-tout en parallèle est approximativement de 90 degrés en n'importe quel point.

Contrairement à une transformée de Hilbert à RIF, la sortie de *hilbert* n'a pas une réponse en phase linéaire. Cependant, la structure à RII utilisée dans *hilbert* est calculée de manière bien plus efficace, et la réponse en phase non-linéaire peut être utilisée pour créer des effets audio intéressants, comme dans le deuxième exemple ci-dessous.

## Exemples

Le premier exemple implémente un décalage de fréquence, ou modulation d'amplitude à bande latérale unique. Le décalage de fréquence est semblable à la modulation en anneau, sauf que les bandes latérales supérieure et inférieure sont séparées sur des sorties différentes. En n'utilisant qu'une seule de ces sorties, le signal d'entrée peut être "désaccordé" car le décalage des composants du signal détruit leur relation harmonique ; par exemple, un signal d'harmoniques à 100, 200, 300, 400 et 500 Hz, décalé vers le haut de 50 Hz, aura ses composants placés à 150, 250, 350, 450 et 550 Hz.

Premier exemple de l'opcode *hilbert*. Il utilise les fichiers *hilbert.csd* [exemples/hilbert.csd] et *beats.wav* [exemples/beats.wav].

**Exemple 413. Exemple de l'opcode hilbert implémentant le décalage de fréquence.**

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o hilbert.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
  idur = p3
  ; Initial amount of frequency shift.
  ; It can be positive or negative.
  ibegshift = p4
  ; Final amount of frequency shift.
  ; It can be positive or negative.
  iendshift = p5

  ; A simple envelope for determining the
  ; amount of frequency shift.
  kfreq linseg ibegshift, idur, iendshift

  ; Use the sound of your choice.
  ain diskin2 "beats.wav", 1, 0, 1

  ; Phase quadrature output derived from input signal.
  areal, aimag hilbert ain

  ; Quadrature oscillator.
  asin oscili 1, kfreq, 1
  acos oscili 1, kfreq, 1, .25

  ; Use a trigonometric identity.
  ; See the references for further details.
  amod1 = areal * acos
  amod2 = aimag * asin

  ; Both sum and difference frequencies can be
  ; output at once.
  ; aupshift corresponds to the sum frequencies.
  aupshift = (amod1 - amod2) * 0.7
  ; adownshift corresponds to the difference frequencies.
  adownshift = (amod1 + amod2) * 0.7

  ; Notice that the adding of the two together is
  ; identical to the output of ring modulation.

  outs aupshift, aupshift
endin

</CsInstruments>
<CsScore>

; Sine table for quadrature oscillator.

```

```
f 1 0 16384 10 1

; Starting with no shift, ending with all
; frequencies shifted up by 2000 Hz.
i 1 0 6 0 2000

; Starting with no shift, ending with all
; frequencies shifted down by 250 Hz.
i 1 7 6 0 -250
e
```

```
</CsScore>
</CsoundSynthesizer>
```

Le second exemple est une variation du premier, mais avec réinjection de la sortie dans l'entrée. Avec de très petits décalages (entre 0 et  $\pm 6$  Hz), le résultat est un son qui a été décrit comme un « déphaseur en enseigne de salon de coiffure américain » ou comme un « déphaseur en gamme de Shepard ». Plusieurs creux apparaissent dans le spectre et glissent à vitesse constante dans la direction opposée au décalage, produisant un effet de filtrage rappelant le « glissando sans fin » de Risset.

Second exemple de l'opcode hilbert. Il utilise le fichier *hilbert\_barberpole.csd* [exemples/hilbert\_barberpole.csd].

#### Exemple 414. Exemple de l'opcode hilbert sonnante comme une « enseigne de coiffeur ».

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o hilbert_barberpole.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
; kr must equal sr for the barberpole effect to work.
kr = 44100
ksmps = 1
nchnls = 2

; Instrument #1
instr 1
  idur = p3
  ibegshift = p4
  iendshift = p5

  ; sawtooth wave, not bandlimited
  asaw phasor 100
  ; add offset to center phasor amplitude between -.5 and .5
  asaw = asaw - .5
  ; sawtooth wave, with amplitude of 10000
  ain = asaw * 20000

  ; The envelope of the frequency shift.
  kfreq linseg ibegshift, idur, iendshift
```

```
; Phase quadrature output derived from input signal.
areal, aimag hilbert ain

; The quadrature oscillator.
asin oscili 1, kfreq, 1
acos oscili 1, kfreq, 1, .25

; Based on trigonometric identities.
amod1 = areal * acos
amod2 = aimag * asin

; Calculate the up-shift and down-shift.
aupshift = (amod1 + amod2) * 0.7
adownshift = (amod1 - amod2) * 0.7

; Mix in the original signal to achieve the barberpole effect.
amix1 = aupshift + ain
amix2 = adownshift + ain

; Make sure the output doesn't get louder than the original signal.
aout1 balance amix1, ain
aout2 balance amix2, ain

outs aout1, aout2
endin

</CsInstruments>
<CsScore>

; Table 1: A sine wave for the quadrature oscillator.
f 1 0 16384 10 1

; The score.
; p4 = frequency shifter, starting frequency.
; p5 = frequency shifter, ending frequency.
i 1 0 6 -10 10
e

</CsScore>
</CsoundSynthesizer>
```

## Historique Technique

L'utilisation de réseaux déphaseurs dans le décalage de fréquence fut initialisée par Harald Bode<sup>1</sup>. Bode et Bob Moog donnent une excellente description de l'implémentation et de l'utilisation du décalage de fréquence dans le domaine analogique dans <sup>2</sup>; c'est une source excellente pour une première exploration des possibilités de la modulation à bande latérale unique. Bernie Hutchins donne plus d'applications du décalage de fréquence ainsi qu'une analyse technique détaillée <sup>3</sup>. Un papier récent de Scott Wardle<sup>4</sup> décrit une implémentation numérique du décalage de fréquence ainsi que quelques applications uniques.

## Références

1. H. Bode, "Solid State Audio Frequency Spectrum Shifter." AES Preprint No. 395 (1965).
2. H. Bode and R.A. Moog, "A High-Accuracy Frequency Shifter for Professional Audio Applications." *Journal of the Audio Engineering Society*, Juillet/Août 1972, vol. 20, no. 6, p. 453.
3. B. Hutchins. *Musical Engineer's Handbook* (Ithaca, NY: Electronotes, 1975), ch. 6a.
4. S. Wardle, "A Hilbert-Transformer Frequency Shifter for Audio". Accessible en ligne à <http://www.iaa.upf.es/dafx98/papers/>.



## Crédits

Auteur : Sean Costello  
Seattle, Washington  
1999

Nouveau dans la version 3.55 de Csound.

Les exemples ont été mis à jour en avril 2002. L'exemple barberpole a été corrigé par Sean Costello.

# hilbert2

hilbert2 — Une transformée de Hilbert.

## Description

Une implémentation TFD de la transformée de Hilbert.

## Syntaxe

```
ar1, ar2 hilbert2 asig, ifftsize, ihopsize
```

## Initialisation

*ifftsize* -- taille d'analyse de la TFD

*ihopsize* -- taille du saut dans l'analyse

## Exécution

*asig* -- signal en entrée

*ar1* -- sortie réelle de *asig*

*ar2* -- sortie imaginaire de *asig*

*hilbert2* is a DFT-based implementation of the Hilbert Transform producing two outputs in quadrature (90 degree phase difference across the spectrum). Unlike the IIR-based *hilbert* opcode, *hilbert2* has a linear frequency response. Given that it employs a streaming algorithm, a delay of *fftsize* samples will be imposed between input and output.

## Exemple

Voici un exemple de l'opcode *hilbert2*. Il utilise le fichier *hilbert2.csd* [examples/hilbert2.csd].

### Exemple 415. Exemple de l'opcode *hilbert2*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

instr 1

asig oscili p4, p5
a1,a2 hilbert2 asig,1024,256
outs a1, a2

endin
```

```
</CsInstruments>  
<CsScore>  
i1 0 10 0.5 440  
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
2017

# hrtfearly

*hrtfearly* — Génération audio binaurale 3D avec des premières réflexions haute-fidélité dans une pièce paramétrique au moyen d'un algorithme de troncature de phase.

## Description

Cet opcode inclut essentiellement l'opcode *hrtfmove* dans la modélisation d'une pièce en forme de boîte à chaussure définie par l'utilisateur. On peut choisir une pièce par défaut ou l'on peut définir des paramètres avancés de la pièce. Les surfaces de la pièce peuvent être contrôlées avec les coefficients d'absorption hautes et basses fréquences et avec les facteurs de gain d'un égaliseur à trois bandes.

Bien que valide comme opcode autonome, *hrtfearly* est conçu pour fonctionner avec *hrtfverb* afin de fournir une réverbération binaurale dynamique spatialement fidèle. Plusieurs sources peuvent être traitées dynamiquement en utilisant plusieurs instances de *hrtfearly*. Toutes peuvent ensuite être traitées avec une instance de *hrtfverb*.

## Syntaxe

```
aleft, aright, irt60low, irt60high, imfp hrtfearly asrc, ksrcx, ksrcy, ksrcz, klstnrx, klstnry, klstnrz,
ifilel, ifiler, ideoom [ifade, isr, iorder, ithreed, kheadrot, iroomx, iroomy, iroomz, iwallh,
iwalllow, iwallgain1, iwallgain2, iwallgain3, ifloorhigh, ifloorlow, ifloorgain1, ifloorgain2, \
ifloorgain3, iceilinghigh, iceilinglow, iceilinggain1, iceilinggain2, iceilinggain3]
```

## Initialisation

## Initialisation

## Initialisation

*ifilel* - fichier des données spectrales HRTF de gauche.

*ifiler* - fichier des données spectrales HRTF de droite.



### Note

Des fichiers de données spectrales (basés sur la base de données HRTF du MIT) sont disponibles sous trois différents taux d'échantillonnage : 44.1, 48 et 96 kHz et sont nommés en conséquence. Le *sr* d'entrée et de traitement doit correspondre au *sr* du fichier de données. Les fichiers doivent se trouver dans le répertoire courant ou dans SADIR (voir *Variables d'environnement*).



### Note

Les fichiers de données HRTF à utiliser avec *hrtfmove*, *hrtfmove2*, *hrtfstat*, *hrtfearly* et *hrtfverb* ont été mis à jour pour les versions 5.15 et ultérieures de Csound (le code a été mis à jour et il est plus efficace). Les anciens fichiers de données sont maintenant obsolètes.

*ideoom* - pièce par défaut, moyenne (1 : 10\*10\*3), petite (2 : 3\*4\*3) ou grande (3 : 20\*25\*7). Détails des murs (coefficients d'absorption hautes et basses fréquences, gain1, gain2, gain3) : 0.3, 0.1, 0.75, 0.95,

0.9. Plancher : 0.6, 0.1, 0.95, 0.6, 0.35. Plafond : 0.2, 0.1, 1, 1, 1. Si ce paramètre vaut 0, les paramètres de pièce facultatifs seront lus.

*ifade* - facultatif, nombre de tampons de traitement pour les fondus de changement de phase (8 par défaut). Compris entre 1 et 24. Voir *hrtfmove*.

*isr* - facultatif, 44.1kHz par défaut. Valeurs acceptées : 44100, 48000 et 96000.

*iorder* - facultatif, ordre des images traitées : plus il est élevé et plus il y a de premières réflexions. Vaut 1 par défaut. Valeurs acceptées : 0 à 4.

*ithreed* - facultatif, les sources d'image sont traitées en trois dimensions (1) ou en deux dimensions (0, par défaut).

*iroomx* - facultatif, taille x de la pièce en mètres. Ce paramètre est lu si aucune pièce par défaut n'est choisie (tous les paramètres ci-dessous se comportent de la même manière). La taille minimale de la pièce est 2\*2\*2.

*iroomy* - facultatif, taille y de la pièce.

*iroomz* - facultatif, taille z de la pièce.

*iwallhigh* - facultatif, coefficient d'absorption hautes fréquences des murs (les quatre murs sont supposés identiques). Les coefficients d'absorption affectent la durée de la réverbération en sortie.

*iwallow* - facultatif, coefficient d'absorption basses fréquences des murs.

*iwallgain1* - facultatif, gain du filtre centré sur 250 Hz (tous les filtres ont un Q impliquant 4 octaves).

*iwallgain2* - facultatif, gain du filtre centré sur 1000 Hz.

*iwallgain3* - facultatif, gain du filtre centré sur 4000 Hz.

*ifloorhigh*, *ifloorlow*, *ifloorgain1*, *ifloorgain2*, *ifloorgain3* - facultatif, comme ci-dessus, pour le plancher.

*iceilinghigh*, *iceilinglow*, *iceilinggain1*, *iceilinggain2*, *iceilinggain3* - facultatif, comme ci-dessus, pour le plafond.

*ksrcx* position x de la source, au moins 10 cm à l'intérieur de la pièce. De plus, les HRTF proches ne sont pas traités, afin que la source ne soit pas modifiée spatialement dans un rayon de 45 cm autour de l'auditeur. Ces restrictions s'appliquent également aux paramètres ci-dessous.

*ksrcy* position y de la source.

*ksrcz* position z de la source.

*klstnr<sub>x</sub>*, *klstnr<sub>y</sub>*, *klstnr<sub>z</sub>* position de l'auditeur, comme ci-dessus.

*kheadrot* - facultatif, angle de rotation de la tête.

*asrc* - Signal source en entrée.

*irt60low* - durée de réverbération basses fréquences suggérée pour une réverbération binaurale consécutive.

*irt60high* - comme ci-dessus, pour les hautes fréquences.

*imfp* - chemin libre moyen de la pièce, à utiliser avec une réverbération consécutive.

## Exemples

Voici un exemple des opcodes `hrtfearly` et `hrtfreverb`. Il utilise le fichier `hrtfearly.csd` [examples/hrtfearly.csd].

### Exemple 416. Exemple de l'opcode `hrtfearly`.

```
<CsoundSynthesizer>
<CsOptions>

; Select flags here
; realtime audio out
-o dac
; file output
; -o hrtf.wav

</CsOptions>
<CsInstruments>

nchnls = 2

gasrc init 0 ;global

instr 1 ;a plucked string, distorted and filtered

    iamp = 15000
    icps = cpspch(p4)

    a1 pluck iamp, icps, icps, 0, 1
    adist distort1 a1, 10, .5, 0, 0
    afilt moogvcf2 adist, 8000, .5
    aout linen afilt, 0, p3, .01

    gasrc = gasrc + aout

endin

instr 10 ;uses output from instr1 as source

    ;simple path for source
    kx line 2, p3, 9

    ;early reflections, room default 1
    aearlyl, aearlyr, irt60low, irt60high, imfp hrtfearly gasrc, kx, 5, 1, 5, 1, 1, "hrtf-44100-left.dat",

    ;later reverb, uses outputs from above
    arevl, arevr, idel hrtfreverb gasrc, irt60low, irt60high, "hrtf-44100-left.dat", "hrtf-44100-right.dat",

    ;delayed and scaled
    alatel delay arevl * .1, idel
    alater delay arevr * .1, idel

    outs aearlyl + alatel, aearlyr + alater

    gasrc = 0

endin

</CsInstruments>
<CsScore>

; Play Instrument 1: a simple arpeggio
i1 0 .2 8.00
```

```
i1 + .2 8.04
i1 + .2 8.07
i1 + .2 8.11
i1 + .2 9.02
i1 + 1.5 8.11
i1 + 1.5 8.07
i1 + 1.5 8.04
i1 + 1.5 8.00
i1 + 1.5 7.09
i1 + 4 8.00

; Play Instrument 10 for 13 seconds.
i10 0 13

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*hrtfreverb hrtfmove, hrtfmove2, hrtfstat, hrtfer.*

## Crédits

Auteur : Brian Carty  
Maynooth  
2011

# hrtfmove

**hrtfmove** — Génère un signal audio 3D binaural pour casque par magnitude interpolée et phase tronquée.

## Description

Cet opcode prend un signal source et le spatialise dans les trois dimensions entourant l'auditeur en réalisant le produit de convolution de la source et de filtres basés sur une fonction de transfert stockée en relation avec la tête (HRTF).

## Syntaxe

```
aleft, aright hrtfmove asrc, kAz, kElev, ifilel, ifiler [, imode, ifade, isr]
```

## Initialisation

### Initialisation

*ifilel* -- fichier des données spectrales HRTF de gauche.

*ifiler* -- fichier des données spectrales HRTF de droite.



#### Note

Des fichiers de données spectrales (basés sur la base de donnée HTRF du MIT) sont disponibles dans trois taux d'échantillonnage : 44.1, 48 et 96 kHz et sont nommés en conséquence. Le *sr* d'entrée et de traitement doit concorder avec celui du fichier de données. Les fichiers doivent se trouver dans le répertoire courant ou le SADIR (voir *Variables d'Environnement*).



#### Note

Les fichiers de données HRTF à utiliser avec *hrtfmove*, *hrtfmove2*, *hrtfstat*, *hrtfearly* et *hrtfverb* ont été mis à jour pour les versions 5.15 et ultérieures de Csound (le code a été mis à jour et il est plus efficace). Les anciens fichiers de données sont maintenant obsolètes.

*imode* -- facultatif, 0, par défaut, pour une phase tronquée, 1 pour une phase minimale.

*ifade* -- facultatif, nombre de tampons de traitement pour le fondu-enchaîné du changement de phase (8 par défaut). L'intervalle autorisé est 1-24. Une faible valeur est recommandée pour les sources complexes (4 ou moins : une valeur plus élevée peut rendre audible le fondu-enchaîné), une valeur plus élevée pour les sources à bande étroite (8 ou plus : une valeur plus faible peut rendre audible l'incohérence due aux changements de phase par le filtre). N'a aucun effet sur le traitement de la phase minimale.



#### Note

Les fondus peuvent parfois se chevaucher (si des trajectoires artificiellement rapides/complexes sont demandées). Dans ce cas, un avertissement est imprimé. Utiliser un fondu-enchaîné plus court ou changer légèrement la trajectoire pour ne pas risquer l'apparition d'incohérences.

*isr* - facultatif, 44.1 kHz par défaut : valeurs autorisées : 44100, 48000 et 96000.



*kAz* -- valeur d'azimut en degrés. Les valeurs positives représentent les positions sur la droite, les valeurs négatives les positions sur la gauche.

*kElev* -- valeur d'élévation en degrés. Les valeurs positives représentent les positions au-dessus de l'horizontale, les valeurs négatives les positions sous l'horizontale (min -40).

Les trajectoires sans artefact définies par l'utilisateur sont rendues possibles par un algorithme basé sur l'interpolation de magnitude spectrale et la troncature de phase. Des fondus-enchaînés sont implémentés pour minimiser/éliminer d'éventuelles incohérences causées par la mise à jour des valeurs de phase. Ces fondus-enchaînés sont réalisés sur des tampons de traitement convolutif dont le nombre peut être défini par l'utilisateur. Les sources complexes peuvent ne nécessiter un fondu-enchaîné que sur un tampon ; les sources à bande étroite peuvent en nécessiter plusieurs. L'opcode offre aussi un traitement basé sur la phase minimale, une méthode plus traditionnelle et complexe. Dans ce mode, les filtres hrtf utilisés sont réduits à des représentation de phase minimale et l'interpolation utilise ensuite la relation entre la magnitude de phase minimale et les spectres de phase. Le délai interaural, qui est inhérent au procédé de phase tronquée, est réintroduit dans le procédé de phase minimale au moyen de lignes à retard variables.

## Exemples

Voici un exemple de l'opcode *hrtfmove*. Il utilise le fichier *hrtfmove.csd* [examples/hrtfmove.csd].

### Exemple 417. Exemple de l'opcode *hrtfmove*.

```
<CsoundSynthesizer>
<CsOptions>
; Select flags here
; realtime audio out
-o dac
; For Non-realtime ouput leave only the line below:
;-o hrtf.wav
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

gasrc init 0

instr 1 ;a plucked string

    kamp = p4
    kcps = cpspch(p5)
    icps = cpspch(p5)

    a1 pluck kamp, kcps, icps, 0, 1

    gasrc = a1

endin

instr 10 ;uses output from instr1 as source

    kaz linseg 0, p3, 720 ;2 full rotations

    aleft,aright hrtfmove gasrc, kaz,0, "hrtf-44100-left.dat", "hrtf-44100-right.dat"

    outs aleft, aright
```

```
endin

</CsInstruments>
<CsScore>

; Play Instrument 1: a simple arpeggio
i1 0 .2 15000 8.00
i1 + .2 15000 8.04
i1 + .2 15000 8.07
i1 + .2 15000 8.11
i1 + .2 15000 9.02
i1 + 1.5 15000 8.11
i1 + 1.5 15000 8.07
i1 + 1.5 15000 8.04
i1 + 1.5 15000 8.00
i1 + 1.5 15000 7.09
i1 + 1.5 15000 8.00

; Play Instrument 10 for 10 seconds.
i10 0 10

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*hrtfmove2, hrtfstat, hrtfer.*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/issue9/newHRTFOpcodes.html>, écrit par Brian Carty.

## Crédits

Auteur : Brian Carty  
Maynooth  
2008

# hrtfmove2

**hrtfmove2** — Génère un signal audio dynamique 3D binaural pour casque en utilisant un modèle de Woodworth de tête sphérique avec précision améliorée de la phase en basse fréquence.

## Description

Cet opcode prend un signal source et le spatialise dans les trois dimensions entourant l'auditeur en utilisant des filtres basés sur une fonction de transfert en relation avec la tête (HRTF).

## Syntaxe

```
aleft, aright hrtfmove2 asrc, kAz, kElev, ifilel, ifiler [,ioverlap, iradius, isr]
```

## Initialisation

*ifilel* -- fichier des données spectrales HRTF de gauche.

*ifiler* -- fichier des données spectrales HRTF de droite.



### Note

Des fichiers de données spectrales (basés sur la base de donnée HTRF du MIT) sont disponibles dans trois taux d'échantillonnage : 44.1, 48 et 96 kHz et sont nommés en conséquence. Le *sr* d'entrée et de traitement doit concorder avec celui du fichier de données. Les fichiers doivent se trouver dans le répertoire courant ou le SADIR (voir *Variables d'Environnement*).



### Note

Les fichiers de données HRTF à utiliser avec *hrtfmove*, *hrtfmove2*, *hrtfstat*, *hrtfearly* et *hrtfverb* ont été mis à jour pour les versions 5.15 et ultérieures de Csound (le code a été mis à jour et il est plus efficace). Les anciens fichiers de données sont maintenant obsolètes.

*ioverlap* -- facultatif, nombre de chevauchements pour le traitement de la TFCT (4 par défaut). Voir la section du manuel sur la TFCT.

*iradius* -- facultatif, rayon de la tête en centimètres utilisé pour le calcul du spectre de phase (9.0 par défaut).

*isr* - facultatif, 44.1 kHz par défaut : valeurs autorisées : 44100, 48000 et 96000.

## Exécution

*asrc* -- Signal source.

*kAz* -- valeur d'azimut en degrés. Les valeurs positives représentent les positions sur la droite, les valeurs négatives les positions sur la gauche.

*kElev* -- valeur d'élévation en degrés. Les valeurs positives représentent les positions au-dessus de l'horizontale, les valeurs négatives les positions sous l'horizontale (min -40).

Les trajectoires sans artefact définies par l'utilisateur sont rendues possibles par un algorithme basé sur l'interpolation de magnitude spectrale et un spectre de phase dérivé basé sur le modèle de tête sphérique de

Woodworth. La précision de l'ensemble de données fourni est augmentée en extrayant et en appliquant au spectre de phase un facteur de pondération dépendant de la fréquence, ce qui conduit à un délai interaural plus précis dans les basses fréquences. On peut contrôler le rayon de la tête pour la dérivation de la phase ce qui donne un niveau simple d'individualisation. La version de l'opcode à source dynamique utilise un algorithme de Transformation de Fourier à Court Terme pour éviter les artefacts causés par les changements des spectres de phase dérivés. Le traitement par TFCT signifie que cet opcode est plus gourmand en ressources que *hrtfmove* qui tronque la phase, mais la phase est constamment actualisée par *hrtfmove2*.

## Exemples

Voici un exemple de l'opcode *hrtfmove2*. Il utilise le fichier *hrtfmove2.csd* [examples/hrtfmove2.csd].

### Exemple 418. Exemple de l'opcode *hrtfmove2*.

```
<CsoundSynthesizer>
<CsOptions>
; Select flags here
; realtime audio out
-o dac
; For Non-realtime output leave only the line below:
; -o hrtf.wav
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

gasrc init 0

instr 1 ;a plucked string

    kamp = p4
    kcps = cpspch(p5)
    icps = cpspch(p5)

    a1 pluck kamp, kcps, icps, 0, 1

    gasrc = a1

endin

instr 10 ;uses output from instr1 as source

    kaz linseg 0, p3, 720 ;2 full rotations

    aleft,aright hrtfmove2 gasrc, kaz,0, "hrtf-44100-left.dat","hrtf-44100-right.dat"

    outs aleft, aright

endin

</CsInstruments>
<CsScore>

; Play Instrument 1: a simple arpeggio
i1 0 .2 15000 8.00
i1 + .2 15000 8.04
i1 + .2 15000 8.07
i1 + .2 15000 8.11
i1 + .2 15000 9.02
```

```
i1 + 1.5 15000 8.11
i1 + 1.5 15000 8.07
i1 + 1.5 15000 8.04
i1 + 1.5 15000 8.00
i1 + 1.5 15000 7.09
i1 + 1.5 15000 8.00

; Play Instrument 10 for 10 seconds.
i10 0 10

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*hrtfmove, hrtfstat, hrtfer.*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/issue9/newHRTFOpcodes.html>, écrit par Brian Carty.

## Crédits

Auteur : Brian Carty  
Maynooth  
2008

# hrtfverb

**hrtfverb** — Une réverbération binaurale à champ de diffusion basée sur un réseau de lignes à retard rétroactives (Feedback Delay Network) dynamique.

## Description

Cet opcode fonctionne de manière autonome comme une réverbération souple et efficace. Il est cependant conçu pour être utilisé avec *hrtfearly* pour obtenir une réverbération spatialement précise avec des trajectoires de la source définissables par l'utilisateur. Il fournit aussi une cohérence inter-oreilles précise.

Un champ réverbérant efficace, fonction de la fréquence, est créé en se basant sur les durées de réverbération souhaitées pour les basses et les hautes fréquences. L'opcode est conçu pour fonctionner avec *hrtfearly*, utilisant idéalement les sorties de ce dernier comme entrées. Cependant, on peut utiliser *hrtfverb* de manière autonome. La stabilité est renforcée.

## Syntaxe

```
aleft, aright, idel hrtfverb asrc, ilowrt60, ihighrt60, ifilel, ifiler [,isr, imfp, iorder]
```

## Initialisation

## Initialisation

## Initialisation

*ilowrt60* - durée de réverbération des basses fréquences.

*ihighrt60* - durée de réverbération des hautes fréquences.

*ifilel* - fichier des données spectrales HRTF de gauche.

*ifiler* - fichier des données spectrales HRTF de droite.



### Note

Des fichiers de données spectrales (basés sur la base de données HRTF du MIT) sont disponibles sous trois différents taux d'échantillonnage : 44.1, 48 et 96 kHz et sont nommés en conséquence. Le *sr* d'entrée et de traitement doit correspondre au *sr* du fichier de données. Les fichiers doivent se trouver dans le répertoire courant ou dans SADIR (voir *Variables d'environnement*).



### Note

Les fichiers de données HRTF à utiliser avec *hrtfmove*, *hrtfmove2*, *hrtfstat*, *hrtfearly* et *hrtfverb* ont été mis à jour pour les versions 5.15 et ultérieures de Csound (le code a été mis à jour et il est plus efficace). Les anciens fichiers de données sont maintenant obsolètes.

*isr* - facultatif, 44.1kHz par défaut. Valeurs acceptées : 44100, 48000 et 96000.

*imfp* - facultatif, chemin libre moyen, celui d'un pièce de taille moyenne par défaut. Si cet opcode est utilisé avec *hrtfearly*, le chemin libre moyen de la pièce peut être utilisé pour calculer le retard approprié pour

la réverbération tardive. Intervalle accepté : le chemin libre moyen de la plus petite pièce autorisée par *hrtfearly* (0.003876) 1.

*iorder* - facultatif, ordre du traitement des premières réflexions. Si cet opcode est utilisé avec *hrtfearly*, l'ordre des premières réflexions peut être utilisé pour calculer le retard pour la réverbération tardive.

*asrc* - Signal source en entrée.

*idel* - Si cet opcode est utilisé avec *hrtfearly*, ce paramètre est le retard approprié pour la réverbération tardive, basé sur la pièce et l'ordre du traitement.

## Exemples

Voici un exemple des opcodes *hrtfearly* et *hrtfreverb*. Il utilise le fichier *hrtfearly.csd* [exemples/hrtfearly.csd].

### Exemple 419. Exemple de l'opcode *hrtfearly*.

```
<CsoundSynthesizer>
<CsOptions>

; Select flags here
; realtime audio out
-o dac
; file output
; -o hrtf.wav

</CsOptions>
<CsInstruments>

nchnls = 2

gasrc init 0 ;global

instr 1 ;a plucked string, distorted and filtered

    iamp = 15000
    icps = cpspch(p4)

    a1 pluck iamp, icps, icps, 0, 1
    adist distort1 a1, 10, .5, 0, 0
    afilt moogvcf2 adist, 8000, .5
    aout linen afilt, 0, p3, .01

    gasrc = gasrc + aout

endin

instr 10 ;uses output from instr1 as source

    ;simple path for source
    kx line 2, p3, 9

    ;early reflections, room default 1
    aearlyl,aearlyr, irt60low, irt60high, imfp hrtfearly gasrc, kx, 5, 1, 5, 1, 1, "hrtf-44100-left.dat",

    ;later reverb, uses outputs from above
    arevl, arevr, idel hrtfreverb gasrc, irt60low, irt60high, "hrtf-44100-left.dat", "hrtf-44100-right.da

    ;delayed and scaled
    alatel delay arevl * .1, idel
    alater delay arevr * .1, idel
```

```
outs aearlyl + alatel, aearlyr + alater

gasrc = 0

endin

</CsInstruments>
<CsScore>

; Play Instrument 1: a simple arpeggio
i1 0 .2 8.00
i1 + .2 8.04
i1 + .2 8.07
i1 + .2 8.11
i1 + .2 9.02
i1 + 1.5 8.11
i1 + 1.5 8.07
i1 + 1.5 8.04
i1 + 1.5 8.00
i1 + 1.5 7.09
i1 + 4 8.00

; Play Instrument 10 for 13 seconds.
i10 0 13

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*hrtfearly hrtfmove, hrtfmove2, hrtfstat, hrtfer.*

## Crédits

Auteur : Brian Carty  
Maynooth  
2011



# hrtfstat

**hrtfstat** — Génère un signal audio statique 3D binaural pour casque en utilisant un modèle de Woodworth de tête sphérique avec précision améliorée de la phase en basse fréquence.

## Description

Cet opcode prend un signal source et le spatialise dans les trois dimensions entourant l'auditeur en utilisant des filtres basés sur une fonction de transfert en relation avec la tête (HRTF). Il produit une sortie statique (les paramètres d'azimut et d'élévation sont de taux-i), car une source statique permet un traitement bien plus efficace que *hrtfmove* et *hrtfmove2*.

## Syntaxe

```
aleft, aright hrtfstat asrc, iAz, iElev, ifilel, ifiler [,iradius, isr]
```

## Initialisation

*iAz* -- valeur d'azimut en degrés. Les valeurs positives représentent les positions sur la droite, les valeurs négatives les positions sur la gauche.

*iElev* -- valeur d'élévation en degrés. Les valeurs positives représentent les positions au-dessus de l'horizontale, les valeurs négatives les positions sous l'horizontale (min -40).

*ifilel* -- fichier des données spectrales HRTF de gauche.

*ifiler* -- fichier des données spectrales HRTF de droite.



### Note

Des fichiers de données spectrales (basés sur la base de donnée HTRF du MIT) sont disponibles dans trois taux d'échantillonnage : 44.1, 48 et 96 kHz et sont nommés en conséquence. Le *sr* d'entrée et de traitement doit concorder avec celui du fichier de données. Les fichiers doivent se trouver dans le répertoire courant ou le SADIR (voir *Variables d'Environnement*).



### Note

Les fichiers de données HRTF à utiliser avec *hrtfmove*, *hrtfmove2*, *hrtfstat*, *hrtfearly* et *hrtfverb* ont été mis à jour pour les versions 5.15 et ultérieures de Csound (le code a été mis à jour et il est plus efficace). Les anciens fichiers de données sont maintenant obsolètes.

*iradius* -- facultatif, rayon de la tête en centimètres utilisé pour le calcul du spectre de phase (9.0 par défaut).

*isr* - facultatif (44.1 kHz par défaut). Les valeurs autorisées sont 44100, 48000 et 96000.

## Exécution

Une spatialisation statique sans artefact définie par l'utilisateur est rendue possible au moyen d'un algorithme basé sur l'interpolation de magnitude spectrale et un spectre de phase dérivé basé sur le modèle de

tête sphérique de Woodworth. La précision de l'ensemble de données fourni est augmentée en extrayant et en appliquant au spectre de phase un facteur de pondération dépendant de la fréquence, ce qui conduit à un délai interaural plus précis dans les basses fréquences. On peut contrôler le rayon de la tête pour la dérivation de la phase ce qui donne un niveau simple d'individualisation. La version à source statique de l'opcode utilise la convolution par chevauchement et addition (le traitement par TFCT n'est pas nécessaire, voir *hrtfmove2*), et elle est ainsi considérablement plus efficace que *hrtfmove2* ou *hrtfmove*, mais elle ne peut pas générer de sources en mouvement.

## Exemples

Voici un exemple de l'opcode *hrtfstat*. Il utilise le fichier *hrtfstat.csd* [examples/hrtfstat.csd].

### Exemple 420. Exemple de l'opcode *hrtfstat*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o hrtfstat.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gasrc init 0

instr 1 ;a plucked string

kamp = p4
kcps = cpspch(p5)
icps = cpspch(p5)
a1 pluck kamp, kcps, icps, 0, 1

gasrc = a1

endin

instr 10;uses output from instr1 as source

aleft,aright hrtfstat gasrc, 90,0, "hrtf-44100-left.dat","hrtf-44100-right.dat"
outs      aleft, aright

clear gasrc
endin

</CsInstruments>
<CsScore>

i1 0 2 .7 8.00 ; Play Instrument 1: a plucked string
i1 .5 2 .7 8.00
i1 1 2 .7 8.00
i1 2 2 .7 7.00

i10 0 12 ; Play Instrument 10 for 2 seconds.
```

```
</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode *hrtfstat*. Il utilise les fichiers *hrtfstat-2.csd* [examples/hrtfstat-2.csd] et *Church.wav* [examples/Church.wav], qui contient un son échantillonné avec boucle.

### Exemple 421. Deuxième exemple de l'opcode *hrtfstat*

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o hrtfstat-2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

iAz = p4
iElev = p5

itim = ftlptim(1)
; transeg a dur ty b dur ty c dur ty d
kamp transeg 0, p3*.1, 0, .9, p3*.3, -3, .5, p3*.3, -2, 0
ain loscil3 kamp, 50, 1
aleft,aright hrtfstat ain, iAz, iElev, "hrtf-44100-left.dat", "hrtf-44100-right.dat"
outs aleft, aright

endin
</CsInstruments>
<CsScore>
f 1 0 0 1 "Church.wav" 0 0 0 ;Csound computes tablesizes

; Azim Elev
i1 0 7 90 0 ;to the right
i1 3 7 -90 -40 ;to the left and below
i1 6 7 180 90 ;behind and up
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*hrtfmove*, *hrtfmove2*, *hrtfer*.

Plus d'information sur cet opcode : <http://www.csoundjournal.com/issue9/newHRTFOpcodes.html>, écrit par Brian Carty.

## Crédits

Auteur : Brian Carty  
Maynooth  
2008

# hsboscil

hsboscil — Un oscillateur qui prend en arguments l'intonation et la brillance.

## Description

Un oscillateur qui prend en arguments l'intonation et la brillance, relativement à une fréquence de base.

## Syntaxe

```
ares hsboscil kamp, ktone, kbrite, ibasfreq, iwfn, ioctfn \  
    [, ioctcnt] [, iphs]
```

## Initialisation

*ibasfreq* -- fréquence de base par rapport à laquelle l'intonation et la brillance sont relatives.

*iwfn* -- table de fonction de la forme d'onde, habituellement une sinus.

*ioctfn* -- table de fonction utilisée pour pondérer les octaves, habituellement quelque chose comme

```
f1 0 1024 -19 1 0.5 270 0.5
```

*ioctcnt* (facultatif) -- nombre d'octaves utilisées pour le mélange de brillance. Doit valoir entre 2 et 10. Par défaut = 3.

*iphs* (facultatif, par défaut = 0) -- phase initiale de l'oscillateur. Si *iphs* = -1, l'initialisation est ignorée.

## Exécution

*kamp* -- amplitude de la note

*ktone* -- paramètre cyclique d'intonation cyclique relatif à *ibasfreq* en octave logarithmique, entre 0 et 1, des valeurs > 1 peuvent être utilisées, et sont réduites en interne à *frac(ktone)*.

*kbrite* -- paramètre de brillance relatif à *ibasfreq*, obtenue en pondérant *ioctcnt* octaves. Il est échelonné de telle manière qu'une valeur de 0 correspond à la valeur originale de *ibasfreq*, 1 correspond à une octave au-dessus de *ibasfreq*, -2 correspond à deux octaves sous *ibasfreq*, etc. *kbrite* peut être fractionnaire.

*hsboscil* prend en arguments l'intonation et la brillance, relativement à une fréquence de base (*ibasfreq*). L'intonation est un paramètre cyclique dans l'octave logarithmique, la brillance est réalisée en mélangeant plusieurs octaves pondérées. Il est utile lorsque l'espace d'intonation est appréhendé dans un concept de coordonnées polaires.

Si *ktone* est une droite et *kbrite* une constante, le résultat produit est le glissando de Risset.

La table de l'oscillateur *iwfn* est toujours lue avec interpolation. Le temps d'exécution est approximativement *ioctcnt* \* *oscili*.

## Exemples

Voici un exemple de l'opcode *hsboscil*. Il utilise le fichier *hsboscil.csd* [examples/hsboscil.csd].

**Exemple 422. Exemple de l'opcode hsboscil.**

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime output leave only the line below:
; -o hsboscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

; synth waveform
giwave ftgen 1, 0, 1024, 10, 1, 1, 1, 1
; blending window
giblend ftgen 2, 0, 1024, -19, 1, 0.5, 270, 0.5

instr 1 ; produces Risset's glissando.

    kamp = .4
    kbrite = 0.3
    ibasfreq = 200
    ioctcnt = 5

    ; Change ktone linearly from 0 to 1,
    ; over the period defined by p3.
    ktone line 0, p3, 1

asig hsboscil kamp, ktone, kbrite, ibasfreq, giwave, giblend, ioctcnt
    outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

Voici un exemple de l'opcode hsboscil dans un instrument MIDI. Il utilise le fichier *hsboscil\_midi.csd* [examples/hsboscil\_midi.csd].

**Exemple 423. Exemple de l'opcode hsboscil dans un instrument MIDI.**

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out
-odac -M0 ;;realtime audio out and realtime MIDI in
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; -o hsboscil_midi.wav -W ;; for file output any platform
```

```
</CsOptions>
<CsInstruments>

  sr = 44100
  ksmps = 32
  nchnls = 2
  0dbfs = 1

  ; synth waveform
  giwave ftgen 1, 0, 1024, 10, 1, 1, 1, 1
  ; blending window
  giblend ftgen 2, 0, 1024, -19, 1, 0.5, 270, 0.5

  instr 1

  ibase = cpsoct(6)
  ioctcnt = 5

  ; all octaves sound alike.
  itona octmidi
  ; velocity is mapped to brightness
  ibrite ampmidi 4

  ; Map an exponential envelope for the amplitude.
  kenv expon .8, 1, .01
  asig hsboscil kenv, itona, ibrite, ibase, giwave, giblend, ioctcnt
  outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 30 ; play for 30 seconds
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Peter Neubäcker  
Munich, Allemagne  
Août 1999

Nouveau dans la version 3.58 de Csound

# hvs1

*hvs1* — Synthèse Hyper Vectorielle (SHV) à une dimension contrôlée par une variable de taux-k mise à jour en externe.

## Description

*hvs1* permet la synthèse Hyper Vectorielle (SHV) à une dimension contrôlée par une variable de taux-k mise à jour en externe.

## Syntaxe

```
hvs1 kx, inumParms, inumPointsX, iOutTab, iPositionsTab, iSnapTab [, iConfigTab]
```

## Initialisation

*inumParms* - nombre de paramètres contrôlés par la SHV. Chaque instantané de la SHV est composé de *inumParms* éléments.

*inumPointsX* - nombre de points dont est composée chaque dimension du cube de la SHV (ou du carré dans le cas de la SHV à deux dimensions, ou du segment de droite dans le cas de la SHV à une dimension).

*iOutTab* - numéro de la table recevant l'ensemble des valeurs instantanées des paramètres de sortie de la SHV. Le nombre total de paramètres est défini par l'argument *inumParms*.

*iPositionsTab* – une table remplie avec les positions individuelles des instantanés dans la matrice de SHV (voir ci-dessous pour plus d'information).

*iSnapTab* – une table remplie avec tous les instantanés. Chaque instantané se compose d'un ensemble de valeurs des paramètres. Le nombre d'éléments contenus dans chaque instantané est spécifié par l'argument *inumParms*. L'ensemble des éléments de chaque instantané suit (et est adjacent à) l'instantané précédent dans la table. Ainsi la taille totale de cette table doit être  $\geq$  à *inumParms* multiplié par le nombre d'instantanés que l'on veut mémoriser pour la SVH.

*iConfigTab* – (facultatif) une table contenant le comportement de la SHV pour chaque paramètre. Si *iConfigTab* vaut zéro (par défaut), cet argument est ignoré, ce qui signifie que chaque paramètre est interpolé linéairement par la SHV. Si *iConfigTab* est différent de zéro, il doit faire référence à une table existante dont le contenu fait référence à son tour à un genre particulier d'interpolation. Dans cette table, une valeur de -1 indique que le paramètre correspondant reste inchangé (ignoré) par la SHV ; une valeur de zéro indique que le paramètre correspondant est traité par interpolation linéaire ; tout autre valeur doit être un nombre entier indiquant une table existante remplie avec une forme qui détermine le genre d'interpolation spéciale à utiliser (interpolation basée sur une table).

## Exécution

*kx* - ce sont des variables modifiées de l'extérieur, qui contrôlent le mouvement du pointeur dans la matrice cubique de SHV (ou carrée ou en ligne dans le cas où les matrices de SHV ont moins de 3 dimensions). Les valeurs de ces arguments d'entrée doivent être comprises entre 0 et 1.

La Synthèse Hyper Vectorielle est une technique qui permet de contrôler un ensemble immense de paramètres en utilisant une approche simple et globale. Les concepts clé de la SHV sont :

L'ensemble des paramètres de SHV, dont le nombre est fixé et défini par l'argument *inumParms*. Durant l'exécution de la SHV, tous ces paramètres sont variables et peuvent être appliqués à n'importe quelle technique de synthèse sonore, de même qu'à n'importe quel contrôle global de composition algorithmique ou de tout autre niveau. L'utilisateur doit définir au préalable plusieurs ensembles de valeurs fixes pour chaque paramètre de la SHV, chaque ensemble correspondant à une configuration de synthèse déterminée. Chaque ensemble de valeurs est appelé un instantané et peut être considéré comme les coordonnées d'un saut dans un espace multi-dimensionnel. La SHV consiste à faire évoluer un point dans cet espace multi-dimensionnel (en utilisant un pointeur de mouvement spécial, voie ci-dessous), selon les points et les limites définis par les instantanés. On peut fixer le nombre de paramètres de la SHV (chaque paramètre suivant une dimension de l'espace multi-dimensionnel), même un nombre très important, la limite ne dépendant que de la puissance de calcul (et de la mémoire) de votre ordinateur et de la complexité de la synthèse sonore utilisée.

Le cube de SHV (ou le carré ou le segment). C'est la matrice (à 3, 2 ou 1 dimensions, en fonction de l'opcode *hvs* utilisé) des points d'appui ou pivots de la SHV. Le nombre total de points pivots dépend de la valeur des arguments *inumPointsX*, *inumPointsY* et *inumPointsZ*. Dans le cas d'une matrice de SHV à 3 dimensions on peut définir, par exemple, 3 points pour la dimension X, 5 pour la dimension Y et 2 pour la dimension Z. Dans ce cas le nombre total de points pivots est  $3 * 5 * 2 = 30$ . Avec cet ensemble de points pivots, le cube est divisé en zones cubiques plus petites, chacune étant délimitée par huit points. Chaque point est numéroté. La numérotation de ces points se fait de la manière suivante : numéro zéro pour le premier point, numéro un pour le second, et ainsi de suite. En supposant que l'on utilise un cube de SHV à 3 dimensions ayant le nombre de points mentionné ci-dessus (c'est-à-dire 3, 5 et 2 respectivement pour les axes X, Y et Z), le premier point (point zéro) est le coin supérieur gauche de la face avant du cube, si l'on regarde le plan XY du cube. Le second point est le centre de l'arête supérieure de la face avant du cube, ainsi de suite. On peut se référer à la figure ci-dessous pour comprendre comment la numérotation des points pivots est réalisée :

Pour la SHV à 2 dimensions, c'est la même chose, en omettant seulement la face arrière du cube, si bien que chaque zone est délimitée par 4 points pivots au lieu de 8. Pour la SHV à 1 dimension, tout devient plus simple car il n'y a qu'un segment de droite sur lequel les points pivots sont répartis de gauche à droite. Chaque point est couplé à un instantané.

L'ordre des instantanés, comme il est stocké dans la table *iSnapTab*, peut suivre ou pas l'ordre des numéros des points pivots. En fait il est possible de modifier cet ordre avec la table *iPositionsTab*, qui repositionne chaque instantané en fonction des points pivots. La table *iPositionsTab* est constituée des positions des instantanés (contenus dans la table *iSnapTab*) sur la grille à deux dimensions. Chaque élément successif est un pointeur représentant la position dans la table *iSnapTab*. Par exemple, dans une matrice de SHV à 2 dimensions comme la suivante (ayant dans ce cas *inumPointsX* = 3 et *inumPointsY* = 5) :

**Tableau 9.**

5	7	1
3	4	9
6	2	0
4	1	3
8	2	7

Ces numéros (à stocker dans la table *iSnapTab* en utilisant, par exemple, le générateur de fonction *GEN02*), représentent la position des instantanés dans la grille (dans ce cas une matrice 3x5). Ainsi, le premier élément 5 à l'indice zéro et représente le sixième (l'élément zéro est le premier) instantané contenu dans la table *iSnapTab*, le second élément 7 représente le huitième élément de *iSnapTab*, ainsi de suite. En résumé, les sommets de chaque zone (une zone cubique est délimitée par 8 sommets, une zone carrée par



4 sommets et une zone linéaire par 2 points) sont liés à un instantané donné, dont le numéro est transformé par la table *iSnapTab*.

Les valeurs de sortie de la SHV sont influencées par le mouvement du pointeur, un point dont la position dans le cube de SHV (ou le carré ou le segment) est déterminée par les arguments *kx*, *ky* et *kz*. Les valeurs de ces arguments, qui doivent être comprises entre 0 et 1, sont fixées extérieurement par l'utilisateur. Les valeurs de sortie, dont le nombre est égal à l'argument *inumParms*, sont stockées dans la table *iOutTab*, laquelle doit avoir été allouée auparavant par l'utilisateur et doit avoir une taille d'au moins *inumParms*. De quelle manière le mouvement du pointeur influence-t-il la sortie ? Lorsque le pointeur tombe dans une zone cubique déterminée, délimitée par exemple par 8 sommets (ou points pivots), nous supposons que chaque sommet est associé à un instantané différent (c'est-à-dire un ensemble de *inumParms* valeurs), la sortie sera la moyenne pondérée des 8 sommets, calculée en fonction de la distance du pointeur à chacun des 8 sommets. Dans le comportement par défaut, lorsque l'argument *iConfigTab* n'est pas défini, la sortie exacte est calculée en utilisant une interpolation linéaire qui est appliquée à chaque paramètre de la SHV. Cependant, il est possible de modifier ce comportement en donnant à l'argument *iConfigTab* le numéro d'une table dont le contenu peut affecter un ou plusieurs paramètres de la SHV. Les éléments de la table *iConfigTab* sont associés à chaque paramètre de SHV et leurs valeurs affectent la sortie de la SHV de la manière suivante :

- Si *iConfigTab* vaut -1, la sortie correspondante est ignorée, c'est-à-dire que l'élément n'est pas calculé, laissant la valeur de l'élément correspondant dans la table *iOutTab* inchangée ;
- Si *iConfigTab* est égal à zéro, la sortie de la SHV normale est calculée (en utilisant la moyenne pondérée des sommets les plus proches de la zone dans laquelle le pointeur mobile est tombé) ;
- Si *iConfigTab* est égal à un nombre entier > zéro, le contenu d'une table ayant ce numéro est utilisé comme la forme d'une interpolation basée sur cette table.

## Exemples

Voici un exemple de l'opcode *hvs1*. Il utilise le fichier *hvs1.csd* [exemples/hvs1.csd].

### Exemple 424. Exemple de l'opcode *hvs1*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=44100
ksmps=100
nchnls=2
0dbfs = 1

; Example by Gabriel Maldonado and Andres Cabrera

ginumLinesX  init 16
ginumParms   init 3

giOutTab  ftgen 5,0,8, -2,      0
giPosTab  ftgen 6,0,32, -2,     3,2,1,0,4,5,6,7,8,9,10, 11, 15, 14, 13, 12
```

```

giSnapTab ftgen 8,0,64, -2,      1,1,1,   2,0,0,   3,2,0,   2,2,2,   5,2,1,   2,3,4,   6,1,7,   0,0,0, \
                                1,3,5,   3,4,4,   1,5,8,   1,1,5,   4,3,2,   3,4,5,   7,6,5,   7,8,9

tb0_init giOutTab

      FLpanel "hsv1",500,100,10,10,0
gk1,ih1 FLslider "X", 0,1, 0,5, -1, 400,30, 50,20
      FLpanel_end
      FLrun

      instr 1
;      kx,   inumParms,   inumPointsX,   iOutTab,   iPosTab,   iSnapTab [, iConfigTab]
      hvs1   gk1,   ginumParms,   ginumLinesX,   giOutTab,   giPosTab,   giSnapTab   [, iConfigTab]

k0 init 0
k1 init 1
k2 init 2

printk2 tb0(k0)
printk2 tb0(k1), 10
printk2 tb0(k2), 20

aosc1 oscil tb0(k0)/20, tb0(k1)*100 + 200, 1
aosc2 oscil tb0(k1)/20, tb0(k2)*100 + 200, 1
aosc3 oscil tb0(k2)/20, tb0(k0)*100 + 200, 1
aosc4 oscil tb0(k1)/20, tb0(k0)*100 + 200, 1
aosc5 oscil tb0(k2)/20, tb0(k1)*100 + 200, 1
aosc6 oscil tb0(k0)/20, tb0(k2)*100 + 200, 1

outs aosc1 + aosc2 + aosc3, aosc4 + aosc5 + aosc6
endin

</CsInstruments>
<CsScore>

f1 0 1024 10 1
f0 3600
i1 0 3600

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*hvs2, hvs3, vphaseseg*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# hvs2

**hvs2** — Synthèse Hyper Vectorielle (SHV) à deux dimensions contrôlée par des variables de taux-k mises à jour en externe.

## Description

*hvs2* permet la synthèse Hyper Vectorielle (SHV) à deux dimensions contrôlée par des variables de taux-k mises à jour en externe.

## Syntaxe

**hvs2** *kx*, *ky*, *inumParms*, *inumPointsX*, *inumPointsY*, *iOutTab*, *iPositionsTab*, *iSnapTab* [, *iConfigTab*]

## Initialisation

*inumParms* - nombre de paramètres contrôlés par la SHV. Chaque instantané de la SHV est composé de *inumParms* éléments.

*inumPointsX*, *inumPointsY* - nombre de points dont est composée chaque dimension du cube de la SHV (ou du carré dans le cas de la SHV à deux dimensions, ou du segment de droite dans le cas de la SHV à une dimension).

*iOutTab* - numéro de la table recevant l'ensemble des valeurs instantanées des paramètres de sortie de la SHV. Le nombre total de paramètres est défini par l'argument *inumParms*.

*iPositionsTab* – une table remplie avec les positions individuelles des instantanés dans la matrice de SHV (voir ci-dessous pour plus d'information).

*iSnapTab* – une table remplie avec tous les instantanés. Chaque instantané se compose d'un ensemble de valeurs des paramètres. Le nombre d'éléments contenus dans chaque instantané est spécifié par l'argument *inumParms*. L'ensemble des éléments de chaque instantané suit (et est adjacent à) l'instantané précédent dans la table. Ainsi la taille totale de cette table doit être  $\geq$  à *inumParms* multiplié par le nombre d'instantanés que l'on veut mémoriser pour la SVH.

*iConfigTab* – (facultatif) une table contenant le comportement de la SHV pour chaque paramètre. Si *iConfigTab* vaut zéro (par défaut), cet argument est ignoré, ce qui signifie que chaque paramètre est interpolé linéairement par la SHV. Si *iConfigTab* est différent de zéro, il doit faire référence à une table existante dont le contenu fait référence à son tour à un genre particulier d'interpolation. Dans cette table, une valeur de -1 indique que le paramètre correspondant reste inchangé (ignoré) par la SHV ; une valeur de zéro indique que le paramètre correspondant est traité par interpolation linéaire ; tout autre valeur doit être un nombre entier indiquant une table existante remplie avec une forme qui détermine le genre d'interpolation spéciale à utiliser (interpolation basée sur une table).

## Exécution

*kx*, *ky* - ce sont des variables modifiées de l'extérieur, qui contrôlent le mouvement du pointeur dans la matrice cubique de SHV (ou carrée ou en ligne dans le cas où les matrices de SHV ont moins de 3 dimensions). Les valeurs de ces arguments d'entrée doivent être comprises entre 0 et 1.

La Synthèse Hyper Vectorielle est une technique qui permet de contrôler un ensemble immense de paramètres en utilisant une approche simple et globale. Les concepts clé de la SHV sont :

L'ensemble des paramètres de SHV, dont le nombre est fixé et défini par l'argument *inumParms*. Durant l'exécution de la SHV, tous ces paramètres sont variables et peuvent être appliqués à n'importe quelle technique de synthèse sonore, de même qu'à n'importe quel contrôle global de composition algorithmique ou de tout autre niveau. L'utilisateur doit définir au préalable plusieurs ensembles de valeurs fixes pour chaque paramètre de la SHV, chaque ensemble correspondant à une configuration de synthèse déterminée. Chaque ensemble de valeurs est appelé un instantané et peut être considéré comme les coordonnées d'un saut dans un espace multi-dimensionnel. La SHV consiste à faire évoluer un point dans cet espace multi-dimensionnel (en utilisant un pointeur de mouvement spécial, voie ci-dessous), selon les points et les limites définis par les instantanés. On peut fixer le nombre de paramètres de la SHV (chaque paramètre suivant une dimension de l'espace multi-dimensionnel), même un nombre très important, la limite ne dépendant que de la puissance de calcul (et de la mémoire) de votre ordinateur et de la complexité de la synthèse sonore utilisée.

Le cube de SHV (ou le carré ou le segment). C'est la matrice (à 3, 2 ou 1 dimensions, en fonction de l'opcode *hvs* utilisé) des points d'appui ou pivots de la SHV. Le nombre total de points pivots dépend de la valeur des arguments *inumPointsX*, *inumPointsY* et *inumPointsZ*. Dans le cas d'une matrice de SHV à 3 dimensions on peut définir, par exemple, 3 points pour la dimension X, 5 pour la dimension Y et 2 pour la dimension Z. Dans ce cas le nombre total de points pivots est  $3 * 5 * 2 = 30$ . Avec cet ensemble de points pivots, le cube est divisé en zones cubiques plus petites, chacune étant délimitée par huit points. Chaque point est numéroté. La numérotation de ces points se fait de la manière suivante : numéro zéro pour le premier point, numéro un pour le second, et ainsi de suite. En supposant que l'on utilise un cube de SHV à 3 dimensions ayant le nombre de points mentionné ci-dessus (c'est-à-dire 3, 5 et 2 respectivement pour les axes X, Y et Z), le premier point (point zéro) est le coin supérieur gauche de la face avant du cube, si l'on regarde le plan XY du cube. Le second point est le centre de l'arête supérieure de la face avant du cube, ainsi de suite. On peut se référer à la figure ci-dessous pour comprendre comment la numérotation des points pivots est réalisée :

Pour la SHV à 2 dimensions, c'est la même chose, en omettant seulement la face arrière du cube, si bien que chaque zone est délimitée par 4 points pivots au lieu de 8. Pour la SHV à 1 dimension, tout devient plus simple car il n'y a qu'un segment de droite sur lequel les points pivots sont répartis de gauche à droite. Chaque point est couplé à un instantané.

L'ordre des instantanés, comme il est stocké dans la table *iSnapTab*, peut suivre ou pas l'ordre des numéros des points pivots. En fait il est possible de modifier cet ordre avec la table *iPositionsTab*, qui repositionne chaque instantané en fonction des points pivots. La table *iPositionsTab* est constituée des positions des instantanés (contenus dans la table *iSnapTab*) sur la grille à deux dimensions. Chaque élément successif est un pointeur représentant la position dans la table *iSnapTab*. Par exemple, dans une matrice de SHV à 2 dimensions comme la suivante (ayant dans ce cas *inumPointsX* = 3 et *inumPointsY* = 5) :

**Tableau 10.**

5	7	1
3	4	9
6	2	0
4	1	3
8	2	7

Ces numéros (à stocker dans la table *iSnapTab* en utilisant, par exemple, le générateur de fonction *GEN02*), représentent la position des instantanés dans la grille (dans ce cas une matrice 3x5). Ainsi, le premier élément 5 a l'indice zéro et représente le sixième (l'élément zéro est le premier) instantané contenu dans la table *iSnapTab*, le second élément 7 représente le huitième élément de *iSnapTab*, ainsi de suite. En résumé, les sommets de chaque zone (une zone cubique est délimitée par 8 sommets, une zone carrée par 4 sommets et une zone linéaire par 2 points) sont liés à un instantané donné, dont le numéro est transformé par la table *iSnapTab*.

Les valeurs de sortie de la SHV sont influencées par le mouvement du pointeur, un point dont la position dans le cube de SHV (ou le carré ou le segment) est déterminée par les arguments  $k_x$ ,  $k_y$  et  $k_z$ . Les valeurs de ces arguments, qui doivent être comprises entre 0 et 1, sont fixées extérieurement par l'utilisateur. Les valeurs de sortie, dont le nombre est égal à l'argument *inumParms*, sont stockées dans la table *iOutTab*, laquelle doit avoir été allouée auparavant par l'utilisateur et doit avoir une taille d'au moins *inumParms*. De quelle manière le mouvement du pointeur influence-t-il la sortie ? Lorsque le pointeur tombe dans une zone cubique déterminée, délimitée par exemple par 8 sommets (ou points pivots), nous supposons que chaque sommet est associé à un instantané différent (c'est-à-dire un ensemble de *inumParms* valeurs), la sortie sera la moyenne pondérée des 8 sommets, calculée en fonction de la distance du pointeur à chacun des 8 sommets. Dans le comportement par défaut, lorsque l'argument *iConfigTab* n'est pas défini, la sortie exacte est calculée en utilisant une interpolation linéaire qui est appliquée à chaque paramètre de la SHV. Cependant, il est possible de modifier ce comportement en donnant à l'argument *iConfigTab* le numéro d'une table dont le contenu peut affecter un ou plusieurs paramètres de la SHV. Les éléments de la table *iConfigTab* sont associés à chaque paramètre de SHV et leurs valeurs affectent la sortie de la SHV de la manière suivante :

- Si *iConfigTab* vaut -1, la sortie correspondante est ignorée, c'est-à-dire que l'élément n'est pas calculé, laissant la valeur de l'élément correspondant dans la table *iOutTab* inchangée ;
- Si *iConfigTab* est égal à zéro, la sortie de la SHV normale est calculée (en utilisant la moyenne pondérée des sommets les plus proches de la zone dans laquelle le pointeur mobile est tombé) ;
- Si *iConfigTab* est égal à un nombre entier > zéro, le contenu d'une table ayant ce numéro est utilisé comme la forme d'une interpolation basée sur cette table.

## Exemples

Voici un exemple de l'opcode hvs2. Il utilise le fichier *hvs2.csd* [examples/hvs2.csd].

### Exemple 425. Exemple de l'opcode hvs2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr=44100
ksmps=100
nchnls=2

0dbfs = 1

ginumLinesX init 4
ginumLinesY init 4
ginumParms init 3

giOutTab ftgen 5,0,8, -2,      0
giPosTab ftgen 6,0,32, -2,    3,2,1,0,4,5,6,7,8,9,10, 11, 15, 14, 13, 12
giSnapTab ftgen 8,0,64, -2,   1,1,1,  2,0,0,  3,2,0,  2,2,2,  5,2,1,  2,3,4,  6,1,7,  0,0,0, \
                             1,3,5,   3,4,4,  1,5,8,  1,1,5,  4,3,2,  3,4,5,  7,6,5,  7,8,9

tb0_init giOutTab

FLpanel "Prova HVS2",600,400,10,100,0
```

```

gk1,    gk2,    ih1, ih2  FLjoy    "HVS controller XY", 0,    1,    1,    0,    0,    0,    -1,

; *ihandle,
; *numlinesX,    *numlinesY, *iwidth, *iheight, *ix, *iy,*image;
gihandle FLhvsBox ginumLinesX,    ginumLinesY,    300,    300,    300,    50, 1

    FLpanel_end
    FLrun

instr 1

; Smooth control signals to avoid clicks
kx portk gk1, 0.02
ky portk gk2, 0.02

;
    kx, ky, inumParms, inumlinesX, inumlinesY, iOutTab, iPosTab, iSnapTab [, iConfigT
hvs2 kx, ky, ginumParms, ginumLinesX, ginumLinesY, giOutTab, giPosTab, giSnapTab ;, iConfigT

;
    *kx, *ky, *ihandle;
    FLhvsBoxSetValue gk1, gk2, gihandle

k0 init 0
k1 init 1
k2 init 2

printk2 tb0(k0)
printk2 tb0(k1), 10
printk2 tb0(k2), 20

    kris init 0.003
    kdur init 0.02
    kdec init 0.007

; Make parameters of synthesis depend on the table values produced by hvs
ares1 fof 0.2, tb0(k0)*100 + 50, tb0(k1)*100 + 200, 0, tb0(k2) * 10 + 50, 0.003, 0.02, 0.007, 20, \
    1, 2, p3
ares2 fof 0.2, tb0(k1)*100 + 50, tb0(k2)*100 + 200, 0, tb0(k0) * 10 + 50, 0.003, 0.02, 0.007, 20, \
    1, 2, p3

outs ares1, ares2
endin

</CsInstruments>
<CsScore>
f 1 0 1024 10 1 ;Sine wave
f 2 0 1024 19 0.5 0.5 270 0.5 ;Grain envelope table

f0 3600

i1 0 3600

</CsScore>
</CsSoundSynthesizer>

```

Voici un second exemple de l'opcode hvs2. Il utilise le fichier *hvs2-2.csd* [examples/hvs2-2.csd].

## Exemple 426. Second exemple de l'opcode hvs2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform

```

```

; Audio out   Audio in
-odac         -iadc   ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o hvs2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=48000
ksmps=100
nchnls=2

; Example by James Hearon 2008
; Edited by Andres Cabrera

ginumPointsX init      16
ginumPointsY init      16
ginumParms   init      3

;Generate 9 tables with arbitrary points
gitmp ftgen 100, 0, 16, -2, 70, 260, 390, 180, 200, 300, 980, 126, \
          330, 860, 580, 467, 220, 399, 1026, 1500
gitmp ftgen 200, 0, 16, -2, 100, 200, 300, 140, 600, 700, 880, 126, \
          330, 560, 780, 167, 220, 999, 1026, 1500
gitmp ftgen 300, 0, 16, -2, 400, 200, 300, 540, 600, 700, 880, 126, \
          330, 160, 780, 167, 820, 999, 1026, 1500
gitmp ftgen 400, 0, 16, -2, 100, 200, 800, 640, 600, 300, 880, 126, \
          330, 660, 780, 167, 220, 999, 1026, 1500
gitmp ftgen 500, 0, 16, -2, 200, 200, 360, 440, 600, 700, 880, 126, \
          330, 560, 380, 167, 220, 499, 1026, 1500
gitmp ftgen 600, 0, 16, -2, 100, 600, 300, 840, 600, 700, 880, 126, \
          330, 260, 980, 367, 120, 399, 1026, 1500
gitmp ftgen 700, 0, 16, -2, 100, 200, 300, 340, 200, 500, 380, 126, \
          330, 860, 780, 867, 120, 999, 1026, 1500
gitmp ftgen 800, 0, 16, -2, 100, 600, 300, 240, 200, 700, 880, 126, \
          130, 560, 980, 167, 220, 499, 1026, 1500
gitmp ftgen 900, 0, 16, -2, 100, 800, 200, 140, 600, 700, 680, 126, \
          330, 560, 780, 167, 120, 299, 1026, 1500

giOutTab ftgen 5,0,8, -2, 0
giPosTab ftgen 6,0,32, -2, 0,1,2,3,4,5,6,7,8,9,10, 11, 15, 14, 13, 12
giSnapTab ftgen 8,0,64, -2, 1,1,1, 2,0,0, 3,2,0, 2,2,2, \
          5,2,1, 2,3,4, 6,1,7, 0,0,0, 1,3,5, 3,4,4, 1,5,8, 1,1,5, \
          4,3,2, 3,4,5, 7,6,5, 7,8,9

tb0_init      giOutTab

      FLpanel "hsv2",440,100,10,10,0
gk1,ih1 FLslider "X", 0,1, 0, 5, -1, 400,20, 20,10
gk2, ih2 FLslider "Y", 0, 1, 0, 5, -1, 400, 20, 20, 50
      FLpanel_end

      FLpanel "hvsBox",280,280,500,1000,0
;ihandle FLhvsBox inumlinesX, inumlinesY, iwidth, iheight, ix, iy [, image]
gih1 FLhvsBox 16, 16, 250, 250, 10, 1
      FLpanel_end
      FLrun

      instr 1
      FLhvsBoxSetValue gk1, gk2, gih1

hvs2      gk1,gk2,  ginumParms, ginumPointsX, ginumPointsY, giOutTab, giPosTab, giSnapTab  ;, iConfigTab

k0      init      0
k1      init      1
k2      init      2
kspeed  init      0

```

```

kspeed = int((tb0(k2)) + 1)*.10

kenv oscil 25000, kspeed*16, 10

k1 phasor kspeed ;slow phasor: 200 sec.
kpch tableikt k1 * 16, int((tb0(k1)) +1)*100 ;scale phasor * length
a1 oscilikt kenv, kpch, int(tb0(k0)) +1000;scale pitch slightly
ahp butterlp a1, 2500
outs ahp, ahp

    endin

</CsInstruments>
<CsScore>

f 10 0 1024 20 5 ;use of windowing function
f1000 0 1024 10 .33 .25 .5
f1001 0 1024 10 1
f1002 0 1024 10 .5 .25 .05
f1003 0 1024 10 .05 .10 .3 .5 1
f1004 0 1024 10 1 .5 .25 .125 .625
f1005 0 1024 10 .33 .44 .55 .66
f1006 0 1024 10 1 1 1 1 1
f1007 0 1024 10 .05 .25 .05 .25 .05 1

f0 3600
i1 0 3600

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*hvs1, hvs3, vphaseseg*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06



# hvs3

**hvs3** — Synthèse Hyper Vectorielle (SHV) à trois dimensions contrôlée par des variables de taux-k mises à jour en externe.

## Description

*hvs3* permet la synthèse Hyper Vectorielle (SHV) à trois dimensions contrôlée par des variables de taux-k mises à jour en externe.

## Syntaxe

**hvs3** *kx*, *ky*, *kz*, *inumParms*, *inumPointsX*, *inumPointsY*, *inumPointsZ*, *iOutTab*, *iPositionsTab*, *iSnapTab* [,

## Initialisation

*inumParms* - nombre de paramètres contrôlés par la SHV. Chaque instantané de la SHV est composé de *inumParms* éléments.

*inumPointsX*, *inumPointsY*, *inumPointsZ* - nombre de points dont est composée chaque dimension du cube de la SHV (ou du carré dans le cas de la SHV à deux dimensions, ou du segment de droite dans le cas de la SHV à une dimension).

*iOutTab* - numéro de la table recevant l'ensemble des valeurs instantanées des paramètres de sortie de la SHV. Le nombre total de paramètres est défini par l'argument *inumParms*.

*iPositionsTab* – une table remplie avec les positions individuelles des instantanés dans la matrice de SHV (voir ci-dessous pour plus d'information).

*iSnapTab* – une table remplie avec tous les instantanés. Chaque instantané se compose d'un ensemble de valeurs des paramètres. Le nombre d'éléments contenus dans chaque instantané est spécifié par l'argument *inumParms*. L'ensemble des éléments de chaque instantané suit (et est adjacent à) l'instantané précédent dans la table. Ainsi la taille totale de cette table doit être  $\geq$  à *inumParms* multiplié par le nombre d'instantanés que l'on veut mémoriser pour la SVH.

*iConfigTab* – (facultatif) une table contenant le comportement de la SHV pour chaque paramètre. Si *iConfigTab* vaut zéro (par défaut), cet argument est ignoré, ce qui signifie que chaque paramètre est interpolé linéairement par la SHV. Si *iConfigTab* est différent de zéro, il doit faire référence à une table existante dont le contenu fait référence à son tour à un genre particulier d'interpolation. Dans cette table, une valeur de -1 indique que le paramètre correspondant reste inchangé (ignoré) par la SHV ; une valeur de zéro indique que le paramètre correspondant est traité par interpolation linéaire ; tout autre valeur doit être un nombre entier indiquant une table existante remplie avec une forme qui détermine le genre d'interpolation spéciale à utiliser (interpolation basée sur une table).

## Exécution

*kx*, *ky*, *kz* - ce sont des variables modifiées de l'extérieur, qui contrôlent le mouvement du pointeur dans la matrice cubique de SHV (ou carrée ou en ligne dans le cas où les matrices de SHV ont moins de 3 dimensions). Les valeurs de ces arguments d'entrée doivent être comprises entre 0 et 1.

La Synthèse Hyper Vectorielle est une technique qui permet de contrôler un ensemble immense de paramètres en utilisant une approche simple et globale. Les concepts clé de la SHV sont :

L'ensemble des paramètres de SHV, dont le nombre est fixé et défini par l'argument *inumParms*. Durant l'exécution de la SHV, tous ces paramètres sont variables et peuvent être appliqués à n'importe quelle technique de synthèse sonore, de même qu'à n'importe quel contrôle global de composition algorithmique ou de tout autre niveau. L'utilisateur doit définir au préalable plusieurs ensembles de valeurs fixes pour chaque paramètre de la SHV, chaque ensemble correspondant à une configuration de synthèse déterminée. Chaque ensemble de valeurs est appelé un instantané et peut être considéré comme les coordonnées d'un saut dans un espace multi-dimensionnel. La SHV consiste à faire évoluer un point dans cet espace multi-dimensionnel (en utilisant un pointeur de mouvement spécial, voie ci-dessous), selon les points et les limites définis par les instantanés. On peut fixer le nombre de paramètres de la SHV (chaque paramètre suivant une dimension de l'espace multi-dimensionnel), même un nombre très important, la limite ne dépendant que de la puissance de calcul (et de la mémoire) de votre ordinateur et de la complexité de la synthèse sonore utilisée.

Le cube de SHV (ou le carré ou le segment). C'est la matrice (à 3, 2 ou 1 dimensions, en fonction de l'opcode *hvs* utilisé) des points d'appui ou pivots de la SHV. Le nombre total de points pivots dépend de la valeur des arguments *inumPointsX*, *inumPointsY* et *inumPointsZ*. Dans le cas d'une matrice de SHV à 3 dimensions on peut définir, par exemple, 3 points pour la dimension X, 5 pour la dimension Y et 2 pour la dimension Z. Dans ce cas le nombre total de points pivots est  $3 * 5 * 2 = 30$ . Avec cet ensemble de points pivots, le cube est divisé en zones cubiques plus petites, chacune étant délimitée par huit points. Chaque point est numéroté. La numérotation de ces points se fait de la manière suivante : numéro zéro pour le premier point, numéro un pour le second, et ainsi de suite. En supposant que l'on utilise un cube de SHV à 3 dimensions ayant le nombre de points mentionné ci-dessus (c'est-à-dire 3, 5 et 2 respectivement pour les axes X, Y et Z), le premier point (point zéro) est le coin supérieur gauche de la face avant du cube, si l'on regarde le plan XY du cube. Le second point est le centre de l'arête supérieure de la face avant du cube, ainsi de suite. On peut se référer à la figure ci-dessous pour comprendre comment la numérotation des points pivots est réalisée :

Pour la SHV à 2 dimensions, c'est la même chose, en omettant seulement la face arrière du cube, si bien que chaque zone est délimitée par 4 points pivots au lieu de 8. Pour la SHV à 1 dimension, tout devient plus simple car il n'y a qu'un segment de droite sur lequel les points pivots sont répartis de gauche à droite. Chaque point est couplé à un instantané.

L'ordre des instantanés, comme il est stocké dans la table *iSnapTab*, peut suivre ou pas l'ordre des numéros des points pivots. En fait il est possible de modifier cet ordre avec la table *iPositionsTab*, qui repositionne chaque instantané en fonction des points pivots. La table *iPositionsTab* est constituée des positions des instantanés (contenus dans la table *iSnapTab*) sur la grille à deux dimensions. Chaque élément successif est un pointeur représentant la position dans la table *iSnapTab*. Par exemple, dans une matrice de SHV à 2 dimensions comme la suivante (ayant dans ce cas *inumPointsX* = 3 et *inumPointsY* = 5) :

**Tableau 11.**

5	7	1
3	4	9
6	2	0
4	1	3
8	2	7

Ces numéros (à stocker dans la table *iSnapTab* en utilisant, par exemple, le générateur de fonction *GEN02*), représentent la position des instantanés dans la grille (dans ce cas une matrice 3x5). Ainsi, le premier élément 5 à l'indice zéro et représente le sixième (l'élément zéro est le premier) instantané contenu dans la table *iSnapTab*, le second élément 7 représente le huitième élément de *iSnapTab*, ainsi de suite. En résumé, les sommets de chaque zone (une zone cubique est délimitée par 8 sommets, une zone carrée par

4 sommets et une zone linéaire par 2 points) sont liés à un instantané donné, dont le numéro est transformé par la table *iSnapTab*.

Les valeurs de sortie de la SHV sont influencées par le mouvement du pointeur, un point dont la position dans le cube de SHV (ou le carré ou le segment) est déterminée par les arguments  $kx$ ,  $ky$  et  $kz$ . Les valeurs de ces arguments, qui doivent être comprises entre 0 et 1, sont fixées extérieurement par l'utilisateur. Les valeurs de sortie, dont le nombre est égal à l'argument *inumParms*, sont stockées dans la table *iOutTab*, laquelle doit avoir été allouée auparavant par l'utilisateur et doit avoir une taille d'au moins *inumParms*. De quelle manière le mouvement du pointeur influence-t-il la sortie ? Lorsque le pointeur tombe dans une zone cubique déterminée, délimitée par exemple par 8 sommets (ou points pivots), nous supposons que chaque sommet est associé à un instantané différent (c'est-à-dire un ensemble de *inumParms* valeurs), la sortie sera la moyenne pondérée des 8 sommets, calculée en fonction de la distance du pointeur à chacun des 8 sommets. Dans le comportement par défaut, lorsque l'argument *iConfigTab* n'est pas défini, la sortie exacte est calculée en utilisant une interpolation linéaire qui est appliquée à chaque paramètre de la SHV. Cependant, il est possible de modifier ce comportement en donnant à l'argument *iConfigTab* le numéro d'une table dont le contenu peut affecter un ou plusieurs paramètres de la SHV. Les éléments de la table *iConfigTab* sont associés à chaque paramètre de SHV et leurs valeurs affectent la sortie de la SHV de la manière suivante :

- Si *iConfigTab* vaut -1, la sortie correspondante est ignorée, c'est-à-dire que l'élément n'est pas calculé, laissant la valeur de l'élément correspondant dans la table *iOutTab* inchangée ;
- Si *iConfigTab* est égal à zéro, la sortie de la SHV normale est calculée (en utilisant la moyenne pondérée des sommets les plus proches de la zone dans laquelle le pointeur mobile est tombé) ;
- Si *iConfigTab* est égal à un nombre entier > zéro, le contenu d'une table ayant ce numéro est utilisé comme la forme d'une interpolation basée sur cette table.

## Voir aussi

*hvs1*, *hvs2*, *vphaseseg*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# hypot

hypot — Fonction distance euclidienne.

## Description

Retourne la valeur de  $\sqrt{\text{arg1}*\text{arg1} + \text{arg2}*\text{arg2}}$ .

## Syntaxe

```
ires[] hypot iarg1[], iarg2[]
```

```
kres[] hypot karg1[], karg2[]
```

## Initialisation

*iarg[]1/2* -- les opérandes.

## Exécution

*karg[]1/2* -- les opérandes.

## Exemples

Voici un exemple de l'opcode hypot. Il utilise le fichier *hypot.csd* [examples/hypot.csd].

### Exemple 427. Exemple de l'opcode hypot.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

instr 1

iArg1[] fillarray 1,2,3
iArg2[] fillarray 4,5,6
iRes[] hypot iArg1,iArg2
ik init 0

while ik < lenarray(iRes) do
  print iRes[ik]
  ik += 1
od

endin

</CsInstruments>
<CsScore>
i1 0 0
</CsScore>
```

`</CsoundSynthesizer>`

## Crédits

Auteur : Victor Lazzarini  
2017

# i

*i* — Retourne un équivalent de *taux-i* d'un argument de *taux-k*, ou directement un argument de *taux-i*.

## Description

Retourne un équivalent de *taux-i* d'un argument de *taux-k* ou d'un élément de tableau, ou directement un argument de *taux-i*.

## Syntaxe

`i(x)` (argument de *taux-k* ou de *taux-i*)

`i(karray,index1, ...)` (k-array with indices)

Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.



### Note

L'utilisation de *i()* avec en argument une expression de *taux-k* n'est pas recommandée et peut produire des résultats inattendus.

Pour obtenir de manière fiable une valeur à partir d'un élément de tableau il faut utiliser la seconde forme.

## Voir aussi

*a, k, abs, exp, frac, int, log, log10, sqrt*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/issue10/CsoundRates.html>, écrit par Andrés Cabrera.

## Crédits

La version de cette fonction prenant un argument de *taux-i* a été ajoutée dans la version 5.14 de Csound.

Le format pour tableau a été ajouté dans la version 6.08.

## if

if — Branchement conditionnel à l'initialisation ou durant l'exécution.

## Description

*if...igoto* -- branchement conditionnel à l'initialisation, dépendant de la valeur de vérité de l'expression logique *ia R ib*. Le branchement n'a lieu que si le résultat est vrai.

*if...kgoto* -- branchement conditionnel durant l'exécution, dépendant de la valeur de vérité de l'expression logique *ka R kb*. Le branchement n'a lieu que si le résultat est vrai.

*if...goto* -- combinaison des deux versions ci-dessus. La condition est testée à chaque passage.

*if...then* -- donne la possibilité de spécifier des blocs conditionnels *if/else/endif*. Tous les blocs *if...then* doivent se terminer par une instruction *endif*. Les instructions *elseif* et *else* sont facultatives. On peut utiliser n'importe quel nombre d'instructions *elseif*. Il ne peut y avoir qu'une seule instruction *else* et elle doit être la dernière instruction conditionnelle avant l'instruction *endif*. Des blocs imbriqués de *if...then* sont permis.



### Note

Notez que si la condition utilise une variable de taux-k (par exemple, « *if kval > 0* »), l'instruction *if...goto* ou *if...then* sera ignorée lors de la phase d'initialisation. Cela permet une initialisation de l'opcode même si la variable de taux-k a déjà reçu une valeur appropriée par une instruction init antérieure.

## Syntaxe

```
if ia R ib igoto label
```

```
if ka R kb kgoto label
```

```
if xa R xb goto label
```

```
if xa R xb then
```

où *label* est dans le même bloc d'instrument et n'est pas une expression, et où *R* est un des opérateurs relationnels (<, =, <=, ==, !=) (et = par commodité, voir aussi *Valeurs Conditionnelles*).

Si l'on utilise *goto* ou *then* à la place de *kgoto* ou *igoto*, le comportement est déterminé par le type étant comparé. Si la comparaison utilise des variables de taux-k, *kgoto* est utilisé et vice-versa.



### Note

Les instructions *If/then/goto* ne peuvent pas effectuer de comparaisons de type audio. On ne peut pas mettre de variables de type-a dans les expressions de comparaison pour ces opcodes. La raison en est que les variables audio sont des vecteurs qui ne peuvent pas être comparés de la même façon que des scalaires. Si l'on doit comparer des échantillons audio individuellement il faut utiliser *kr = 1* ou *Compareurs et Accumulateurs*

## Exemples

Voici une exemple de la combinaison *if ... igoto*. Il utilise le fichier *igoto.csd* [examples/igoto.csd].

## Exemple 428. Exemple de la combinaison if ... igoto.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o igoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Get the value of the 4th p-field from the score.
iparam = p4

; If iparam is 1 then play the high note.
; If not then play the low note.
if (iparam == 1) igoto highnote
    igoto lownote

highnote:
    ifreq = 880
    goto playit

lownote:
    ifreq = 440
    goto playit

playit:
; Print the values of iparam and ifreq.
print iparam
print ifreq

a1 oscil 10000, ifreq, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; p4: 1 = high note, anything else = low note
; Play Instrument #1 for one second, a low note.
i 1 0 1 0
; Play a Instrument #1 for one second, a high note.
i 1 1 1 1
e

</CsScore>
</CsoundSynthesizer>
```



Sa sortie contiendra des lignes comme celles-ci :

```
instr 1: iparam = 0.000
instr 1: ifreq = 440.000
instr 1: iparam = 1.000
instr 1: ifreq = 880.000
```

Voici un exemple de la combinaison if ... kgoto. Il utilise le fichier *kgoto.csd* [examples/kgoto.csd].

### Exemple 429. Exemple de la combinaison if ... kgoto.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o kgoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Change kval linearly from 0 to 2 over
; the period set by the third p-field.
kval line 0, p3, 2

; If kval is greater than or equal to 1 then play the high note.
; If not then play the low note.
if (kval >= 1) kgoto highnote
kgoto lownote

highnote:
kfreq = 880
goto playit

lownote:
kfreq = 440
goto playit

playit:
; Print the values of kval and kfreq.
printks "kval = %f, kfreq = %f\\n", 1, kval, kfreq

a1 oscil 10000, kfreq, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e
```

```
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
kval = 0.000000, kfreq = 440.000000
kval = 0.999732, kfreq = 440.000000
kval = 1.999639, kfreq = 880.000000
```

## Exemples

Voici un exemple de la combinaison if ... then. Il utilise le fichier *ifthen.csd* [examples/ifthen.csd].

### Exemple 430. Exemple de la combinaison if ... then.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o ifthen.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Get the note value from the fourth p-field.
knote = p4

; Does the user want a low note?
if (knote == 0) then
    kcps = 220
; Does the user want a middle note?
elseif (knote == 1) then
    kcps = 440
; Does the user want a high note?
elseif (knote == 2) then
    kcps = 880
endif

; Create the note.
kamp init 25000
ifn = 1
a1 oscili kamp, kcps, ifn

out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1
```

```
; p4: 0=low note, 1=middle note, 2=high note.  
; Play Instrument #1 for one second, low note.  
i 1 0 1 0  
; Play Instrument #1 for one second, middle note.  
i 1 1 1 1  
; Play Instrument #1 for one second, high note.  
i 1 2 1 2  
e  
  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*elseif, else, endif, goto, igoto, kgoto, tigoto, timeout*

## Crédits

Exemples écrits par Kevin Conder.

Note ajoutée par Jim Aikin.

Février 2004. Note ajoutée par Matt Ingalls.

# fftin

fftin — Transformée de Fourier rapide inverse dans le domaine complexe.

## Description

Applique une transformée de Fourier rapide inverse à un tableau unidimensionnel de valeurs complexes produisant une sortie complexe. Cette sortie est un autre tableau contenant les valeurs complexes du signal, et les deux tableaux sont au format réel-imaginaire entrelacé. Le tableau de sortie a la même taille que le tableau d'entrée et la taille de la transformée est équivalente à la moitié de la longueur du tableau. Les transformées non puissance de deux sont restreintes à des tailles paires avec un nombre pas trop élevé de facteurs.

## Syntaxe

```
kout[] fftin kin[]
```

## Exécution

*kout[]* -- tableau contenant les valeurs complexes de sortie. Créé s'il n'existe pas.

*kin[]* -- tableau contenant les valeurs complexes d'entrée.

## Exemples

Voici un exemple de l'opcode *fftin*. Il utilise le fichier *ifft.csd* [examples/ifft.csd].

### Exemple 431. Exemple de l'opcode *fftin*.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-d -o dac
</CsOptions>
<CsInstruments>
ksmps = 64

instr 1
  ifftsize = 1024
  kcnt init 0
  kIn[] init ifftsize
  kOut[] init ifftsize

  a1 oscili 0dbfs/2, 440

  if kcnt >= ifftsize then
    kCmplx[] r2c kIn
    kSpec[] fft kCmplx
    kCmplx fftin kSpec
    kOut c2r kCmplx
    kcnt = 0
  endif
```

```
kIn[] shiftin a1
a2 shiftout kOut
kcnt += ksmps
    out a2
endin
</CsInstruments>
<CsScore>
i1 0 10
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux.

## Crédits

Auteur: Victor Lazzarini  
NUI Maynooth  
2014

Nouveau dans la version 6.04

# igoto

igoto — Transfère le contrôle lors de la phase d'initialisation.

## Description

Transfère le contrôle sans condition vers l'instruction étiquetée par *label*, lors de la phase d'initialisation seulement.

## Syntaxe

```
igoto label
```

où *label* se trouve dans le même bloc d'instrument et n'est pas une expression.

## Exemples

Voici un exemple de l'opcode igoto. Il utilise le fichier *igoto.csd* [examples/igoto.csd].

### Exemple 432. Exemple de l'opcode igoto.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o igoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Get the value of the 4th p-field from the score.
iparam = p4

; If iparam is 1 then play the high note.
; If not then play the low note.
if (iparam == 1) igoto highnote
    igoto lownote

highnote:
    ifreq = 880
    goto playit

lownote:
    ifreq = 440
    goto playit
```

```

playit:
  ; Print the values of iparam and ifreq.
  print iparam
  print ifreq

  a1 oscil 10000, ifreq, 1
  out a1
endin

</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; p4: 1 = high note, anything else = low note
; Play Instrument #1 for one second, a low note.
i 1 0 1 0
; Play a Instrument #1 for one second, a high note.
i 1 1 1 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

instr 1: iparam = 0.000
instr 1: ifreq = 440.000
instr 1: iparam = 1.000
instr 1: ifreq = 880.000

```

## Voir aussi

*cggoto, cigoto, ckgoto, goto, if, kgoto, rigoto, tigoto, timeout*

## Crédits

Exemple écrit par Kevin Conder.

# ihold

ihold — Crée une note tenue.

## Description

Transforme une note de durée finie en une note « tenue ».

## Syntaxe

ihold

## Exécution

*ihold* -- cette instruction de la phase d'initialisation transforme une note de durée finie en une note « tenue ». Elle a ainsi le même effet qu'une valeur de p3 négative (voir l'*instruction i* de la partition), sauf qu'ici p3 reste positif et que l'instrument se redéfinit lui-même pour durer indéfiniment. La note peut être arrêtée explicitement par un *turnoff*, ou son espace peut être utilisé par une autre note ayant le même numéro d'instrument (c-à-d qu'elle est liée à cette note). N'agit que pendant la phase d'initialisation ; inopérant pendant une phase de réinitialisation (*reinit*).

## Exemples

Voici un exemple de l'opcode *ihold*. Il utilise le fichier *ihold.csd* [examples/ihold.csd].

### Exemple 433. Exemple de l'opcode *ihold*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ihold.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; A simple oscillator with its note held indefinitely.
a1 oscil 10000, 440, 1
ihold

; If p4 equals 0, turn the note off.
if (p4 == 0) kgoto offnow
kgoto playit
```



```
offnow:
    ; Turn the note off now.
    turnoff

playit:
    ; Play the note.
    out a1
endin

</CsInstruments>
<CsScore>

    ; Table #1: an ordinary sine wave.
    f 1 0 32768 10 1

    ; p4 = turn the note off (if it is equal to 0).
    ; Start playing Instrument #1.
    i 1 0 1 1
    ; Turn Instrument #1 off after 3 seconds.
    i 1 3 1 0
    e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*i Statement, turnoff*

## Crédits

Exemple écrit par Kevin Conder.

# imagecreate

imagecreate — Crée une image vide de la taille donnée.

## Description

Opcodes du greffon image.

Crée une image vide de taille donnée. On peut ensuite fixer des points individuellement avec *imagegetpixel*.

## Syntaxe

```
iimagenum imagecreate iwidth, iheight
```

## Initialisation

*iimagenum* -- numéro assigné à l'image créée.

*iwidth* -- largeur d'image souhaitée.

*iheight* -- hauteur d'image souhaitée.

## Exemples

Voici un exemple de l'opcode imagecreate. Il utilise le fichier *imageopcodes.csd* [examples/imageopcodes.csd].

### Exemple 434. Exemple de l'opcode imagecreate.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n ;no sound output
</CsOptions>
<CsInstruments>

sr=48000
ksmps=1
nchnls=2

; this test .csd copies image.png into a new file 'imageout.png'

giimage1 imageload "image.png"
giimagew, giimageh imagesize giimage1
giimage2 imagecreate giimagew,giimageh

instr 1

kndx = 0
kx_ linseg 0, p3, 1
;print imagewidth and imageheight of image.png
prints "imagewidth = %f pixels, imageheight = %f pixels\\n", giimagew, giimageh

myloop:
ky_ = kndx/(giimageh)
```

```

kr_, kg_, kb_ imagegetpixel giimage1, kx_, ky_
imagesetpixel giimage2, kx_, ky_, kr_, kg_, kb_
loop_lt kndx, 0.5, giimageh, myloop
    endin

    instr 2

imagesave giimage2, "imageout.png"
    endin

    instr 3
imagefree giimage1
imagefree giimage2
    endin

</CsInstruments>
<CsScore>

i1 1 1
i2 2 1
i3 3 1
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*imageload, imagesize, imagesave, imagegetpixel, imagesetpixel, imagefree*

## Crédits

Auteur : Cesare Marilungo

Nouveau dans la version 5.08

# imagefree

imagecreate — Libère la mémoire allouée pour une image précédemment chargée ou créée.

## Description

Opcodes du greffon image.

Libère la mémoire allouée pour une image précédemment chargée ou créée.

## Syntaxe

```
imagefree iimagenum
```

## Initialisation

*iimagenum* -- référence de l'image à libérer.

## Exemples

Voici un exemple de l'opcode imagefree. Il utilise le fichier *imageopcodes.csd* [examples/imageopcodes.csd].

### Exemple 435. Exemple de l'opcode imagefree.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n ;no sound output
</CsOptions>
<CsInstruments>

sr=48000
ksmps=1
nchnls=2

; this test .csd copies image.png into a new file 'imageout.png'

giimage1 imageload "image.png"
giimagew, giimageh imagesize giimage1
giimage2 imagecreate giimagew,giimageh

instr 1

kndx = 0
kx_ linseg 0, p3, 1
;print imagewidth and imageheight of image.png
prints "imagewidth = %f pixels, imageheight = %f pixels\\n", giimagew, giimageh

myloop:
ky_ = kndx/(giimageh)
kr_, kg_, kb_ imagegetpixel giimage1, kx_, ky_
imagesetpixel giimage2, kx_, ky_, kr_, kg_, kb_
loop_lt kndx, 0.5, giimageh, myloop
endin
```

```
instr 2

imagesave giimage2, "imageout.png"
endin

instr 3
imagefree giimage1
imagefree giimage2
endin

</CsInstruments>
<CsScore>

i1 1 1
i2 2 1
i3 3 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*imageload, imagecreate, imagesize, imagesave, imagegetpixel, imagesetpixel*

## Crédits

Auteur : Cesare Marilungo

Nouveau dans la version 5.08

# imagegetpixel

imagegetpixel — Retourne les valeurs RVB d'un pixel d'une image précédemment ouverte ou créée.

## Description

Opcode du greffon image.

Retourne les valeurs RVB d'un pixel d'une image précédemment ouverte ou créée. On peut charger une image avec *imageload*. On peut créer un image vide avec *imagecreate*.

## Syntaxe

```
ared, agreen, ablue imagegetpixel iimagenum, ax, ay
kred, kgreen, kblue imagegetpixel iimagenum, kx, ky
```

## Initialisation

*iimagenum* -- la référence de l'image. C'est une valeur retournée par *imageload* ou par *imagecreate*.

## Exécution

*ax (kx)* -- position horizontale du pixel (un nombre flottant compris entre 0 et 1).

*ay (ky)* -- position verticale du pixel (un nombre flottant compris entre 0 et 1).

*ared (kred)* -- valeur de rouge du pixel (ramenée à un nombre flottant compris entre 0 et 1).

*agreen (kgreen)* -- valeur de vert du pixel (ramenée à un nombre flottant compris entre 0 et 1).

*ablue (kblue)* -- valeur de bleue du pixel (ramenée à un nombre flottant compris entre 0 et 1).

## Exemples

Voici un exemple de l'opcode imagegetpixel. Il utilise les fichiers *imageopcodesdemo2.csd* [examples/imageopcodesdemo2.csd], *test1.png* [examples/test1.png] et *test2.png* [examples/test2.png].

### Exemple 436. Exemple de l'opcode imagegetpixel.

```
<CsoundSynthesizer>

<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o imageopcodesdemo2.wav -W ;; for file output any platform
</CsOptions>

<CsInstruments>
sr          =      48000
ksmps      =      100
nchnls     =  2
```

```

;By Cesare Marilungo 2008
zakinit 10,1

;Load the image - should be 512x512 pixels
giimage imageload "test1.png"
;giimage imageload "test2.png" ;--try this too
giimagew, giimageh imagesize giimage

giwave ftgen 1, 0, 1024, 10, 1
gifrqs ftgen 2,0,512,-5, 1,512,10
giamps ftgen 3, 0, 512, 10, 1

instr 100

kindex = 0
icnt = giimageh
kx_ linseg 0, p3, 1
kenv linseg 0, .2, 500, p3 - .4, 500, .2, 0

; Read a column of pixels and store the red values
; inside the table 'giamps'
loop:
    ky_ = kindex/giimageh

    ;Get the pixel color values at kx_, ky_
    kred, kgreen, kblue imagegetpixel giimage, kx_, ky_

    ;Write the red values inside the table 'giamps'
    tablew kred, kindex, giamps
    kindex = kindex+1

if (kindex < icnt) kgoto loop

; Use an oscillator bank (additive synthesis) to generate sound
; setting amplitudes for each partial according to the image
asig adsynt kenv, 220, giwave, gifrqs, giamps, icnt, 2
outs asig, asig

endin

instr 101
; Free memory used by the image
imagefree giimage
endin

</CsInstruments>
<CsScore>

t 0 60

i100 1 20
i101 21 1

e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*imageload, imagecreate, imagesize, imagesave, imagesetpixel, imagefree*

## Crédits

Auteur : Cesare Marilungo

Nouveau dans la version 5.08



# imageload

imageload — Charge une image.

## Description

Opcodes du greffon image.

Charge une image et retourne une référence sur celle-ci. On peut ensuite accéder aux valeurs d'un pixel avec *imagegetpixel*.



### Note

Les opcodes de traitement d'image ne peuvent charger que des images png.

## Syntaxe

iimagenum **imageload** filename

## Initialisation

*iimagenum* -- numéro assigné à l'image chargée.

*filename* -- le nom de fichier de l'image à charger (ce doit être un fichier d'image PNG valide).

## Exemples

Voici un exemple de l'opcode imageload. Il utilise le fichier *imageopcodes.csd* [examples/imageopcodes.csd].

### Exemple 437. Exemple de l'opcode imageload.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n ;no sound output
</CsOptions>
<CsInstruments>

sr=48000
ksmps=1
nchnls=2

; this test .csd copies image.png into a new file 'imageout.png'

giimage1 imageload "image.png"
giimagew, giimageh imagesize giimage1
giimage2 imagecreate giimagew,giimageh

instr 1

kndx = 0
kx_ linseg 0, p3, 1
;print imagewidth and imageheight of image.png
```

```

prints "imagewidth = %f pixels, imageheight = %f pixels\\n", giimagew, giimageh

myloop:
ky_ = kndx/(giimageh)
kr_, kg_, kb_ imagegetpixel giimage1, kx_, ky_
imagesetpixel giimage2, kx_, ky_, kr_, kg_, kb_
loop_lt kndx, 0.5, giimageh, myloop
    endin

    instr 2

imagesave giimage2, "imageout.png"
    endin

    instr 3
imagefree giimage1
imagefree giimage2
    endin

</CsInstruments>
<CsScore>

i1 1 1
i2 2 1
i3 3 1
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*imagecreate, imagesize, imagesave, imagegetpixel, imagesetpixel, imagefree*

## Crédits

Auteur : Cesare Marilungo

Nouveau dans la version 5.08

# imagesave

imagesave — Sauvegarde une image précédemment créée.

## Description

Opcodes du greffon image.

Sauvegarde une image précédemment créée. On peut créer une image vide avec *imagecreate* et on peut fixer ses valeurs RVB de pixel avec *imagegetpixel*. L'image sera sauvegardée au format PNG.

## Syntaxe

```
imagesave iimagenum, filename
```

## Initialisation

*iimagenum* -- la référence de l'image à sauvegarder. C'est une valeur retournée par *imagecreate*.

*filename* -- le nom de fichier à utiliser pour sauvegarder l'image.

## Exemples

Voici un exemple de l'opcode imagesave. Il utilise le fichier *imageopcodes.csd* [exemples/imageopcodes.csd].

### Exemple 438. Exemple de l'opcode imagesave.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n ;no sound output
</CsOptions>
<CsInstruments>

sr=48000
ksmps=1
nchnls=2

; this test .csd copies image.png into a new file 'imageout.png'

giimage1 imageload "image.png"
giimagew, giimageh imagesize giimage1
giimage2 imagecreate giimagew,giimageh

instr 1

kndx = 0
kx_ linseg 0, p3, 1
;print imagewidth and imageheight of image.png
prints "imagewidth = %f pixels, imageheight = %f pixels\\n", giimagew, giimageh

myloop:
ky_ = kndx/(giimageh)
kr_, kg_, kb_ imagegetpixel giimage1, kx_, ky_
```

```

imagesetpixel giimage2, kx_, ky_, kr_, kg_, kb_
loop_lt kndx, 0.5, giimageh, myloop
    endin

    instr 2

imagesave giimage2, "imageout.png"
    endin

    instr 3
imagefree giimage1
imagefree giimage2
    endin

</CsInstruments>
<CsScore>

i1 1 1
i2 2 1
i3 3 1
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*imageload, imagecreate, imagesize, imagegetpixel, imagesetpixel, imagefree*

## Crédits

Auteur : Cesare Marilungo

Nouveau dans la version 5.08

# imagesetpixel

imagesetpixel — Fixe la valeur RVB d'un pixel dans une image précédemment ouverte ou créée.

## Description

Opcodes du greffon image.

Fixe la valeur RVB d'un pixel dans une image précédemment ouverte ou créée. On peut charger une image avec *imageload*. On peut créer un image vide avec *imagecreate* et la sauver avec *imagesave*.

## Syntaxe

```
imagesetpixel iimagenum, ax, ay, ared, agreen, ablue
```

```
imagesetpixel iimagenum, kx, ky, kred, kgreen, kblue
```

## Initialisation

*iimagenum* -- la référence de l'image. C'est une valeur retournée par *imageload* ou par *imagecreate*.

## Exécution

*ax* (*kx*) -- position horizontale du pixel (un nombre flottant compris entre 0 et 1).

*ay* (*ky*) -- position verticale du pixel (un nombre flottant compris entre 0 et 1).

*ared* (*kred*) -- valeur de rouge du pixel (ramenée à un nombre flottant compris entre 0 et 1).

*agreen* (*kgreen*) -- valeur de vert du pixel (ramenée à un nombre flottant compris entre 0 et 1)

*ablue* (*kblue*) -- valeur de bleue du pixel (ramenée à un nombre flottant compris entre 0 et 1).

## Exemples

Voici un exemple de l'opcode imagesetpixel. Il utilise le fichier *imageopcodes.csd* [examples/imageopcodes.csd].

### Exemple 439. Exemple de l'opcode imagesetpixel.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n ;no sound output
</CsOptions>
<CsInstruments>

sr=48000
ksmps=1
nchnls=2

; this test .csd copies image.png into a new file 'imageout.png'
```

```

giimage1 imageload "image.png"
giimagew, giimageh imagesize giimage1
giimage2 imagecreate giimagew,giimageh

    instr 1

kndx = 0
kx_ linseg 0, p3, 1
;print imagewidth and imageheight of image.png
prints "imagewidth = %f pixels, imageheight = %f pixels\\n", giimagew, giimageh

myloop:
ky_ = kndx/(giimageh)
kr_, kg_, kb_ imagegetpixel giimage1, kx_, ky_
imagesetpixel giimage2, kx_, ky_, kr_, kg_, kb_
loop_lt kndx, 0.5, giimageh, myloop
    endin

    instr 2

imagesave giimage2, "imageout.png"
    endin

    instr 3
imagefree giimage1
imagefree giimage2
    endin

</CsInstruments>
<CsScore>

i1 1 1
i2 2 1
i3 3 1
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*imageload, imagecreate, imagesize, imagesave, imagegetpixel, imagefree*

## Crédits

Auteur : Cesare Marilungo

Nouveau dans la version 5.08

# imagesize

`imagesize` — Retourne la largeur et la hauteur d'une image précédemment ouverte ou créée.

## Description

Opcodes du greffon `image`.

Retourne la largeur et la hauteur d'une image précédemment ouverte ou créée. On peut charger une image avec *imageload*. On peut créer un image vide avec *imagecreate*.

## Syntaxe

```
iwidth, iheight imagesize iimagenum
```

## Initialisation

*iimagenum* -- la référence de l'image. C'est une valeur retournée par *imageload* ou par *imagecreate*.

*iwidth* -- largeur de l'image.

*iheight* -- hauteur de l'image.

## Exemples

Voici un exemple de l'opcode `imagesize`. Il utilise le fichier *imageopcodes.csd* [examples/imageopcodes.csd].

### Exemple 440. Exemple de l'opcode `imagesize`.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n ;no sound output
</CsOptions>
<CsInstruments>

sr=48000
ksmps=1
nchnls=2

; this test .csd copies image.png into a new file 'imageout.png'

giimage1 imageload "image.png"
giimagew, giimageh imagesize giimage1
giimage2 imagecreate giimagew,giimageh

instr 1

kndx = 0
kx_ linseg 0, p3, 1
;print imagewidth and imageheight of image.png
prints "imagewidth = %f pixels, imageheight = %f pixels\\n", giimagew, giimageh

myloop:
```

```
ky_ = kndx/(giimageh)
kr_, kg_, kb_ imagegetpixel giimage1, kx_, ky_
imagesetpixel giimage2, kx_, ky_, kr_, kg_, kb_
loop_lt kndx, 0.5, giimageh, myloop
  endin

  instr 2

imagesave giimage2, "imageout.png"
endin

  instr 3
imagefree giimage1
imagefree giimage2
endin

</CsInstruments>
<CsScore>

i1 1 1
i2 2 1
i3 3 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
imagewidth = 180.000000 pixels, imageheight = 135.000000 pixels
```

## Voir aussi

*imagerload, imagecreate, imagesave, imagegetpixel, imagesetpixel, imagefree*

## Crédits

Auteur : Cesare Marilungo

Nouveau dans la version 5.08



# in

in — Lit des données audio depuis un périphérique externe ou un flot.

## Description

Lit des données audio depuis un périphérique externe ou un flot.



### Avertissement

Il y a deux versions de cet opcode. La première est conçue pour ne fonctionner qu'avec des orchestres qui ont *inchnls*=1. Avec des orchestres dont *inchnls* > 1, la sortie audio ne sera pas correcte.

La seconde forme lit des canaux multiples vers un tableau.

## Syntaxe

```
arl in
```

```
aarray in
```

## Exécution

La première forme lit des données audio mono depuis un périphérique externe ou un flot. Si l'option de ligne de commande *-i* est positionnée, le son est lu en continu depuis le flot audio en entrée (par exemple *stdin* ou un fichier son) dans un tampon interne. N'importe quel nombre de ces opcodes peuvent lire librement depuis ce tampon.

Le second format lit jusqu'à *inchnls* audio dans un tableau audio qui doit être initialisé.

## Exemples

Voici un exemple de l'opcode in. Il utilise le fichier *in.csd* [examples/in.csd].

### Exemple 441. Exemple de l'opcode in.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -idac   ;;realtime audio I/O
; For Non-realtime ouput leave only the line below:
; in.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;1 channel in, two channels out
```

```
ain1 in ;grab your mic and sing
adel linseg 0, p3*.5, 0.02, p3*.5, 0 ;max delay time = 20ms
aout flanger ain1, adel, .7
      fout "in_1.wav", 14, aout, aout ;write to stereo file,
      outs aout, aout ;16 bits with header

endin
</CsInstruments>
<CsScore>

i 1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*diskin, inh, inh, ino, inq, ins, soundin*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997  
Authors: John ffitich  
NUIM, 2013

Déjà dans la version 3.30

Variante pour tableau ajoutée dans la version 6.01

# in32

in32 — Lit un signal audio sur 32 canaux depuis un périphérique externe ou un flot.

## Description

Lit un signal audio sur 32 canaux depuis un périphérique externe ou un flot.



### Avertissement

Cet opcode est prévu pour ne fonctionner qu'avec des orchestres qui ont *nchnls\_i*=32. Avec des orchestres dont *nchnls\_i* > 32, la sortie audio ne sera pas correcte.

## Syntaxe

```
ar1, ar2, ar3, ar4, ar5, ar6, ar7, ar8, ar9, ar10, ar11, ar12, ar13, ar14, \  
ar15, ar16, ar17, ar18, ar19, ar20, ar21, ar22, ar23, ar24, ar25, ar26, \  
ar27, ar28, ar29, ar30, ar31, ar32 in32
```

## Exécution

*in32* lit un signal audio sur 32 canaux depuis un périphérique externe ou un flot. Si l'option de ligne de commande *-i* est positionnée, le son est lu en continu depuis le flot audio en entrée (par exemple *stdin* ou un fichier son) dans un tampon interne.

## Voir aussi

*inch*, *inx*, *inz*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.07 de Csound

# inch

*inch* — Lit depuis des canaux numérotés d'un signal audio externe ou d'un flot.

## Description

Lit depuis des canaux numérotés d'un signal audio externe ou d'un flot.

## Syntaxe

```
ain1[, ...] inch kchan1[,...]
```

## Exécution

*ain*, ... - signaux audio en entrée

*kchan1*, ... - numéro des canaux

*inch* lit depuis des canaux numérotés déterminés par les *kchan* correspondants vers les *ain* associés. Si l'option de ligne de commande *-i* est positionnée, le son est lu en continu depuis le flot audio en entrée (par exemple *stdin* ou un fichier son). On peut aussi utiliser *inch* pour recevoir des données audio en temps réel depuis l'interface audio au moyen de *-iadc*.



### Note

La plus grande valeur de *kchan* utilisable avec *inch* dépend de *nchnls\_i*. Si *kchan* est supérieur à *nchnls\_i*, *ain* restera silencieux. Noter que dans ce cas *inch* donnera un avertissement et non une erreur.

## Exemples

Voici un exemple de l'opcode *inch*. Il utilise le fichier *inch.csd* [examples/inch.csd].

### Exemple 442. Exemple de l'opcode *inch*.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -iadc ;;realtime audio I/O
; For Non-realtime ouput leave only the line below:
; inch.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;nchnls channels in, two channels out

ain1, ainr inch 1, 2 ;grab your mic and sing
adel linseg 0, p3*.5, 0.02, p3*.5, 0 ;max delay time = 20ms
```

```
aoutl flanger ainl, adel, .7
aoutr flanger ainl, adel*2, .8
fout "in_ch.wav", 14, aoutl, aoutr ;write to stereo file,
outs aoutl, aoutr ;16 bits with header

endin
</CsInstruments>
<CsScore>

i 1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*in32, inx, inz*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.07 de Csound

Arguments multiples depuis la version 5.13

# inh

inh — Lit des données audio sur six canaux depuis un périphérique externe ou un flot.

## Description

Lit des données audio sur six canaux depuis un périphérique externe ou un flot.



### Avertissement

Cet opcode est prévu pour ne fonctionner qu'avec des orchestres qui ont *nchnls\_i*=6. Avec des orchestres dont *nchnls\_i* > 6, la sortie audio ne sera pas correcte.

## Syntaxe

*ar1, ar2, ar3, ar4, ar5, ar6 inh*

## Exécution

Lit des données audio sur six canaux depuis un périphérique externe ou un flot. Si l'option de ligne de commande *-i* est positionnée, le son est lu en continu depuis le flot audio en entrée (par exemple *stdin* ou un fichier son) dans un tampon interne. N'importe quel nombre de ces opcodes peuvent lire librement depuis ce tampon.

## Voir aussi

*diskin, in, ino, inq, ins, soundin*

## Crédits

Auteur : John ffitich

# init

`init` — Met la valeur de l'expression de `taux-i` dans une variable de `taux-k` ou de `taux-a`, ou dans une variable-`t`.

## Syntaxe

```
ares init iarg
ires init iarg
kres init iarg
ares, ... init iarg, ...
ires, ... init iarg, ...
kres, ... init iarg, ...
tab init isize[, ival]
```

## Description

Met la valeur de l'expression de `taux-i` dans une variable de `taux-k` ou de `taux-a`.

## Initialisation

Met la valeur de l'expression de `taux-i` *iarg* dans une variable de `taux-k` ou de `taux-a`, ou dans une variable-`t`, c-à-d., initialise le résultat. Noter que `init` présente le seul cas d'une instruction de la période d'initialisation autorisée à écrire dans un résultat de la période d'exécution (`taux-k` ou `-a`) ; cette instruction n'a aucun effet pendant l'exécution.

Depuis la version 5.13 il est possible d'initialiser jusqu'à 24 variables de la même classe dans une instruction. S'il y a plus de variables en sortie que d'expressions en entrée, la dernière expression est répétée. C'est une erreur d'avoir plus d'entrées que de sorties.

La forme variable-`t`, introduite dans la version 5.14, alloue de l'espace pour un vecteur de la taille donnée, initialisé avec la valeur donnée (qui vaut zéro par défaut).

## Exemples

Voici un exemple de l'opcode `init`. Il utilise le fichier *init.csd* [examples/init.csd].

### Exemple 443. Exemple de l'opcode `init`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n ;no sound output
</CsOptions>
<CsInstruments>

sr      = 44100
```

```
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;shows what init does
  kinit init 0
  kinit = kinit + 1
  printk .1, kinit
endin

instr 2 ;shows what an assignment does
  knoinit = 0
  knoinit = knoinit + 1
  printk .1, knoinit
endin
</CsInstruments>
<CsScore>
;play one second each
i1 0 1
i2 2 1
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
i 1 time 0.00073: 1.00000
i 1 time 0.10014: 138.00000
i 1 time 0.20027: 276.00000
i 1 time 0.30041: 414.00000
i 1 time 0.40054: 552.00000
i 1 time 0.50068: 690.00000
i 1 time 0.60009: 827.00000
i 1 time 0.70023: 965.00000
i 1 time 0.80036: 1103.00000
i 1 time 0.90050: 1241.00000

i 2 time 2.00054: 1.00000
i 2 time 2.09995: 1.00000
i 2 time 2.20009: 1.00000
i 2 time 2.30023: 1.00000
i 2 time 2.40036: 1.00000
i 2 time 2.50050: 1.00000
i 2 time 2.59991: 1.00000
i 2 time 2.70005: 1.00000
i 2 time 2.80018: 1.00000
i 2 time 2.90032: 1.00000
```

## Voir aussi

*=, divz, tival*

D'autres informations sur cet opcode dans les Floss Manuals : <http://write.flossmanuals.net/csound/a-initialization-and-performance-pass> [http://write.flossmanuals.net/csound/a-initialization-and-performance-pass]

## Crédits

*init* était présent dans le Csound original, mais l'extension aux valeurs multiples a été ajoutée par

Auteur : John ffitch



Université de Bath, and Codemist Ltd.  
Bath, UK  
February 2010

La forme multiple a été introduite dans la version 5.13 ; la forme variable-t apparait dans la version 5.14.

# initc14

*initc14* — Initialise les contrôleurs pour créer une valeur MIDI sur 14 bit.

## Description

Initialise les contrôleurs pour créer une valeur MIDI sur 14 bit.

## Syntaxe

```
initc14 ichan, ictlno1, ictlno2, ivalue
```

## Initialisation

*ichan* -- canal MIDI (1-16)

*ictlno1* -- numéro de contrôleur pour l'octet de poids fort (0-127)

*ictlno2* -- numéro de contrôleur pour l'octet de poids faible (0-127)

*ivalue* -- valeur décimale (doit être entre 0 et 1)

## Exécution

*initc14* peut être utilisé conjointement avec les opcodes *midic14* et *ctrl14* pour initialiser la première valeur du contrôleur. L'argument *ivalue* doit être un nombre entre 0 et 1. Une erreur aura lieu si ce n'est pas le cas. Utiliser cette formule afin d'ajuster *ivalue* selon les limites min et max de l'intervalle de *midic14* et de *ctrl14*:

$$\text{ivalue} = (\text{valeur\_initiale} - \text{min}) / (\text{max} - \text{min})$$

## Voir aussi

*ctrl7*, *ctrl14*, *ctrl21*, *ctrlinit*, *initc7*, *initc21*, *midic7*, *midic14*, *midic21*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

# initc21

*initc21* — Initialise les contrôleurs pour créer une valeur MIDI sur 21 bit.

## Description

Initialise les contrôleurs pour créer une valeur MIDI sur 21 bit.

## Syntaxe

```
initc21 ichan, ictlno1, ictlno2, ictlno3, ivalue
```

## Initialisation

*ichan* -- canal MIDI (1-16)

*ictlno1* -- numéro de contrôleur pour l'octet de poids fort (0-127)

*ictlno2* -- numéro de contrôleur pour l'octet de poids moyen (0-127)

*ictlno3* -- numéro de contrôleur pour l'octet de poids faible (0-127)

*ivalue* -- valeur décimale (doit être entre 0 et 1)

## Exécution

*initc21* peut être utilisé conjointement avec les deux opcodes *midic21* et *ctrl21* pour initialiser la première valeur du contrôleur. L'argument *ivalue* doit être un nombre entre 0 et 1. Une erreur aura lieu si ce n'est pas le cas. Utiliser cette formule afin d'ajuster *ivalue* selon les limites min et max de l'intervalle de *midic21* et de *ctrl21*:

$$ivalue = (valeur\_initiale - min) / (max - min)$$

## Voir aussi

*ctrl7*, *ctrl14*, *ctrl21*, *ctrlinit*, *initc7*, *initc14*, *midic7*, *midic14*, *midic21*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

# initc7

*initc7* — Initialise le contrôleur utilisé pour créer une valeur MIDI sur 7 bit.

## Description

Initialise le contrôleur MIDI *ictlno* avec *ivalue*

## Syntaxe

```
initc7 ichan, ictlno, ivalue
```

## Initialisation

*ichan* -- canal MIDI (1-16)

*ictlno* -- numéro du contrôleur (0-127)

*ivalue* -- valeur décimale (doit être entre 0 et 1)

## Exécution

*initc7* peut être utilisé conjointement avec les opcodes *midic7* et *ctrl7* pour initialiser la première valeur du contrôleur. L'argument *ivalue* doit être un nombre entre 0 et 1. Une erreur aura lieu si ce n'est pas le cas. Utiliser cette formule afin d'ajuster *ivalue* selon les limites min et max de l'intervalle de *midic7* et de *ctrl7*:

$$ivalue = (valeur\_initiale - min) / (max - min)$$

## Exemples

Voici un exemple de l'opcode *initc7*. Il utilise le fichier *initc7.csd* [examples/initc7.csd].

### Exemple 444. Exemple de l'opcode *initc7*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac  -M0  ;;realtime audio I/O with MIDI in
;-iadc  ;;uncomment -iadc if RT audio input is needed too
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; expects MIDI controller input on channel 1
; run and move your midi controller to see result
```

```
imax = 1
imin = 0
ichan = 1
ictlno = 7

  initc7 1, 7, 1 ; start at max. volume
  kamp ctrl7 ichan, ictlno, imin, imax ; controller 7
  printk2 kamp
  asig oscil kamp, 220, 1
  outs asig, asig

endin

</CsInstruments>
<CsScore>
f 1 0 4096 10 1

i1 0 20

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*ctrl7, ctrl14, ctrl21, ctrlinit, initc14, initc21, midic7, midic14, midic21*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué les bons intervalles pour le canal MIDI et le numéro de contrôleur.

# inleta

inleta — Reçoit un signal de taux-a sur un port nommé dans un instrument.

## Description

Opcode du greffon signalflowgraph.

Reçoit un signal de taux-a sur un port nommé dans un instrument.

## Syntaxe

asignal **inleta** Sname

## Initialisation

*Sname* -- Nom sous forme de chaîne de caractères du port entrant. Le nom du connecteur entrant est qualifié implicitement par le nom ou le numéro de l'instrument, si bien qu'il est permis d'utiliser le même nom de connecteur entrant dans plus d'un instrument (mais par contre on ne peut pas utiliser deux fois le même nom de connecteur entrant dans un instrument).

## Exécution

*asignal* -- signal audio en entrée.

Durant l'exécution, le signal de taux-a reçu sur le connecteur entrant provient de chaque instance d'un instrument contenant un connecteur sortant auquel ce connecteur entrant a été relié au moyen de l'opcode connect. Les signaux de tous les connecteurs sortants reliés à un connecteur entrant sont additionnés dans le connecteur entrant.

## Exemples

Voici un exemple de l'opcode inleta. Il utilise le fichier *inleta.csd* [examples/inleta.csd].

### Exemple 445. Exemple de l'opcode inleta.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o inleta.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

; Connect up instruments and effects to create the signal flow graph.

connect "SimpleSine", "leftout", "Reverberator", "leftin"
connect "SimpleSine", "rightout", "Reverberator", "rightin"
```

```

connect "Moogy",          "leftout", "Reverberator", "leftin"
connect "Moogy",          "rightout", "Reverberator", "rightin"

connect "Reverberator", "leftout", "Compressor", "leftin"
connect "Reverberator", "rightout", "Compressor", "rightin"

connect "Compressor", "leftout", "Soundfile", "leftin"
connect "Compressor", "rightout", "Soundfile", "rightin"

; Turn on the "effect" units in the signal flow graph.

alwayson "Reverberator", 0.91, 12000
alwayson "Compressor"
alwayson "Soundfile"

; Define instruments and effects in order of signal flow.

instr SimpleSine
; Default values:  p1 p2 p3 p4 p5 p6 p7 p8 p9 p10
; pset            0, 0, 10, 0, 0, 0, 0, 0.5
iattack = 0.015
idecay = 0.07
isustain = p3
irelease = 0.3
p3 = iattack + idecay + isustain + irelease
adamping linsegr 0.0, iattack, 1.0, idecay + isustain, 1.0, irelease, 0.0
iHz = cpsmidinn(p4)
; Rescale MIDI velocity range to a musically usable range of dB.
iamplitude = ampdb(p5 / 127 * 15.0 + 60.0)
; Use ftgenonce instead of ftgen, ftgentmp, or f statement.
icosine ftgenonce 0, 0, 65537, 11, 1
aoscili oscili iamplitude, iHz, icosine
aadsr madsr iattack, idecay, 0.6, irelease
asignal = aoscili * aadsr
aleft, aright pan2 asignal, p7
; Stereo audio output to be routed in the orchestra header.
outleta "leftout", aleft
outleta "rightout", aright
endin

instr Moogy
; Default values:  p1 p2 p3 p4 p5 p6 p7 p8 p9 p10
; pset            0, 0, 10, 0, 0, 0, 0, 0.5
iattack = 0.003
isustain = p3
irelease = 0.05
p3 = iattack + isustain + irelease
adamping linsegr 0.0, iattack, 1.0, isustain, 1.0, irelease, 0.0
iHz = cpsmidinn(p4)
; Rescale MIDI velocity range to a musically usable range of dB.
iamplitude = ampdb(p5 / 127 * 20.0 + 60.0)
print iHz, iamplitude
; Use ftgenonce instead of ftgen, ftgentmp, or f statement.
isine ftgenonce 0, 0, 65537, 10, 1
asignal vco iamplitude, iHz, 1, 0.5, isine
kfco line 2000, p3, 200
krez = 0.8
asignal moogvcf asignal, kfco, krez, 100000
asignal = asignal * adamping
aleft, aright pan2 asignal, p7
; Stereo audio output to be routed in the orchestra header.
outleta "leftout", aleft
outleta "rightout", aright
endin

```

```

instr Reverberator
//////////
; Stereo input.
aleftin  inleta "leftin"
arightin inleta "rightin"
idelay   = p4
icutoff  = p5
aleft, aright reverbbsc aleftin, arightin, idelay, icutoff
; Stereo output.
outleta "leftout", aleft
outleta "rightout", aright
endin

instr Compressor
//////////
; Stereo input.
aleftin  inleta "leftin"
arightin inleta "rightin"
kthreshold = 25000
icompl    = 0.5
icomp2    = 0.763
irtime    = 0.1
ifetime   = 0.1
aleftout  dam aleftin, kthreshold, icomp1, icomp2, irtime, iftime
arightout dam arightin, kthreshold, icomp1, icomp2, irtime, iftime
; Stereo output.
outleta "leftout", aleftout
outleta "rightout", arightout
endin

instr Soundfile
//////////
; Stereo input.
aleftin  inleta "leftin"
arightin inleta "rightin"
outs aleftin, arightin
endin

</CsInstruments>
<CsScore>

; It is not necessary to activate "effects" or create f-tables in the score!
; Overlapping notes create new instances of instruments with proper connections.

i "SimpleSine" 1 5 60 85
i "SimpleSine" 2 5 64 80
i "Moogy" 3 5 67 75
i "Moogy" 4 5 71 70
; 1 extra second after the performance
e 1

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*outleta, outlekt, outletkid, outlekt, inlekt, inletkid, inlekt, connect, alwayson, ftgenonce.*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/issue13/signalFlowGraphOp-codes.html>, écrit par Michael Gogins.

## Crédits

Par Michael Gogins, 2009



# inletk

inletk — Reçoit un signal de taux-k sur un port nommé dans un instrument.

## Description

Opcodes du greffon signalflowgraph.

Reçoit un signal de taux-k sur un port nommé dans un instrument.

## Syntaxe

```
ksignal inletk Sname
```

## Initialisation

*Sname* -- Nom sous forme de chaîne de caractères du port entrant. Le nom du connecteur entrant est qualifié implicitement par le nom ou le numéro de l'instrument, si bien qu'il est permis d'utiliser le même nom de connecteur entrant dans plus d'un instrument (mais par contre on ne peut pas utiliser deux fois le même nom de connecteur entrant dans un instrument).

## Exécution

*ksignal* -- signal de taux-k en entrée.

Durant l'exécution, le signal de taux-k reçu sur le connecteur entrant provient de chaque instance d'un instrument contenant un connecteur sortant auquel ce connecteur entrant a été relié au moyen de l'opcode connect. Les signaux de tous les connecteurs sortants reliés à un connecteur entrant sont additionnés dans le connecteur entrant.

## Exemples

Voici un exemple de l'opcode inletk. Il utilise le fichier *inletk.csd* [examples/inletk.csd].

### Exemple 446. Exemple de l'opcode inletk.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o inletk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

connect "bend", "bendout", "guitar", "bendin"
```

```

instr bend

kbend line p4, p3, p5
      outletk "bendout", kbend
endin

instr guitar

kbend inletk "bendin"
kpch pow 2, kbend/12
      printk2 kpch
asig oscili .4, 440*kpch, 1
      outs asig, asig
endin

</CsInstruments>
<CsScore>
f1 0 1024 10 1

i"guitar" 0 5 8.00
i"bend" 3 .2 -12 12
i"bend" 4 .1 -17 40
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*outleta, outletk, outletkid, outletf, inleta, inletf, connect, alwayson, figenonce.*

## Crédits

Par Michael Gogins, 2009

# inletkid

inletkid — Reçoit un signal de taux-k sur un port nommé dans un instrument.

## Description

Opcode du greffon signalflowgraph.

Reçoit un signal de taux-k sur un port nommé dans un instrument.

## Syntaxe

```
ksignal inletkid Sname, SinstanceID
```

## Initialisation

*Sname* -- Nom sous forme de chaîne de caractères du port entrant. Le nom du connecteur entrant est qualifié implicitement par le nom ou le numéro de l'instrument, si bien qu'il est permis d'utiliser le même nom de connecteur entrant dans plus d'un instrument (mais par contre on ne peut pas utiliser deux fois le même nom de connecteur entrant dans un instrument).

*SinstanceID* -- Nom sous forme de chaîne de caractères de l'ID de l'instance du port sortant. Cela permet au port entrant de distinguer différentes instances du port sortant, par exemple une instance du port sortant pourrait être créée par une note spécifiant un ID d'instance et une autre instance pourrait être créée par une note spécifiant un autre ID. On pourrait utiliser ceci pour situer différentes instances d'un instrument à différents points d'un espace Ambisonic dans un processeur d'effet de spatialisation.

## Exécution

*ksignal* -- signal de taux-k en entrée.

Durant l'exécution, le signal de taux-k reçu sur le connecteur entrant provient de chaque instance d'un instrument contenant un connecteur sortant auquel ce connecteur entrant a été relié au moyen de l'opcode connect. Les signaux de tous les connecteurs sortants reliés à un connecteur entrant, mais seulement ceux qui partagent l'ID d'instance spécifié, sont additionnés dans le connecteur entrant.

## Voir aussi

*outleta, outletk, outletkid, outletf, inleta, inletf, connect, alwayson, fngenonce.*

## Crédits

Par Michael Gogins, 2009

# inletf

inletf — Reçoit un signal de taux-f (fsig) sur un port nommé dans un instrument.

## Description

Opcodes du greffon signalflowgraph.

Reçoit un signal de taux-f (fsig) sur un port nommé dans un instrument.

## Syntaxe

```
fsignal inletf Sname
```

## Initialisation

*Sname* -- Nom sous forme de chaîne de caractères du port entrant. Le nom du connecteur entrant est qualifié implicitement par le nom ou le numéro de l'instrument, si bien qu'il est permis d'utiliser le même nom de connecteur entrant dans plus d'un instrument (mais par contre on ne peut pas utiliser deux fois le même nom de connecteur entrant dans un instrument).

## Exécution

*fsignal* -- signal de taux-f en entrée.

Durant l'exécution, le signal de taux-f reçu sur le connecteur entrant provient de chaque instance d'un instrument contenant un connecteur sortant auquel ce connecteur entrant a été relié au moyen de l'opcode connect. Les signaux de tous les connecteurs sortants reliés à un connecteur entrant sont additionnés dans le connecteur entrant.

## Voir aussi

*outleta*, *outletk*, *outletkid*, *outletf*, *inleta*, *inletk*, *inletkid*, *connect*, *alwayson*, *ftgenonce*.

## Crédits

Par Michael Gogins, 2009

# inletv

inletv — Reçoit un signal, tableau de taux-a, sur un port nommé dans un instrument.

## Description

Opcodes du greffon signalflowgraph.

Reçoit un signal, tableau de taux-a, sur un port nommé dans un instrument.

## Syntaxe

array **inletv** Sname

## Initialisation

*Sname* -- Nom sous forme de chaîne de caractères du port entrant. Le nom du connecteur entrant est qualifié implicitement par le nom ou le numéro de l'instrument, si bien qu'il est permis d'utiliser le même nom de connecteur entrant dans plus d'un instrument (mais par contre on ne peut pas utiliser deux fois le même nom de connecteur entrant dans un instrument).

## Exécution

*array* -- tableau du signal audio en entrée.

Durant l'exécution, le tableau du signal de taux-a reçu sur le connecteur entrant provient de chaque instance d'un instrument contenant un connecteur sortant auquel ce connecteur entrant a été relié au moyen de l'opcode connect. Les signaux de tous les connecteurs sortants reliés à un connecteur entrant sont additionnés dans le connecteur entrant. Les ports peuvent avoir n'importe quel nombre de canaux, mais le port du connecteur d'entrée doit avoir le même nombre de canaux que les ports des connecteurs de sortie.

## Voir aussi

*outleta outletf outletk outletkid outletv inleta inletk inletkid inletf connect alwayson ftgenonce*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/issue13/signalFlowGraphOp-codes.html>, écrit par Michael Gogins.

## Crédits

Par Michael Gogins, 2009

# ino

ino — Lit des données audio sur huit canaux depuis un périphérique externe ou un flot.

## Description

Lit des données audio sur huit canaux depuis un périphérique externe ou un flot.



### Avertissement

Cet opcode est prévu pour ne fonctionner qu'avec des orchestres qui ont *nchnls\_i*=8. Avec des orchestres dont *nchnls\_i* > 8, la sortie audio ne sera pas correcte.

## Syntaxe

*ar1, ar2, ar3, ar4, ar5, ar6, ar7, ar8 ino*

## Exécution

Lit des données audio sur huit canaux depuis un périphérique externe ou un flot. Si l'option de ligne de commande *-i* est positionnée, le son est lu en continu depuis le flot audio en entrée (par exemple *stdin* ou un fichier son) dans un tampon interne. N'importe quel nombre de ces opcodes peuvent lire librement depuis ce tampon.

## Voir aussi

*diskin, in, inh, inh, inq, ins, soundin*

## Crédits

Auteur : John ffitich

# inq

inq — Lit des données audio quadro depuis un périphérique externe ou un flot.

## Description

Lit des données audio quadro depuis un périphérique externe ou un flot.



### Avertissement

Cet opcode est prévu pour ne fonctionner qu'avec des orchestres qui ont *nchnls\_i*=4. Avec des orchestres dont *nchnls\_i* > 4, la sortie audio ne sera pas correcte.

## Syntaxe

```
ar1, ar2, ar3, a4 inq
```

## Exécution

Lit des données audio quadro depuis un périphérique externe ou un flot. Si l'option de ligne de commande -i est positionnée, le son est lu en continu depuis le flot audio en entrée (par exemple *stdin* ou un fichier son) dans un tampon interne. N'importe quel nombre de ces opcodes peuvent lire librement depuis ce tampon.

## Exemples

Voici un exemple de l'opcode inq. Il utilise le fichier *inq.csd* [examples/inq.csd].

### Exemple 447. Exemple de l'opcode inq.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -idac ;;realtime audio I/O
; For Non-realtime ouput leave only the line below:
; inq.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 32
nchnls  = 2 ;2 channels out
0dbfs   = 1
nchnls_i = 4 ;4 channels in

instr 1 ;4 channels in, two channels out

ain1, ain2, ain3, ain4 inq ;grab your mics and sing

adel1    linseg 0, p3*.5, 0.02, p3*.5, 0 ;max delay time = 20ms
adel2    linseg 0.02, p3*.5, 0, p3*.5, 0.02 ;max delay time = 20ms
aout1    flanger ain1, adel1, .7
aoutr    flanger ain2, adel2, .8
aoutla   flanger ain3, adel2, .9
aoutra   flanger ain4, adel2*2, .5
```

```
;write to quad file, 16 bits with header
fout "in_4.wav", 14, aoutl, aoutl, aoutl, aoutl
outs (aoutl+aoutl)*.5, (aoutl+aoutl)*.5 ;stereo out

endin
</CsInstruments>
<CsScore>

i 1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*diskin, in, inh, inh, ino, ins, soundin*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997



# inrg

*inrg* — Permet une entrée depuis un ensemble de canaux contigus du périphérique d'entrée audio.

## Description

*inrg* lit des données audio depuis un ensemble de canaux contigus du périphérique d'entrée audio.

## Syntaxe

```
inrg kstart, ain1 [,ain2, ain3, ..., ainN]
```

## Exécution

*kstart* - le numéro du premier canal du périphérique d'entrée à lire (les numéros des canaux commencent à 1, qui est le premier canal).

*ain1*, *ain2*, ... *ainN* - les arguments de sortie remplis avec les données audio lues depuis les canaux correspondants.

*inrg* permet une entrée depuis un ensemble de canaux contigus du périphérique d'entrée audio. *kstart* indique le premier canal à lire (le canal 1 étant le premier canal). Il faut s'assurer que le nombre obtenu en ajoutant à *kstart* le nombre de canaux à lire - 1 est  $\leq nchnls_i$ .



### Note

Noter que cet opcode est exceptionnel en ce sens qu'il produit sa « sortie » dans les paramètres situés à sa droite.

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# ins

ins — Lit des données audio stéréo depuis un périphérique externe ou un flot.

## Description

Lit des données audio stéréo depuis un périphérique externe ou un flot.



### Avertissement

Cet opcode est prévu pour ne fonctionner qu'avec des orchestres qui ont *nchnls\_i*=2. Avec des orchestres dont *nchnls\_i* > 2, la sortie audio ne sera pas correcte.

## Syntaxe

ar1, ar2 **ins**

## Exécution

Lit des données audio stéréo depuis un périphérique externe ou un flot. Si l'option de ligne de commande *-i* est positionnée, le son est lu en continu depuis le flot audio en entrée (par exemple *stdin* ou un fichier son) dans un tampon interne. N'importe quel nombre de ces opcodes peuvent lire librement depuis ce tampon.

## Exemples

Voici un exemple de l'opcode ins. Il utilise le fichier *ins.csd* [examples/ins.csd].

### Exemple 448. Exemple de l'opcode ins.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -idac    ;;realtime audio I/O
; For Non-realtime ouput leave only the line below:
; ins.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2    ;two channels out
0dbfs = 1

instr 1

ainl, ainr ins    ;grab your mic and sing
adel linseg 0, p3*.5, 0.02, p3*.5, 0 ;max delay time = 20ms
aoutl flanger ainl, adel, .7
aoutr flanger ainl, adel*2, .8
fout "in_s.wav", 14, aoutl, aoutr ;write to stereo file,
outs aoutl, aoutr    ;16 bits with header

endin
</CsInstruments>
```

```
<CsScore>  
  
i 1 0 10  
e  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*diskin, in, inh, inh, ino, inq, soundin mp3in,*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

# insremot

**insremot** — Un opcode que l'on peut utiliser pour implémenter un orchestre distant. Cet opcode envoie des événements de note d'une machine source vers une machine de destination.

## Description

Avec les opcodes *insremot* et *insglobal*, il est possible d'exécuter des instruments sur des machines distantes et de les contrôler depuis une machine maître. Les opcodes distants sont implémentés suivant le modèle maître/client. Toutes les machines concernées contiennent le même orchestre mais seule la machine maître possède l'information de la partition. Durant l'exécution, la machine maître envoie les événements de note aux clients. L'opcode *insremot* envoie des événements d'une machine source à une machine de destination. Pour envoyer des événements à plusieurs destinations (diffusion), on utilise l'opcode *insglobal*. Ces deux opcodes peuvent être utilisés en combinaison.

## Syntaxe

```
insremot idestination, isource, instrnum [,instrnum...]
```

## Initialisation

*idestination* -- une chaîne représentant l'ordinateur client (par exemple 192.168.0.100). C'est l'hôte de destination qui reçoit les événements de l'instrument donné.

*isource* -- une chaîne représentant l'ordinateur serveur (par exemple 192.168.0.100). C'est l'hôte source qui génère les événements pour l'instruments donné et les envoie à l'adresse donnée par *idestination*.

*instrnum* -- liste des numéros des instruments qui seront joués sur la machines destinataire.

## Exécution



### Note

Il est essentiel que les ordinateurs qui utilisent cet opcode aient les mêmes ordre des octets, taille des données (double ou float) et taille de pointeur. On ne peut pas l'utiliser par exemple en mélangeant des ordinateurs 32 bit et 64 bit.



### Note

Cet opcode peut utiliser en interne les fonctions *gethostname* et *gethostbyname* pour déterminer les adresses IP du client et du serveur afin de tester quels messages sont destinés à quelle machine, ou d'autres techniques sur les systèmes différents de Windows. Si un ordinateur a plus d'une adresse IP il n'y a aucun moyen de contrôler quelle adresse IP est choisie (mais voir ci-dessous). Sur les systèmes différents de Windows, l'interface réseau par défaut est eth0 ou en0, et si cela échoue wlan0. Depuis la version 6.05, on peut indiquer l'interface réseau à utiliser avec la variable d'environnement CS\_ETHER.



### Note

L'opération à distance ne permet pas du tout l'envoi de chaînes de caractères.

## Exemples

Voici un exemple de l'opcode `insremot`. Il utilise les fichiers `insremot.csd` [examples/insremot.csd] et `insremotM.csd` [examples/insremotM.csd].

### Exemple 449. Exemple de l'opcode `insremot`.

L'exemple ci-dessous montre l'exemple *barmodel* joué sur une machine distante. La machine maître et le client sont respectivement nommés "192.168.1.100" et "192.168.1.101". Démarrer le client sur sa machine (il attendra de recevoir des événements de la machine maître), puis démarrer le maître. Sur un système linux on démarre un client avec la commande (`csound -+rtaudio=alsa -odac -dm0 insremot.csd`), tandis que la commande sur la machine maître ressemble à ceci (`csound -+rtaudio=alsa -odac -dm0 insremotM.csd`).

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>

<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o insremot.wav -W ;; for file output any platform
</CsOptions>

<CsInstruments>
nchnls = 1

insremot "192.168.1.100", "192.168.1.101", 1

instr 1
  aq barmodel 1, 1, p4, 0.001, 0.23, 5, p5, p6, p7
  out      aq
endin

</CsInstruments>

<CsScore>
f0 360

e
</CsScore>

</CsoundSynthesizer>

<CsoundSynthesizer>

<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o insremotM.wav -W ;; for file output any platform
</CsOptions>

<CsInstruments>
nchnls = 1

insremot "192.168.1.100", "192.168.1.101", 1
```

```
instr 1
  aq barmodel 1, 1, p4, 0.001, 0.23, 5, p5, p6, p7
  out      aq
endin

</CsInstruments>

<CsScore>
i1 0 0.5 3 0.2 500 0.05
i1 0.5 0.5 -3 0.3 1000 0.05
i1 1.0 0.5 -3 0.4 1000 0.1
i1 1.5 4.0 -3 0.5 800 0.05
e
</CsScore>

</CsoundSynthesizer>
```

## Voir aussi

*insglobal, midglobal, midremot, remoteport*

## Crédits

Auteur : Simon Schampijer  
2006

Nouveau dans la version 5.03

# insglobal

**insglobal** — Un opcode que l'on peut utiliser pour implémenter un orchestre distant. Cet opcode envoie des évènements de note d'une machine source vers plusieurs destinations.

## Description

Avec les opcodes *insremot* et *insglobal*, il est possible d'exécuter des instruments sur des machines distantes et de les contrôler depuis une machine maître. Les opcodes distants sont implémentés suivant le modèle maître/client. Toutes les machines concernées contiennent le même orchestre mais seule la machine maître possède l'information de la partition. Durant l'exécution, la machine maître envoie les évènements de note aux clients. L'opcode *insglobal* envoie les évènements à toutes les machines impliquées dans le concert à distance. Ces machines sont déterminées par les définitions *insremot* placées avant la commande *insglobal*. Pour envoyer des évènements à une seule machine on utilise *insremot*.

## Syntaxe

```
insglobal isource, instrnum [,instrnum...]
```

## Initialisation

*isource* -- une chaîne représentant l'ordinateur serveur (par exemple 192.168.0.100). C'est l'hôte source qui génère les évènements pour le ou les instruments donnés et les envoie à toute les machines impliquées dans le concert à distance.

*instrnum* -- liste des numéros des instruments qui seront joués sur les machines destinataires.

## Exécution



### Note

Il est essentiel que les ordinateurs qui utilisent cet opcode aient les mêmes ordre des octets, taille des données (double ou float) et taille de pointeur. On ne peut pas l'utiliser par exemple en mélangeant des ordinateurs 32 bit et 64 bit.



### Note

Cet opcode utilise en interne les fonctions *gethostname* et *gethostbyname* pour déterminer les adresses IP du client et du serveur afin de tester quels messages sont destinés à chaque machine. Si un ordinateur a plus d'une adresse IP il n'y a aucun moyen de contrôler quelle adresse IP est choisie.



### Note

L'opération à distance ne permet pas du tout l'envoi de chaînes de caractères.

## Exemples

Voir l'entrée *insremot* pour un exemple d'utilisation.

## Voir aussi

*insremot*, *midglobal*, *midremot*, *remoteport*

## Crédits

Auteur : Simon Schampijer  
2006

Nouveau dans la version 5.03



# instr

instr — Commence un bloc d'instrument.

## Description

Commence un bloc d'instrument.

## Syntaxe

```
instr i, j, ...
```

## Initialisation

Commence un bloc d'instrument, définissant les instruments *i, j, ...*

*i, j, ...* doivent être des nombres, pas des expressions. Tout entier positif convient, dans n'importe quel ordre, mais on préfère éviter les nombres excessivement grands.



### Note

Il peut y avoir n'importe quel nombre de blocs d'instrument dans un orchestre.

On peut définir les instruments dans n'importe quel ordre (mais ils seront toujours initialisés et exécutés par ordre de numéro d'instrument ascendant, à l'exception des notes provoquées par des événements en temps réel, qui sont initialisées dans l'ordre où elles sont reçues mais toujours exécutées par ordre de numéro d'instrument ascendant). Les blocs d'instruments ne peuvent pas être imbriqués (un bloc ne peut pas en contenir un autre).

## Exemples

Voici un exemple de l'opcode *instr*. Il utilise le fichier *instr.csd* [examples/instr.csd].

### Exemple 450. Exemple de l'opcode *instr*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o instr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  iamp = 10000
  icps = 440
  iphs = 0

  a1 oscils iamp, icps, iphs
  out a1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*endin, in, out, opcode, endop, setksmps, xin, xout, subinstr, subinstrinit*

## Crédits

Exemple écrit par Kevin Conder.

# int

int — Extrait la partie entière d'un nombre décimal.

## Description

Retourne la partie entière de  $x$ .

## Syntaxe

`int(x)` (taux-i ou taux-k ; fonctionne aussi au taux-a dans Csound5)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode int. Il utilise le fichier *int.csd* [examples/int.csd].

### Exemple 451. Exemple de l'opcode int.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o int.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

icount init 0
loop:
  inum = icount / 3
  inm = int(inum)
  prints "integer (%f/3) = %f\\n", icount, inm
  loop_lt icount, 1, 10, loop

endin
</CsInstruments>
<CsScore>

i 1 0 0
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
integer (0.000000/3) = 0.000000
integer (1.000000/3) = 0.000000
integer (2.000000/3) = 0.000000
integer (3.000000/3) = 1.000000
integer (4.000000/3) = 1.000000
integer (5.000000/3) = 1.000000
integer (6.000000/3) = 2.000000
integer (7.000000/3) = 2.000000
integer (8.000000/3) = 2.000000
integer (9.000000/3) = 3.000000
```

## Voir aussi

*abs, exp, frac, log, log10, i, sqrt*

## Crédits

# integ

integ — Modifie un signal par intégration.

## Description

Modifie un signal par intégration.

## Syntaxe

```
ares integ asig [, iskip]  
kres integ ksig [, iskip]
```

## Initialisation

*iskip* (facultatif) --état initial de l'espace mémoire interne (voir *reson*). La valeur par défaut est 0.

## Exécution

*integ* et *diff* réalisent l'intégration et la différentiation d'un signal de contrôle ou audio en entrée. Ils sont mutuellement inverses et l'application des deux reconstruit le signal original. Comme ces unités sont des cas particuliers de filtres passe-bas et passe-haut, ils produisent une sortie pondérée (et modifiée en phase) en fonction de la fréquence. Ainsi *diff* d'un sinus produit un cosinus dont l'amplitude vaut  $2 * \pi * \text{Hz} / \text{sr}$  de l'original (pour chaque partiel) ; *integ* affectera inversement les amplitudes de ses composants en entrée. Sachant cela, ces unités peuvent fournir d'utiles modifications de signal.

## Exemples

Voici un exemple de l'opcode *integ*. Il utilise le fichier *integ.csd* [examples/integ.csd].

### Exemple 452. Exemple de l'opcode *integ*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac      ;;RT audio out  
;-iadc      ;;uncomment -iadc if RT audio input is needed too  
; For Non-realtime ouput leave only the line below:  
; -o integ.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 32  
nchnls = 2  
0dbfs = 1  
  
instr 1  
  
asig disk12 "fox.wav", 1
```

```

        outs asig, asig

    endin

    instr 2 ; with diff

    asig diskin2 "fox.wav", 1
    ares diff asig
        outs ares, ares

    endin

    instr 3 ; with integ

    asig diskin2 "fox.wav", 1
    aint integ asig
    aint = aint*.05 ;way too loud
        outs aint, aint

    endin

    instr 4 ; with diff and integ

    asig diskin2 "fox.wav", 1
    ares diff asig
    aint integ ares
        outs aint, aint

    endin

</CsInstruments>
<CsScore>

i 1 0 1
i 2 1 1
i 3 2 1
i 4 3 1

e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*diff, downsamp, interp, samphold, upsamp*

# interleave

interleave — Entrelace des tableaux dans un tableau unique en plaçant les données dans des positions alternées.

## Description

Prend des tableaux en entrée et entrelace leurs données (en plaçant les valeurs dans des positions alternées).

## Syntaxe

```
kout[] interleave kin1[], kin2[]
```

## Exécution

*kout[]* -- tableau en sortie contenant les données entrelacées. Il est créé s'il n'existe pas.

*kin1[], kin2[]* -- tableaux en entrée contenant les valeurs à entrelacer.

## Exemples

Voici un exemple de l'opcode interleave. Il utilise le fichier *interleave.csd* [exemples/interleave.csd].

### Exemple 453. Exemple de l'opcode interleave.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-n
</CsOptions>
<CsInstruments>

instr 1

kin1[] fillarray 1,2,3,4
kin2[] fillarray 5,6,7,8

kInt[] interleave kin1, kin2

printf "inputs: \n%d %d %d %d \n%d %d %d %d\n", 1,
      kin1[0], kin1[1], kin1[2], kin1[3],
      kin2[0], kin2[1], kin2[2], kin2[3]

printf "interleaved:\n%d %d %d %d %d %d %d %d\n", 1,
      kInt[0], kInt[1], kInt[2], kInt[3],
      kInt[4], kInt[5], kInt[6], kInt[7]

endin

</CsInstruments>
<CsScore>
i1 0 1
e
</CsScore>
```

`</CsoundSynthesizer>`

## Voir aussi

Vectorial opcodes, array opcodes

## Crédits

Auteur : Victor Lazzarini  
NUI Maynooth  
2018

Nouveau dans la version 6.12



# interp

*interp* — Convertit un signal de contrôle en signal audio avec interpolation linéaire.

## Description

Convertit un signal de contrôle en signal audio avec interpolation linéaire.

## Syntaxe

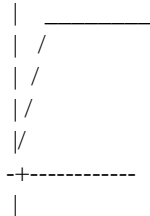
```
ares interp ksig [, iskip] [, imode] [, ivalue]
```

## Initialisation

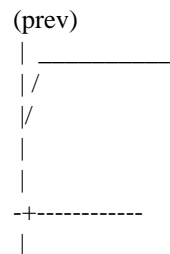
*iskip* (facultatif, 0 par défaut) -- s'il n'est pas nul, l'initialisation de l'espace mémoire interne est ignorée (voir *reson*).

*imode* (facultatif, 0 par défaut) -- donne à la valeur de sortie initiale la valeur de la première entrée de taux-*k* au lieu de zéro. Les graphes suivants montre la sortie de *interp* avec une valeur d'entrée constante, en mode original, en ignorant l'initialisation et avec le nouveau mode :

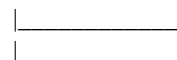
### Exemple 454. iskip=0, imode=0

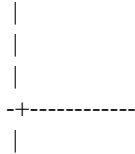


### Exemple 455. iskip=1, imode=0



### Exemple 456. iskip=0, imode=1





*ivalue* (facultatif, 0 par défaut) -- valeur initiale si à la fois *imode* et *iskip* valent zéro.

## Exécution

*ksig* -- signal de taux-k en entrée.

*interp* convertit un signal de contrôle en signal audio. Il utilise l'interpolation linéaire entre les valeurs successives de *ksig*.

## Exemples

Voici un exemple de l'opcode *interp*. Il utilise le fichier *interp.csd* [examples/interp.csd].

### Exemple 457. Exemple de l'opcode *interp*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o interp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 8000
kr = 8
ksmps = 1000
nchnls = 1

; Instrument #1 - a simple instrument.
instr 1
; Create an amplitude envelope.
kamp linseg 0, p3/2, 20000, p3/2, 0

; The amplitude envelope will sound rough because it
; jumps every ksmps period, 1000.
a1 oscil kamp, 440, 1
out a1
endin

; Instrument #2 - a smoother sounding instrument.
instr 2
; Create an amplitude envelope.
kamp linseg 0, p3/2, 25000, p3/2, 0
aamp interp kamp

; The amplitude envelope will sound smoother due to
; linear interpolation at the higher a-rate, 8000.
```

```
    a1 oscil aamp, 440, 1
    out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 256 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*diff, downsamp, integ, samphold, upsamp*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/issue10/CsoundRates.html>, écrit par Andrés Cabrera.

## Crédits

Mis à jour en novembre 2002, grâce à une note de Rasmus Ekman et Istvan Varga.

# invalue

`invalue` — Lit un signal de taux-k ou de taux-i ou une chaîne de caractères depuis un canal défini par l'utilisateur.

## Description

Lit un signal de taux-k ou de taux-i ou une chaîne de caractères depuis un canal défini par l'utilisateur.

## Syntaxe

```
invalue invalue "channel name"
```

```
kvalue invalue "channel name"
```

```
Sname invalue "channel name"
```

## Exécution

*invalue*, *kvalue* -- La valeur lue depuis le canal.

*Sname* -- La variable chaîne de caractères lue depuis le canal.

*"channel name"* -- Un entier, une chaîne de caractères (entre guillemets), ou une variable chaîne de caractères identifiant le canal.

## Exemples

Voici un exemple de l'opcode `invalue`. Il utilise le fichier *invalue.csd* [examples/invalue.csd].

### Exemple 458. Exemple de l'opcode `invalue`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
;run this example in CsoundQt, a Csound editor that provides widgets
;make the Widgets-panel visible, by clicking the Widgets symbol in the menu or pressing (Alt+1).

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
; written by Andres Cabrera
instr 1

kfreq invalue "freq" ; Quotes are needed here
asig oscil 0.1, kfreq, 1
outs asig, asig

endin
</CsInstruments>
```

```
<CsScore>
f 1 0 1024 10 1 ;sine
i 1 0 300 ;play for 300 seconds
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*outvalue*

## Crédits

Auteur : Matt Ingalls

Version de taux-i ajoutée dans Csound 6.04

# inx

*inx* — Lit des données audio sur 16 canaux depuis un périphérique externe ou un flot.

## Description

Lit des données audio sur 16 canaux depuis un périphérique externe ou un flot.



### Avertissement

Cet opcode est prévu pour ne fonctionner qu'avec des orchestres qui ont *nchnls*=16. Avec des orchestres dont *nchnls* > 16, la sortie audio ne sera pas correcte.

## Syntaxe

```
ar1, ar2, ar3, ar4, ar5, ar6, ar7, ar8, ar9, ar10, ar11, ar12, \  
ar13, ar14, ar15, ar16 inx
```

## Exécution

*inx* lit un signal audio sur 16 canaux depuis un périphérique externe ou un flot. Si l'option de ligne de commande *-i* est positionnée, le son est lu en continu depuis le flot audio en entrée (par exemple *stdin* ou un fichier son) dans un tampon interne.

## Voir aussi

*in32*, *inch*, *inz*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.07 de Csound

# inz

*inz* — Lit des échantillons audio multi-canaux depuis un périphérique externe ou un flot vers un tableau ZAK.

## Description

Lit des échantillons audio multi-canaux depuis un périphérique externe ou un flot vers un tableau ZAK.

## Syntaxe

```
inz ksig1
```

## Exécution

*inz* lit des échantillons audio sur *nchnls* vers un tableau ZAK commençant à *ksig1*. Si l'option de ligne de commande *-i* est positionnée, le son est lu en continu depuis le flot audio en entrée (par exemple *stdin* ou un fichier son) dans un tampon interne.

## Voir aussi

*in32*, *inch*, *inx*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.07 de Csound

# JackoAudioIn

JackoAudioIn — Reçoit un signal audio depuis un port de Jack.



## Note

Greffon : nécessite le greffon *jacko*

## Description

Reçoit un signal audio depuis un port d'entrée audio de Jack dans cette instance de Csound, qui a elle-même reçu le signal depuis sa connexion à un port de sortie externe de Jack.

## Syntaxe

```
asignal JackoAudioIn ScsoundPortName
```

## Initialisation

*ScsoundPortName* -- Le nom abrégé ("portname") du port d'entrée audio interne de Jack.

## Exécution

*asignal* -- Signal audio reçu depuis le port de sortie externe de Jack auquel *ScsoundPortName* est connecté.

## Voir aussi

*JackoInfo*, *JackoInfo*, *JackoFreewheel*, *JackoAudioOutConnect*, *JackoAudioOutConnect*, *JackoMidiInConnect*, *JackoMidiOutConnect*, *JackoOn*, *JackoAudioOut*, *JackoMidiOut*, *JackoNoteOut*, *JackoTransport*.

## Crédits

Par : Michael Gogins 2010



# JackoAudioInConnect

JackoAudioInConnect — Crée une connexion audio depuis un port de Jack vers Csound.



## Note

Greffon : nécessite le greffon *jacko*

## Description

Depuis l'en-tête de l'orchestre, crée une connexion audio depuis un port de sortie externe de Jack vers un port d'entrée audio de Jack dans l'instance de Csound.

## Syntaxe

```
JackoAudioInConnect SexternalPortName, ScsoundPortName
```

## Initialisation

*SexternalPortName* -- Le nom complet ("clientname:portname") d'un port de sortie audio externe de Jack.

*ScsoundPortName* -- Le nom abrégé ("portname") du port d'entrée audio interne de Jack.

## Exécution

Le signal audio doit être lu par l'opcode *JackoAudioIn*.

## Voir aussi

*JackoInfo*, *JackoInfo*, *JackoFreewheel*, *JackoAudioOutConnect*, *JackoMidiInConnect*, *JackoMidiOutConnect*, *JackoOn*, *JackoAudioIn*, *JackoAudioOut*, *JackoMidiOut*, *JackoNoteOut*, *JackoTransport*.

## Crédits

Par : Michael Gogins 2010

# JackoAudioOut

JackoAudioOut — Envoie un signal audio vers un port de Jack.



## Note

Greffon : nécessite le greffon *jacko*

## Description

Envoie un signal audio vers un port de sortie audio interne de Jack, qui lui-même l'envoie vers son port d'entrée audio externe de Jack connecté.

Notez qu'il est possible d'envoyer le flot audio vers l'interface audio du système via Jack, tandis qu'on l'envoie en même temps dans un fichier de sortie de Csound.

## Syntaxe

```
JackoAudioOut ScsoundPortName, asignal
```

## Initialisation

*ScsoundPortName* -- Le nom abrégé ("portname") du port de sortie audio interne de Jack.

## Exécution

*asignal* -- Signal audio à envoyer vers le port d'entrée audio externe de Jack auquel *CsoundPortName* est connecté.

Les signaux audio provenant de plusieurs instances de l'opcode routant vers le même port de Jack sont additionnés avant l'envoi.

## Voir aussi

*JackoInfo*, *JackoInfo*, *JackoFreewheel*, *JackoAudioOutConnect*, *JackoMidiInConnect*, *JackoMidiOutConnect*, *JackoOn*, *JackoAudioIn*, *JackoMidiOut*, *JackoMidiOut*, *JackoTransport*.

## Crédits

Par : Michael Gogins 2010

# JackoAudioOutConnect

JackoAudioOutConnect — Creates an audio connection from Csound to a Jack port.



## Note

Greffon : nécessite le greffon *jacko*

## Description

Dans l'en-tête de l'orchestre, crée une connexion audio depuis un port de sortie de Jack dans cette instance de Csound vers un port d'entrée audio externe de Jack.

## Syntaxe

```
JackoAudioOutConnect ScsoundPortName, SexternalPortName
```

## Initialisation

*ScsoundPortName* -- Le nom abrégé ("portname") du port audio de sortie interne de Jack.

*SexternalPortName* -- Le nom complet ("clientname:portname") d'un port audio d'entrée externe de Jack.

## Exécution

Le signal audio doit être écrit avec l'opcode *JackoAudioOut*.

## Voir aussi

*The Jacko Opcodes*, *JackoInfo*, *JackoInfo*, *JackoFreewheel*, *JackoAudioOutConnect*, *JackoMidiInConnect*, *JackoMidiOutConnect*, *JackoOn*, *JackoAudioIn*, *JackoAudioOut*, *JackoMidiOut*, *JackoNoteOut*, *JackoTransport*.

## Crédits

Par : Michael Gogins 2010

# JackoFreewheel

JackoFreewheel — Active ou désactive le mode roue libre de Jack.



## Note

Greffon : nécessite le greffon *jacko*

## Description

Active ou désactive le mode roue libre de Jack.

Lorsque le mode roue libre est activé, s'il est supporté par le reste du système Jack, Csound s'exécutera aussi vite que possible, ce qui peut être aussi bien plus rapide que moins rapide que le .

Ceci est essentiel pour restituer vers un fichier son, sans à coups et sans perte, des partitions qui sont trop denses pour une exécution en .

## Syntaxe

```
JackoFreewheel [ienabled]
```

## Initialisation

*ienabled* -- Active le mode roue libre (par défaut) ou le désactive.

## Voir aussi

*JackoInit*, *JackoInfo*, *JackoAudioInConnect*, *JackoAudioOutConnect*, *JackoMidiInConnect*, *JackoMidiOutConnect*, *JackoOn*, *JackoAudioIn*, *JackoAudioOut*, *JackoMidiOut*, *JackoNoteOut*, *JackoTransport*.

## Crédits

Par : Michael Gogins 2010

# JackoInfo

JackoInfo — Affiche de l'information sur le système Jack.



## Note

Greffon : nécessite le greffon *jacko*

## Description

Affiche les noms du démon et du client Jack, le taux d'échantillonnage et le nombre de trames par période, ainsi que les noms de tous les ports de Jack actifs, leur type, leur état et les connexions.

## Syntaxe

`JackoInfo`

## Initialisation

Peut être appelé n'importe quel nombre de fois dans l'en-tête de l'orchestre, par exemple une fois avant et une fois après la création de ports de Jack.

## Exemples

Voici un exemple de l'opcode JackoInfo. Il utilise le fichier *JackInfo.csd* [examples/JackoInfo.csd].

### Exemple 459. Exemple de l'opcode JackoInfo.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-n
</CsOptions>
<CsInstruments>

sr      = 48000
ksmps   = 128
nchnls  = 2
0dbfs   = 1

instr 1

JackoInit "default", "csound"
JackoInfo

endin
</CsInstruments>
<CsScore>

i 1 0 0
e
</CsScore>
```

`</CsoundSynthesizer>`

## Voir aussi

*JackoInit, JackoFreewheel, JackoAudioInConnect, JackoAudioOutConnect, JackoMidiInConnect, JackoMidiOutConnect, JackoOn, JackoAudioIn, JackoAudioOut, JackoMidiOut, JackoNoteOut, JackoTransport.*

## Crédits

Par : Michael Gogins 2010

# JackoInit

JackoInit — Initialise Csound comme client de Jack.



## Note

Greffon : nécessite le greffon *jacko*

## Description

Initialise cette instance de Csound comme client de Jack.

Le *sr* de Csound doit être égal au taux de trames par seconde du démon Jack.

Le *ksmps* de Csound doit être égal au taux de trames par période du démon Jack.

Le nombre de trames par période doit non seulement (a) être une puissance de 2, mais aussi (b) être un diviseur du nombre de trames par seconde, par exemple 128 trames par période divise 375 fois 48000 trames par seconde, pour une latence ou une granularité temporelle du MIDI d'environ 2.7 ms (aussi bien voire mieux que les meilleures performance humaines absolues).

L'ordre de traitement de tous les signaux allant des ports d'entrée de Jack aux ports de sortie de Jack en passant par Csound, doit être correctement déterminé par la suite des définitions d'instruments et d'opcodes dans Csound.

## Syntaxe

```
JackoInit ServerName, SclientName
```

## Initialisation

*SclientName* -- Le nom du client de Jack ; doit être normalement "csound".

*ServerName* -- Le nom du démon Jack ; normalement c'est "default".

Il faut appeler cet opcode une et une seule fois dans l'en-tête de l'orchestre, et avant tout autre opcode Jack. Si plus d'une instance de Csound utilisent les opcodes Jack en même temps, chaque instance de Csound doit utiliser un nom de client différent.

## Exemples

Voici un exemple de l'opcode JackoInit. Il utilise le fichier *JackoInit.csd* [exemples/JackoInit.csd].

### Exemple 460. Exemple de l'opcode JackoInit.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
-n
```

```

</CsOptions>
<CsInstruments>

  sr = 48000
  ksmpr = 128
  nchnls = 2
  odbfs = 1

  instr 1

  JackoInit "default", "csound"
  JackoInfo

  endin
</CsInstruments>
<CsScore>

  i 1 0 0
  e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*JackoInfo, JackoFreewheel, JackoAudioInConnect, JackoAudioOutConnect, JackoMidiInConnect, JackoMidiOutConnect, JackoOn, JackoAudioIn, JackoAudioOut, JackoMidiOut, JackoNoteOut, JackoTransport.*

## Crédits

Par : Michael Gogins 2010



# JackoMidiInConnect

JackoMidiInConnect — Crée une connexion MIDI depuis un port de Jack vers Csound.



## Note

Greffon : nécessite le greffon *jacko*

## Description

Dans l'en-tête de l'orchestre, crée une connexion MIDI depuis un port de sortie MIDI externe de Jack vers cette instance de Csound.

## Syntaxe

```
JackoMidiInConnect SexternalPortName, ScsoundPortName
```

## Initialisation

*SexternalPortName* -- Le nom complet ("clientname:portname") d'un port de sortie MIDI externe de Jack.

*ScsoundPortName* -- Le nom abrégé ("portname") du port d'entrée MIDI interne de Jack.

Doit être utilisé de concert avec les options `-M0 --rtmidi=null` de la ligne de commande de Csound. On peut l'utiliser avec les options de ligne de commande et/ou les opcodes d'interopérabilité MIDI pour permettre l'utilisation de définitions ordinaires d'instruments de Csound afin de restituer des partitions externes ou des séquences MIDI.

Notez que Csound peut se connecter à des ports ALSA en passant par Jack, mais que dans ce cas il faut identifier le port par son alias dans l'affichage de *JackInfo*.

## Exécution

Les événements MIDI seront reçus de la manière habituelle dans Csound, c'est-à-dire par un pilote MIDI et par le mécanisme de détection d'évènement plutôt que par un opcode de port d'entrée de Jack.

La granularité temporelle est celle de la période-k de Csound.

## Voir aussi

*JackoInfo*, *JackoInfo*, *JackoFreewheel*, *JackoAudioOutConnect*, *JackoAudioOutConnect*, *JackoMidiOutConnect*, *JackoOn*, *JackoAudioIn*, *JackoAudioOut*, *JackoMidiOut*, *JackoNoteOut*, *JackoTransport*.

## Crédits

Par : Michael Gogins 2010

# JackoMidiOutConnect

JackoMidiOutConnect — Crée une connexion MIDI de Csound à un port de Jack.



## Note

Greffon : nécessite le greffon *jacko*

## Description

Dans l'en-tête de l'orchestre, crée une connexion depuis un port de sortie MIDI de Jack dans cette instance de Csound à un port d'entrée MIDI externe de Jack.

## Syntaxe

```
JackoMidiOutConnect ScsoundPortName, SexternalPortName
```

## Initialisation

*ScsoundPortName* -- Le nom abrégé ("portname") du port de sortie MIDI interne de Jack.

*SexternalPortName* -- Le nom complet ("clientname:portname") d'un port d'entrée MIDI externe de Jack.

## Exécution

Les données MIDI doivent être écrites avec les opcodes *JackoMidiOut* ou *JackoNoteOut*.

## Voir aussi

*JackoInfo JackoInfo JackoFreewheel JackoAudioOutConnect JackoMidiInConnect JackoMidiOutConnect JackoOn JackoAudioIn JackoAudioOut JackoMidiOut JackoNoteOut JackoTransport*

## Crédits

Par : Michael Gogins 2010

# JackoMidiOut

JackoMidiOut — Envoie un message de canal MIDI vers un port de Jack.



## Note

Greffon : nécessite le greffon *jacko*

## Description

Envoie un message de canal MIDI à un port de sortie MIDI de Jack dans cette instance de Csound qui l'envoie au port d'entrée MIDI externe de Jack auquel il est connecté.

## Syntaxe

```
JackoMidiOut ScsoundPortName, kstatus, kchannel, kdata1[, kdata2]
```

## Initialisation

*ScsoundPortName* -- Le nom abrégé ("portname") du port de sortie MIDI interne de Jack.

## Exécution

*kstatus* -- octet d'état MIDI ; doit indiquer un message de canal MIDI.

*kchannel* -- canal MIDI (de 0 à 15).

*kdata1* -- Premier octet de donnée d'un message de canal MIDI.

*kdata2* -- Second octet de donnée facultatif d'un message de canal MIDI.

Cet opcode peut être appelé n'importe quel nombre de fois dans la même période-k. Les messages venant de plusieurs instances d'un opcode qui les envoient sur le même port sont rassemblés avant l'envoi.

L'état courant, les messages système exclusif et les messages temps réel ne sont pas supportés.

La granularité temporelle est celle de la période-k de Csound.

## Voir aussi

*JackoInfo*, *JackoInfo*, *JackoFreewheel*, *JackoAudioOutConnect*, *JackoMidiInConnect*, *JackoMidiOutConnect*, *JackoOn*, *JackoAudioIn*, *JackoNoteOut*, *JackoTransport*.

## Crédits

Par : Michael Gogins 2010

# JackoNoteOut

JackoNoteOut — Envoie un message de canal MIDI vers un port de Jack.



## Note

Greffon : nécessite le greffon *jacko*

## Description

Envoie un message de canal MIDI à un port de sortie MIDI de Jack dans cette instance de Csound qui l'envoie au port d'entrée MIDI externe de Jack auquel il est connecté.

## Syntaxe

```
JackoNoteOut ScsoundPortName, kstatus, kchannel, kdata1[, kdata2]
```

## Initialisation

*ScsoundPortName* -- Le nom abrégé ("portname") du port de sortie MIDI interne de Jack.

## Exécution

*kstatus* -- octet d'état MIDI ; doit indiquer un message de canal MIDI.

*kchannel* -- canal MIDI (de 0 à 15).

*kdata1* -- Premier octet de donnée d'un message de canal MIDI.

*kdata2* -- Second octet de donnée facultatif d'un message de canal MIDI.

Cet opcode peut être appelé n'importe quel nombre de fois dans la même période-k. Les messages venant de plusieurs instances d'un opcode qui les envoie sur le même port sont rassemblés avant l'envoi.

L'état courant, les messages système exclusif et les messages temps réel ne sont pas supportés.

La granularité temporelle est celle de la période-k de Csound.

## Voir aussi

*JackoInfo*, *JackoInfo*, *JackoFreewheel*, *JackoAudioInConnect*, *JackoAudioOutConnect*, *JackoMidiInConnect*, *JackoMidiOutConnect*, *JackoOn*, *JackoAudioIn*, *JackoMidiOut*, *The Jacko Opcodes*.

## Crédits

Par : Michael Gogins 2010

# JackoOn

JackoOn — Active ou désactive tous les ports de Jack.



## Note

Greffon : nécessite le greffon *jacko*

## Description

Dans l'en-tête de l'orchestre, une fois que toutes les connexions Jack ont été créées, active ou désactive tous les opcodes d'entrée et de sortie Jack dans cette instance de Csound pour lire ou écrire des données.

## Syntaxe

```
JackoOn [iactive]
```

## Initialisation

*iactive* -- Un indicateur qui active les ports (par défaut) ou les désactive.

## Voir aussi

*JackoInit*, *JackoFreewheel*, *JackoAudioInConnect*, *JackoAudioOutConnect*, *JackoMidiInConnect*, *JackoMidiOutConnect*, *JackoAudioIn*, *JackoAudioOut*, *JackoMidiOut*, *JackoNoteOut*, *JackoTransport*.

## Crédits

Par : Michael Gogins 2010

# JackoTransport

JackoTransport — Contrôle le transport de Jack.



## Note

Greffon : nécessite le greffon *jacko*

## Description

Démarre, stoppe ou repositionne le transport de Jack. C'est utile, par exemple, pour démarrer un séquenceur externe qui enverra des messages MIDI à Csound.

## Syntaxe

```
JackoTransport kcommand, [kposition]
```

## Exécution

*kcommand* -- 0 signifie "aucune action", 1 démarre le transport, 2 stoppe le transport et 3 positionne le transport à *kposition* secondes du début de l'exécution (c'est-à-dire la date 0 dans la partition).

*kposition* -- Temps auquel on veut positionner le transport en secondes depuis le début de l'exécution (c'est-à-dire la date 0 dans la partition).

On peut utiliser cet opcode à l'initialisation ou durant l'exécution.

La granularité temporelle est celle de la période-k de Csound.

## Voir aussi

*JackoInfo*, *JackoInfo*, *JackoFreewheel*, *JackoAudioOutConnect*, *JackoMidiInConnect*, *JackoMidiOutConnect*, *JackoOn*, *JackoAudioIn*, *JackoMidiOut*, *JackoNoteOut*.

## Crédits

Par : Michael Gogins 2010

# jacktransport

jacktransport — Démarre/arrête jack\_transport et peut optionnellement repositionner la tête de lecture.



## Note

Greffon : nécessite le greffon *jackTransport*

## Description

Opcode du greffon jackTransport.

Démarre/arrête jack\_transport et peut optionnellement repositionner la tête de lecture.

## Syntaxe

```
jacktransport icommand [, ilocation]
```

## Initialisation

*icommand* -- 1 pour démarrer, 0 pour arrêter.

*ilocation* -- position facultative en secondes pour indiquer où la tête de lecture doit aller. S'il est omis, le transport commence depuis la position courante.



## Note

Comme *jacktransport* dépend du kit de connexion audio de jack, il ne fonctionne que sur les systèmes Linux ou Mac OS X qui exécutent un serveur jack.

## Exemples

Voici un exemple simple de l'opcode jacktransport. Il utilise le fichier *jacktransport.csd* [exemples/jack-transport.csd].

### Exemple 461. Exemple simple de l'opcode jacktransport.

```
<CsoundSynthesizer>

<CsOptions>
+rtaudio=JACK -b 64 --sched -o dac:system:playback_
</CsOptions>

<CsInstruments>
sr          =      44100
ksmps      =      16
nchnls     = 2

instr 1
jacktransport p4, p5
endin
```

```
instr 2
jacktransport p4
endin

</CsInstruments>
<CsScore>

i2 0 5 1; play
i2 5 1 0; stop
i1 6 5 1 2 ; move at 2 seconds and start playing back
i1 11 1 0 0 ; stop and rewind

e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Cesare Marilungo

Nouveau dans la version 5.08



# jitter

`jitter` — Génère aléatoirement une suite de segments de droite.

## Description

Génère aléatoirement une suite de segments de droite.

## Syntaxe

```
kout jitter kamp, kcpsMin, kcpsMax
```

## Exécution

*kamp* -- Amplitude de la déviation de jitter

*kcpsMin* -- Vitesse minimale des variations aléatoires de fréquence (exprimée en cps)

*kcpsMax* -- Vitesse maximale des variations aléatoires de fréquence (exprimée en cps)

*jitter* génère aléatoirement une suite de segments de droite entre *-kamp* et *+kamp*. La durée de chaque segment est une valeur aléatoire générée en fonction des valeurs *kcpsmin* et *kcpsmax*.

On peut utiliser *jitter* pour donner plus de naturel et une « touche analogique » à des sons statiques et monotones. Pour de meilleurs résultats il est conseillé de garder une amplitude modérée.

## Exemples

Voici un exemple de l'opcode jitter. Il utilise le fichier *jitter.csd* [examples/jitter.csd].

### Exemple 462. Exemple de l'opcode jitter.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o jitter.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kamp      init p4
kcpsmin   init 4
kcpsmax   init 8
```

```
kj2  jitter kamp, kcpsmin, kcpsmax
aout pluck 1, 200+kj2, 1000, 0, 1
aout dcblock aout ;remove DC
     outs aout, aout

endin
</CsInstruments>
<CsScore>

i 1 0 15 2 ;a bit jitter
i 1 8 15 10 ;some more
i 1 16 15 20 ;lots more
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*jitter2, vibr, vibrato*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.15

# jitter2

jitter2 — Génère aléatoirement une suite de segments de droite contrôlables par l'utilisateur.

## Description

Génère aléatoirement une suite de segments de droite contrôlables par l'utilisateur.

## Syntaxe

```
kout jitter2 ktotamp, kamp1, kcps1, kamp2, kcps2, kamp3, kcps3[ ,iopt]
```

## Initialisation

*iopt* -- Facultatif, contrôle la date de début des effets. S'il est nul (valeur par défaut) la sortie est nulle jusqu'à la fin de la valeur de *kcps* la plus courte. S'il est différent de zéro l'effet démarre immédiatement.

## Exécution

*ktotamp* -- Amplitude résultante de jitter2

*kamp1* -- Amplitude du premier composant de jitter

*kcps1* -- Vitesse de la variation aléatoire du premier composant de jitter (exprimée en cps)

*kamp2* -- Amplitude du second composant de jitter

*kcps2* -- Vitesse de la variation aléatoire du second composant de jitter (exprimée en cps)

*kamp3* -- Amplitude du troisième composant de jitter

*kcps3* -- Vitesse de la variation aléatoire du troisième composant de jitter (exprimée en cps)

*jitter2* génère une ligne segmentée comme *jitter*, mais ici le résultat est semblable à la somme de trois opcodes *randi*, chacun avec ses propres valeurs d'amplitude et de fréquence (voir *randi* pour plus de détails), qui sont modifiables au taux-k. On peut obtenir différents effets en variant les arguments en entrée.

On peut utiliser *jitter2* pour donner plus de naturel et une « touche analogique » à des sons statiques et monotones. Pour de meilleurs résultats il est conseillé de garder une amplitude modérée.

## Exemples

Voici un exemple de l'opcode jitter2. Il utilise le fichier *jitter2.csd* [examples/jitter2.csd].

### Exemple 463. Exemple de l'opcode jitter2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>
```

```

; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o jitter2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ktotamp init p4
kamp1 init .5
kcps1 init 10
kamp2 init .5
kcps2 init 2
kamp3 init .5
kcps3 init 3

kj2 jitter2 ktotamp, kamp1, kcps1, kamp2, kcps2, kamp3, kcps3
aout pluck 1, 200+kj2, 1000, 0, 1
aout dcblock aout
outs aout, aout

endin
</CsInstruments>
<CsScore>

i 1 0 15 2 ;a bit jitter
i 1 8 15 10 ;some more
i 1 16 15 20 ;lots more
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*jitter, vibr, vibrato*

## Crédits

Auteur : Gabriel Maldonado, John ffitich

Nouveau dans la version 4.15

iopt ajouté dans la version 6.08

# joystick

joystick — Lit les données provenant d'un joystick.

## Description

Opcodes du greffon joystick.

Lit les données provenant d'un joystick sous Linux.

## Syntaxe

```
kres joystick kdevice ktab
```

## Exécution



### Note

Prière de noter que pour le moment cet opcode ne fonctionne que sous Linux.

*kdevice* -- l'index de périphérique du joystick, soit /dev/js*N*, soit /dev/input/js*N*.

*ktab* -- Une table pour recevoir les valeurs en entrée, devant pouvoir stocker au moins une valeur par axe du joystick, une valeur pour chaque bouton, plus deux valeurs. Les deux premiers éléments de la table sont initialisés avec le nombre d'axes et le nombre de boutons, respectivement, lorsqu'un joystick est ouvert. Si un joystick est débranché durant l'exécution, l'opcode va tenter de réouvrir le périphérique de manière répétitive, avec un délai entre chaque essai.

## Exemples

Voici un exemple de l'opcode joystick. Il utilise le fichier *joystick.csd* [examples/joystick.csd].

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;;realtime audio out
;-iadc     ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o joystick.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
;0dbfs = 1

instr 1 ; gives information about your joystick in real time

kmask    joystick    0, 1
kidx     = 2
```

```

kaxes    tab 0, 1 ; number of axes has been stored in position 0
kbuttons tab 1, 1 ; number of buttons has been stored in position 1
printf "this joystick has %d axes, %d buttons\n", kidx, kaxes, kbuttons
kuniq    init 0

reportaxis:; first we see if we have any x/y input
kcheck   = kmask & (1<<kidx)
if kcheck == 0 kgoto nexta
kres     tab kidx, 1
kuniq    = kuniq + 1 ; to be sure to make the printf print
printf   "axis %d, value %6d\n", kuniq, kidx-2, kres
nexta:
kidx     = kidx+1
if kidx < (kaxes+2) kgoto reportaxis

reportbutton:; now we check for any buttons pressed
kcheck   = kmask & 1<<kidx
if kcheck == 0 kgoto nextb
kres     tab kidx, 1 ; a button has been pressed, get from table
printf   "button %d, pushed\n", kidx*kres, (kidx-(kaxes+2))
printf   "button %d, released\n", kidx*(1-kres), (kidx-(kaxes+2))

nextb:
kidx     = kidx+1
if kidx < (kaxes+kbuttons+2) kgoto reportbutton

endin
</CsInstruments>
<CsScore>

f1 0 32 7 0 7 0 ; will hold the joystick data

i1 0 60000

e
</CsScore>
</CsSoundSynthesizer>

```

Voici un autre exemple de l'opcode joystick. Il utilise le fichier *joystick-2.csd* [examples/joystick-2.csd].

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac ;;realtime audio out
;-iadc ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o joystick-2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
;0dbfs = 1

instr 1

kmask joystick 0, 1
kaxes init 0
kbuttons init 0
kx0 init 0 ; first two entries are # of axes and # of buttons,
ky0 init 0 ; then axes, then buttons
vtabk 0, 1, kaxes, kbuttons, kx0, ky0
kidx = 2+kaxes

buttons:

```

```

    kcheck = kmask & 1<<kidx ; if the button was just now pressed and...
    kres   tab    kidx, 1    ; if button value is one, start a note
          schedkwhen kres*kcheck, 1, 20, 2, 0, 60000, kidx, kx0, ky0
    kidx   = kidx+1
    if kidx < (kaxes+kbuttons+2) kgoto buttons

    endin

    instr 2 ; play a tone until the button is released

    kstop  tab p4, 1 ; when this button is released, we fade out
    ihz    init cpsoct(((p5+32767)/9362)+5) ; ~ 30 hz to 4khz
    print ihz
    ito    init ampdb(((p6+32767)/2184)+60) ; ~ 60 - 90 db
    kenv   init 0
    kdelta init ito/(kr*10)
    if kstop == 1 kgoto output
    if kdelta < 0 kgoto output
    kdelta  = kdelta*-1

    output:
        kenv = kenv+delta
        kenv limit kenv, 0, ito
        aout oscils 1, ihz, 0
        aout = kenv*aout
        outs aout, aout
    if kenv != 0 kgoto noexit
    if kdelta > 0 kgoto noexit
    turnoff
    noexit:

    endin
</CsInstruments>
<CsScore>
f1 0 32 7 0 7 0 ; will hold the joystick data

i1 0 60000

e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Justin Glenn Smith  
2010

Nouveau dans la version 5.17.12

# jspline

jspline — Un générateur de spline avec gigue.

## Description

Un générateur de spline avec gigue.

## Syntaxe

```
ares jspline xamp, kcpsMin, kcpsMax
```

```
kres jspline kamp, kcpsMin, kcpsMax
```

## Exécution

*kres, ares* -- Signal de sortie.

*xamp* -- Facteur d'amplitude.

*kcpsMin, kcpsMax* -- Intervalle de définition du taux de génération des points. Les limites minimale et maximale sont exprimées en Hz.

*jspline* (générateur de spline avec gigue) génère une courbe lisse basée sur des points aléatoires engendrés au taux [*cpsMin, cpsMax*]. Cet opcode est semblable à *randomi* ou à *randi* ou à *jitter*, toutefois les segments ne sont pas des lignes droites, mais des courbes splines cubiques. Les valeurs de sortie sont approximativement comprises entre *-xamp* et *xamp*. Dans la réalité, l'intervalle peut être un peu plus grand, à cause des courbes d'interpolation entre chaque paire de points aléatoires.

Actuellement les courbes générées sont assez lisses quand *cspMin* n'est pas trop différent de *cpsMax*. Quand l'intervalle *cpsMin-cpsMax* est grand, quelques petites discontinuités peuvent se produire, mais, dans la plupart des cas, cela ne devrait pas poser de problème. L'algorithme sera peut-être amélioré dans les prochaines versions.

Ces opcodes sont souvent meilleurs que *jitter* lorsque l'on veut un rendu « naturel » ou « analogique » de sons numériques. On peut aussi les utiliser dans la composition algorithmique, pour générer des lignes mélodiques aléatoires lisses lors d'une utilisation conjointe avec l'opcode *samphold*.

Noter que le résultat est assez différent de celui que l'on obtiendrait en filtrant un bruit blanc, et que l'on peut ainsi obtenir un contrôle bien plus précis.

## Exemples

Voici un exemple de l'opcode *jspline*. Il utilise le fichier *jspline.csd* [exemples/jspline.csd].

### Exemple 464. Exemple de l'opcode *jspline*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>
```



```

; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o jspline.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kamp      init p4
kcpsmin   init 2
kcpsmax   init 20

ksp  jspline kamp, kcpsmin, kcpsmax
aout pluck 1, 200+ksp, 1000, 0, 1
aout dcblock aout ;remove DC
      outs aout, aout

endin
</CsInstruments>
<CsScore>

i 1 0 10 2 ;a bit jitter
i 1 8 10 10 ;some more
i 1 16 10 20 ;lots more
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la Version 4.15

# k

k — Convertit un paramètre de taux-i en une valeur de taux-k. Ou convertit une valeur de taux-a en une valeur de taux-k par sous-échantillonnage.

## Description

Convertit une valeur de taux-i en une valeur de taux-k, par exemple pour une utilisation avec *rnd()* et *birnd()* pour générer des nombres aléatoires au taux-k.

## Syntaxe

**k**(x) (arguments de taux-i seulement)

**k**(x) (arguments de taux-a seulement)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Voir aussi

*i a*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/issue10/CsoundRates.html>, écrit par Andrés Cabrera.

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 5.00 de Csound

Auteur : John ffitich

Conversion de taux-a en taux-k ajoutée dans la version 6.00 de Csound.

# K35\_hpf

K35\_hpf — Implémentation du filtre passe-haut résonant du Korg35 avec rétroaction sans retard.

## Description

Implémentation du filtre passe-haut résonant du Korg35 avec rétroaction sans retard. On trouve ce type de filtre sur les Korg MS10 et les premiers MS20.

## Syntaxe

```
asig K35_hpf ain, xcf, xQ [, inlp, isaturation, istor]
```

## Initialisation

*istor* --état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal de sortie.

*ain* -- signal d'entrée.

*xcf* -- fréquence de coupure du filtre (taux-i-, k ou a).

*xQ* -- Q du filtre (taux-i-, k ou a). Dans l'intervalle 1.0-10.0 (lié à l'opcode). L'auto-oscillation survient à 10.0.

*knlp* (facultatif, 0 par défaut) -- méthode de traitement non linéaire. 0 = pas de traitement, 1 = traitement non linéaire. La méthode 1 utilise  $\tanh(ksaturation * input)$ . L'activation du TNL peut pousser la sortie globale du filtre au-delà de l'unité et doit être compensée pour l'environnement externe du filtre.

*ksaturation* (facultatif, 1 par défaut) -- quantité de saturation à utiliser pour le traitement non linéaire. Les valeurs > 1 augmentent la pente de la courbe de TNL.

## Exemples

Voici un exemple de l'opcode K35\_hpf. Il utilise le fichier *k35.csd* [examples/k35.csd].

### Exemple 465. Exemple de l'opcode K35\_hpf.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
; =====
<CsInstruments>

sr = 48000
ksmps = 1
nchnls = 2
0dbfs = 1
```

```

;; test instruments to demo filter cutoff sweep with high resonance

instr 1

asig = vco2(0.5, cps2pch(6.00, 12))
asig = K35_lpf(asig, expseg:a(10000, p3, 30), 9.9, 0, 1)
asig *= 0.25
asig = limit(asig, -1.0, 1.0)

outc(asig, asig)

endin

instr 2

asig = vco2(0.5, cps2pch(6.00, 12))
asig = K35_lpf(asig, expseg:k(10000, p3, 30), 9.9, 0, 1)
asig *= 0.25
asig = limit(asig, -1.0, 1.0)

outc(asig, asig)

endin

instr 3

asig = vco2(0.5, cps2pch(6.00, 12))
asig = K35_hpf(asig, expseg:a(10000, p3, 30), 9.9, 0, 1)
asig *= 0.25
asig = limit(asig, -1.0, 1.0)

outc(asig, asig)

endin

instr 4

asig = vco2(0.5, cps2pch(6.00, 12))
asig = K35_hpf(asig, expseg:k(10000, p3, 30), 9.9, 0, 1)
asig *= 0.25
asig = limit(asig, -1.0, 1.0)

outc(asig, asig)

endin

;; beat instruments

instr ms20_drum

ipch = cps2pch(p4, 12)
iamp = ampdbfs(p5)
aenv = expseg:a(10000, 0.05, ipch, p3 - .05, ipch)

asig = rand:a(-1.0, 1.0)
asig = K35_hpf(asig, 60, 7, 1, 1)
asig = K35_lpf(asig, aenv, 9.8, 1, 1)

asig = tanh(asig * 16)

asig *= expon(iamp, p3, 0.0001)

outc(asig, asig)

endin

```

```

instr ms20_bass
  ipch = cps2pch(p4, 12)
  iamp = ampdbfs(p5)
  aenv = expseg(1000, 0.1, ipch * 2, p3 - .05, ipch * 2)

  asig = vco2(1.0, ipch)
  asig = K35_hpf(asig, ipch, 5, 0, 1)
  asig = K35_lpf(asig, aenv, 8, 0, 1)

  asig *= expon:a(iamp, p3, 0.0001) * 0.8

  outc(asig, asig)
endin

;; perf code

gktempo init 122

opcode beat_dur,i,0
  xout 60 / i(gktempo)
endop

instr bass_player
  idur = beat_dur() / int(random(1,3))
  ipch = 6.00 + int(random(1,3)) + int(random(1,3)) / 100

  schedule("ms20_bass", 0, idur, ipch, -11)

  if(p2 < 37.5) then
    schedule("bass_player", idur, 0.1)
  endif
  turnoff
endin

instr beat_player
  istep_total = p4
  istep = istep_total % 16

  if(istep % 4 == 0) then
    ipch = ((istep_total % 128) < 112) ? 4.00 : 8.00
    iamp = (istep == 0) ? -9 : -12
    schedule("ms20_drum", 0, 0.5, ipch, iamp)
  endif

  schedule("ms20_drum", 0, 0.125, 14.00,
    (istep % 4 == 0) ? -12 : -18)

  if(p2 < 37.5) then
    schedule("beat_player", beat_dur() / 4, 0.1, istep_total + 1)
  endif
  turnoff
endin

;; start play of beats

instr start_beats
  schedule("beat_player", 0, 0.1, 0)
  schedule("bass_player", 0, 0.1)
endin

</CsInstruments>
; =====
<CsScore>
i1 0 5.0
i2 5 5.0

```

```
i3 10 5.0
i4 15 5.0

i "start_beats" 22 0.5 0

</CsScore>
</CsoundSynthesizer>
```

## Références

Ce filtre est basé sur les travaux de Will Pirkle qui emploie le travail de Vadim Zavalishin pour créer des implémentations de filtres analogiques à transformation préservant la topologie (TPT), avec des transformations bilinéaires.

1. Pirkle, Will. Designing Software Synthesizer Plug-ins in C++: For RackAFX, VST3, and Audio Units. CRC Press, 2014.
2. Pirkle, Will. AN-7A: Virtual Analog (VA) Korg35 Highpass Filter v2.0 Simplified. 2013.
3. Zavalishin, Vadim. "The Art of VA filter design." Native Instruments, 2012.

## Crédits

Auteur : Steven Yi  
Avril 2017

Nouveau dans Csound 6.09.0

# K35\_lpf

K35\_lpf — Implémentation du filtre passe-bas résonant du Korg35 avec rétroaction sans retard.

## Description

Implémentation du filtre passe-bas résonant du Korg35 avec rétroaction sans retard. On trouve ce type de filtre sur les Korg MS10, les premiers MS20 et les séries Monotron.

## Syntaxe

```
asig K35_lpf ain, xcf, xQ [, inlp, isaturation, istor]
```

## Initialisation

*istor* --état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal de sortie.

*ain* -- signal d'entrée.

*xcf* -- fréquence de coupure du filtre (taux-i-, k ou a).

*xQ* -- Q du filtre (taux-i-, k ou a). Dans l'intervalle 1.0-10.0 (lié à l'opcode). L'auto-oscillation survient à 10.0.

*knlp* (facultatif, 0 par défaut) -- méthode de traitement non linéaire. 0 = pas de traitement, 1 = traitement non linéaire. La méthode 1 utilise  $\tanh(k_{\text{saturation}} * \text{input})$ . L'activation du TNL peut pousser la sortie globale du filtre au-delà de l'unité et doit être compensée pour l'environnement externe du filtre.

*ksaturation* (facultatif, 1 par défaut) -- quantité de saturation à utiliser pour le traitement non linéaire. Les valeurs > 1 augmentent la pente de la courbe de TNL.

## Exemples

Voici un exemple de l'opcode K35\_lpf. Il utilise le fichier *k35.csd* [examples/k35.csd].

### Exemple 466. Exemple de l'opcode K35\_lpf.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
; =====
<CsInstruments>

sr = 48000
ksmps = 1
nchnls = 2
0dbfs = 1
```

```

;; test instruments to demo filter cutoff sweep with high resonance

instr 1

asig = vco2(0.5, cps2pch(6.00, 12))
asig = K35_lpf(asig, expseg:a(10000, p3, 30), 9.9, 0, 1)
asig *= 0.25
asig = limit(asig, -1.0, 1.0)

outc(asig, asig)

endin

instr 2

asig = vco2(0.5, cps2pch(6.00, 12))
asig = K35_lpf(asig, expseg:k(10000, p3, 30), 9.9, 0, 1)
asig *= 0.25
asig = limit(asig, -1.0, 1.0)

outc(asig, asig)

endin

instr 3

asig = vco2(0.5, cps2pch(6.00, 12))
asig = K35_hpf(asig, expseg:a(10000, p3, 30), 9.9, 0, 1)
asig *= 0.25
asig = limit(asig, -1.0, 1.0)

outc(asig, asig)

endin

instr 4

asig = vco2(0.5, cps2pch(6.00, 12))
asig = K35_hpf(asig, expseg:k(10000, p3, 30), 9.9, 0, 1)
asig *= 0.25
asig = limit(asig, -1.0, 1.0)

outc(asig, asig)

endin

;; beat instruments

instr ms20_drum

ipch = cps2pch(p4, 12)
iamp = ampdbfs(p5)
aenv = expseg:a(10000, 0.05, ipch, p3 - .05, ipch)

asig = rand:a(-1.0, 1.0)
asig = K35_hpf(asig, 60, 7, 1, 1)
asig = K35_lpf(asig, aenv, 9.8, 1, 1)

asig = tanh(asig * 16)

asig *= expon(iamp, p3, 0.0001)

outc(asig, asig)

endin

```



```

instr ms20_bass
  ipch = cps2pch(p4, 12)
  iamp = ampdbfs(p5)
  aenv = expseg(1000, 0.1, ipch * 2, p3 - .05, ipch * 2)

  asig = vco2(1.0, ipch)
  asig = K35_hpf(asig, ipch, 5, 0, 1)
  asig = K35_lpf(asig, aenv, 8, 0, 1)

  asig *= expon:a(iamp, p3, 0.0001) * 0.8

  outc(asig, asig)
endin

;; perf code

gktempo init 122

opcode beat_dur,i,0
  xout 60 / i(gktempo)
endop

instr bass_player
  idur = beat_dur() / int(random(1,3))
  ipch = 6.00 + int(random(1,3)) + int(random(1,3)) / 100

  schedule("ms20_bass", 0, idur, ipch, -11)

  if(p2 < 37.5) then
    schedule("bass_player", idur, 0.1)
  endif
  turnoff
endin

instr beat_player
  istep_total = p4
  istep = istep_total % 16

  if(istep % 4 == 0) then
    ipch = ((istep_total % 128) < 112) ? 4.00 : 8.00
    iamp = (istep == 0) ? -9 : -12
    schedule("ms20_drum", 0, 0.5, ipch, iamp)
  endif

  schedule("ms20_drum", 0, 0.125, 14.00,
    (istep % 4 == 0) ? -12 : -18)

  if(p2 < 37.5) then
    schedule("beat_player", beat_dur() / 4, 0.1, istep_total + 1)
  endif
  turnoff
endin

;; start play of beats

instr start_beats
  schedule("beat_player", 0, 0.1, 0)
  schedule("bass_player", 0, 0.1)
endin

</CsInstruments>
; =====
<CsScore>
i1 0 5.0
i2 5 5.0

```

```
i3 10 5.0
i4 15 5.0

i "start_beats" 22 0.5 0

</CsScore>
</CsoundSynthesizer>
```

## Références

Ce filtre est basé sur les travaux de Will Pirkle qui emploie le travail de Vadim Zavalishin pour créer des implémentations de filtres analogiques à transformation préservant la topologie (TPT), avec des transformations bilinéaires.

1. Pirkle, Will. Designing Software Synthesizer Plug-ins in C++: For RackAFX, VST3, and Audio Units. CRC Press, 2014.
2. Pirkle, Will. AN-5: Virtual Analog (VA) Korg35 Lowpass Filter v3.5. 2013.
3. Zavalishin, Vadim. "The Art of VA filter design." Native Instruments, 2012.

## Crédits

Auteur : Steven Yi  
Avril 2017

Nouveau dans Csound 6.09.0

# kgoto

kgoto — Transfère le contrôle lors des phases d'exécution.

## Description

Transfère le contrôle sans condition vers l'instruction étiquetée par *label*, lors des phases d'exécution seulement.

## Syntaxe

```
kgoto label
```

où *label* se trouve dans le même bloc d'instrument et n'est pas une expression.

## Exemples

Voici un exemple de l'opcode kgoto. Il utilise le fichier *kgoto.csd* [examples/kgoto.csd].

### Exemple 467. Exemple de l'opcode kgoto.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o kgoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Change kval linearly from 0 to 2 over
; the period set by the third p-field.
kval line 0, p3, 2

; If kval is greater than or equal to 1 then play the high note.
; If not then play the low note.
if (kval >= 1) kgoto highnote
    kgoto lownote

highnote:
    kfreq = 880
    goto playit

lownote:
    kfreq = 440
```

```

    goto playit

playit:
    ; Print the values of kval and kfreq.
    printks "kval = %f, kfreq = %f\\n", 1, kval, kfreq

    a1 oscil 10000, kfreq, 1
    out a1
endin

</CsInstruments>
<CsScore>

; Table #1: a simple sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

kval = 0.000000, kfreq = 440.000000
kval = 0.999732, kfreq = 440.000000
kval = 1.999639, kfreq = 880.000000

```

## Voir aussi

*cggoto, cigoto, ckgoto, goto, if, igoto, tigoto, timeout*

## Crédits

Exemple écrit by Kevin Conder.

# kr

kr — Fixe le taux de contrôle.

## Description

Ces instructions sont des *affectations* de valeurs globales réalisées au début d'un orchestre, avant que tout bloc d'instrument ne soit défini. Leur fonction est de fixer certaines *variables* dont le nom est un mot réservé et qui sont nécessaires à l'exécution. Une fois fixés, ces mots réservés peuvent être utilisés dans des expressions n'importe où dans l'orchestre.

## Syntaxe

```
kr = iarg
```

## Initialisation

*kr* = (facultatif) -- fixe le taux de contrôle à *iarg* échantillons par seconde. La valeur par défaut est 4410.

De plus, toute *variable globale* peut être initialisée par une *instruction de la période d'initialisation* n'importe où avant la première *instruction instr.* Toutes les affectations ci-dessus sont exécutées dans l'instrument 0 (passe-i seulement) au début de l'exécution réelle.

Depuis la version 3.46 de Csound, on peut omettre *kr*. Csound utilisera les valeurs par défaut, ou calculera *kr* à partir des valeurs définies de *ksmps* et *sr*. Habituellement, il est mieux de ne spécifier que *ksmps* et *sr* et de laisser csound calculer *kr*.

## Exemples

```
sr = 10000
kr = 500
ksmps = 20
gil = sr/2.
ga init 0
itranspose = octpch(.01)
```

## Voir aussi

*ksmps*, *nchnls*, *nchnls\_i*, *sr*

# ksmps

ksmps — Fixe le nombre d'échantillons dans une période de contrôle.

## Description

Ces instructions sont des *affectations* de valeurs globales réalisées au début d'un orchestre, avant que tout bloc d'instrument ne soit défini. Leur fonction est de fixer certaines *variables* dont le nom est un mot réservé et qui sont nécessaires à l'exécution. Une fois fixés, ces mots réservés peuvent être utilisés dans des expressions n'importe où dans l'orchestre.

## Syntaxe

```
ksmps = iarg
```

## Initialisation

*ksmps* = (facultatif) -- fixe le nombre d'échantillons dans une période de contrôle. Cette valeur doit être égale à *sr/kr*. La valeur par défaut est 10.

De plus, toute *variable globale* peut être initialisée par une *instruction de la période d'initialisation* n'importe où avant la première *instruction instr*. Toutes les affectations ci-dessus sont exécutées dans l'instrument 0 (passe-i seulement) au début de l'exécution réelle.

Depuis la version 3.46 de Csound, on peut omettre *ksmps*. Csound essaiera de calculer la valeur omise à partir des valeurs spécifiées pour *sr* et *kr*, mais le résultat devra être un nombre entier.



### Avertissement

ksmps doit avoir une valeur entière.

## Exemples

```
sr = 10000
kr = 500
ksmps = 20
gil = sr/2.
ga init 0
itranspose = octpch(.01)
```

## Voir aussi

*kr, nchnls, nchnls\_i, sr*

## Crédits

Grâce à une note de Gabriel Maldonado, un avertissement sur les valeurs entières a été ajouté.

# lenarray

lenarray — Evalue la taille ou le nombre de dimensions d'un tableau.

## Description

Evalue la taille ou le nombre de dimensions d'un tableau.

## Syntaxe

```
ir lenarray karray[, iwhich]
kr lenarray karray[, iwhich]
```

## Initialisation

*karray* -- tableau à interroger. Peut avoir n'importe quelle dimension.

*iwhich* -- sélectionne la dimension dont on veut évaluer la taille. S'il est inférieur ou égal à zéro, le nombre de dimensions est retourné. Vaut 1 par défaut, comme pour un vecteur.

## Execution

*kr* -- longueur du vecteur.

*karray* -- tableau à interroger.

Si la dimension interrogée dépasse celles du tableau ou si le tableau n'est pas initialisé, la valeur -1 est retournée.

## Exemples

Voici un exemple de l'opcode lenarray. Il utilise le fichier *lenarray.csd* [examples/lenarray.csd].

### Exemple 468. Exemple de l'opcode lenarray.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-n -m128 ;no sound output, reduced messages
</CsOptions>
<CsInstruments>
;example by joachim heintz
sr = 44100
ksmps = 32
nchnls = 1
0dbfs = 1

instr 1 ;simple example
kArr[] fillarray 1, 2, 3, 4 ;fill array manually
printks "Length of kArr = %d\n\n", 0, lenarray(kArr) ;print out its length
turnoff ;only do this in the first k-cycle
```

```

    endin

    instr 2 ;random array length
iNumEls random 1, 11 ;create random number between 1 and 10
kArr[] init int(iNumEls) ;create array of this length
    printks "Random length of kArr = %d\n", 0, lenarray(kArr) ;print out
    turnoff

    endin

    instr 3 ;fill random array length with random elements
iNumEls random 1, 11 ;create random number between 1 and 10
kArr[] init int(iNumEls) ;create array of this length
    printks "Random length of kArr = %d\n", 0, lenarray(kArr) ;print out

;fill
kIndx = 0 ;initialize index
    until kIndx == lenarray(kArr) do
kArr[kIndx] rnd31 10, 0 ;set element to random value -10...10
kIndx += 1 ;increase index
    od

;print
kIndx = 0 ;initialize index
    until kIndx == lenarray(kArr) do
printf("kArr[%d] = %f\n", kIndx+1, kIndx, kArr[kIndx])
kIndx += 1 ;increase index
    od

    turnoff
    endin
</CsInstruments>
<CsScore>
i 1 0 .1
i 2 0 .1
i 2 0 .1
i 3 0 .1
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

Opcodes vectoriels

## Crédits

Auteur : John ffitch  
 Codemist Ltd  
 2013

Nouveau dans la version 6.00

Extension multi-dimensionnelle dans la version 6.04



# lfo

lfo — Un oscillateur basse fréquence avec différentes formes d'onde.

## Description

Un oscillateur basse fréquence avec différentes formes d'onde.

## Syntaxe

```
kres lfo kamp, kcps [, itype]
ares lfo kamp, kcps [, itype]
```

## Initialisation

*itype* (facultatif, par défaut 0) -- détermine la forme d'onde de l'oscillateur. La valeur par défaut est 0.

- *itype* = 0 - sinus
- *itype* = 1 - triangle
- *itype* = 2 - carrée (bipolaire)
- *itype* = 3 - carrée (unipolaire)
- *itype* = 4 - dent de scie
- *itype* = 5 - dent de scie (vers le bas)

L'onde sinus est implémentée comme une table de 4096 éléments avec interpolation linéaire. Les autres sont calculées.

## Exécution

*kamp* -- amplitude de la sortie

*kcps* -- fréquence de l'oscillateur

## Exemples

Voici un exemple de l'opcode lfo. Il utilise le fichier *lfo.csd* [examples/lfo.csd].

### Exemple 469. Exemple de l'opcode lfo.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;;realtime audio out
```

```

;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o lfo.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kcps = 5
itype = p4 ;lfo type

klfo line 0, p3, 20
al lfo klfo, kcps, itype
asig poscil .5, 220*al, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
; sine wave.
f 1 0 32768 10 1

i 1 0 3 0 ;lfo = sine
i 1 + 3 2 ;lfo = square
i 1 + 3 5 ;lfo = saw-tooth down
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch  
 University of Bath/Codemist Ltd.  
 Bath, UK  
 Novembre 1998

Nouveau dans la version 3.491 de Csound

# limit

limit — Fixe les limites inférieure et supérieure de la valeur à traiter.

## Description

Fixe les limites inférieure et supérieure de la valeur à traiter.

## Syntaxe

```
ares limit asig, klow, khigh  
ires limit isig, ilow, ihigh  
kres limit ksig, klow, khigh  
ires[] limit isig[], ilow, ihigh  
kres[] limit ksig[], klow, khigh
```

## Initialisation

*isig* -- signal d'entrée

*ilow* -- limite inférieure

*ihigh* -- limite supérieure

## Exécution

*xsig* -- signal d'entrée

*klow* -- limite inférieure

*khigh* -- limite supérieure

*limit* fixe les limites inférieure et supérieure pour la valeur du *xsig* qu'il traite. *ixhigh* est inférieur à *ixlow*, la sortie est la moyenne des deux - elle n'est pas affectée par *xsig*.

Cet opcode est utile dans plusieurs situations, telles que l'indexation de table ou pour l'écrêtage et le modelage de signaux de taux-a, de taux-i ou de taux-k.

## Exemples

Voici un exemple de l'opcode *limit*. Il utilise le fichier *limit.csd* [examples/limit.csd].

### Exemple 470. Exemple de l'opcode limit.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>
```

```

; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o limit.wav -W    ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
Odbfs = 1
nchnls = 2

instr    1 ; Limit / Mirror / Wrap

igain    = p4    ;gain
ilevl1   = p5    ; + level
ilevl2   = p6    ; - level
imode    = p7    ;1 = limit, 2 = mirror, 3 = wrap

ain      soundin "fox.wav"
ain      = ain*igain

if       imode = 1 goto limit
if       imode = 2 goto mirror

asig     wrap ain, ilevl2, ilevl1
goto     outsignal

limit:
asig     limit ain, ilevl2, ilevl1
goto     outsignal

mirror:
asig     mirror ain, ilevl2, ilevl1
outsigal:

outs     asig*.5, asig*.5    ;mind your speakers

endin

</CsInstruments>
<CsScore>

;          Gain  +Levl -Levl Mode
i1  0  3   4.00  .25  -1.00  1 ;limit
i1  4  3   4.00  .25  -1.00  2 ;mirror
i1  8  3   4.00  .25  -1.00  3 ;wrap
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*mirror, wrap*

## Crédits

Auteur : Robin Whittle  
Australie

Nouveau dans la version 3.46 de Csound

# limit1

limit1 — Fonction limiteur.

## Description

Limite la valeur de son argument à l'intervalle [0, 1].

## Syntaxe

```
iRes[] limit1 iarg
```

```
kres[] limit1 karg
```

## Initialisation

*iarg[]* -- l'argument.

## Exécution

*karg[]* -- l'argument.

## Exmples

Voici un exemple de l'opcode limit1. Il utilise le fichier *limit1.csd* [examples/limit1.csd].

### Exemple 471. Exemple de l'opcode limit1.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

instr 1

iArg1[] fillarray 1,2,3
iRes[] limit1 iArg1/2.5
ik init 0

while ik < lenarray(iRes) do
  print iRes[ik]
  ik += 1
od

endin

</CsInstruments>
<CsScore>
i1 0 0
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
2017

# line

line — Trace un segment de droite entre les points spécifiés.

## Description

Trace un segment de droite entre les points spécifiés.

## Syntaxe

```
ares line ia, idur, ib
```

```
kres line ia, idur, ib
```

## Initialisation

*ia* -- valeur initiale.

*ib* -- valeur après *idur* secondes.

*idur* -- durée en secondes du segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

## Exécution

*line* génère des signaux de contrôle ou des signaux audio dont les valeurs évoluent linéairement depuis une valeur initiale jusqu'à une valeur finale.



### Note

Une erreur habituelle avec cet opcode est de croire que la valeur de *ib* est tenue après la durée *idur*. *line* ne s'arrête pas automatiquement à la fin de la durée donnée. Si la longueur de votre note dépasse *idur* secondes, *kres* (ou *ares*) ne s'arrêtera pas sur *ib*, mais au contraire il continuera à monter ou à descendre à la même vitesse. Si l'on a besoin d'une pente ascendante (ou descendante) suivie d'une tenue il faut utiliser l'opcode *linseg*.

## Exemples

Voici un exemple de l'opcode *line*. Il utilise le fichier *line.csd* [examples/line.csd].

### Exemple 472. Exemple de l'opcode *line*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o line.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kp = p6
;choose between expon or line
if (kp == 0) then
    kpitch expon p4, p3, p5
elseif (kp == 1) then
    kpitch line p4, p3, p5
endif

asig vco2 .6, kpitch
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 2 300 600 0 ;if p6=0 then expon is used
i 1 3 2 300 600 1 ;if p6=1 then line is used
i 1 6 2 600 1200 0
i 1 9 2 600 1200 1
i 1 12 2 1200 2400 0
i 1 15 2 1200 2400 1
i 1 18 2 2400 30 0
i 1 21 2 2400 30 1
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*expon, expseg, expsegr, linseg, linsegr*



# linen

**linen** — Applique un motif constitué d'une attaque et d'une chute en segments de droite à un signal d'amplitude.

## Description

*linen* -- applique un motif constitué d'une attaque et d'une chute en segments de droite à un signal d'amplitude.

## Syntaxe

```
ares linen xamp, irise, idur, idec
```

```
kres linen kamp, irise, idur, idec
```

## Initialisation

*irise* -- durée de l'attaque en secondes. Une valeur nulle ou négative signifie pas d'attaque.

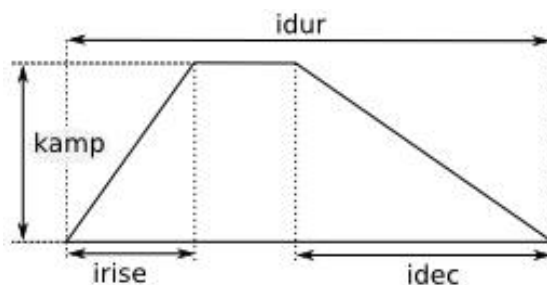
*idur* -- durée totale en secondes. Avec une valeur nulle ou négative, l'initialisation sera ignorée.

*idec* -- durée de la chute en secondes. Si *idec* > *idur* la chute sera tronquée.

## Exécution

*kamp*, *xamp* -- signal d'amplitude en entrée.

L'attaque est appliquée pendant les *irise* premières secondes, et la chute à partir de *idur* - *idec*. Si ces périodes sont séparées dans le temps il y aura un entretien durant lequel *amp* ne sera pas modifié. Si l'attaque et la chute de *linen* se chevauchent, les deux modifications agiront en même temps pendant cette période. Si la durée totale *idur* est dépassée pendant l'exécution, la chute continuera dans la même direction, devenant négative.



Enveloppe générée par l'opcode *linen*



### Note

Il est faux de croire que la valeur 0 sera tenue après la fin de l'enveloppe à *idur* secondes. *linen* ne se termine pas automatiquement à la fin de la durée donnée. Si la longueur de la note est supérieure à *idur* secondes, *kres* (ou *ares*) ne s'arrêtera pas à 0, mais continuera au

contraire à chuter à la même vitesse. Si l'on a besoin d'une chute suivie d'une valeur stable il vaut mieux utiliser l'opcode *linseg*.

## Exemples

Voici un exemple de l'opcode *linen*. Il utilise le fichier *linen.csd* [examples/linen.csd].

### Exemple 473. Exemple de l'opcode *linen*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o linen.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
; p4=amp
; p5=freq
; p6=attack time
; p7=release time
ares linen p4, p6, p3, p7
asig poscil ares, p5, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 4096 10 1 ; sine wave

;ins strt dur amp freq attack release
i1 0 1 .5 440 0.5 0.7
i1 1.5 1 .2 440 0.9 0.1
i1 3 1 .2 880 0.02 0.99
i1 4.5 1 .2 880 0.7 0.01
i1 6 3 .7 220 0.5 0.5
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*envlpx*, *envlpxr*, *linenr*

# linenr

linenr — L'opcode *linen* rallongé avec un segment de relâchement.

## Description

*linenr* -- comme *linen* sauf que le dernier segment n'est entamé qu'après la détection d'un relâchement de note MIDI. La note est alors rallongée de la durée de la chute.

## Syntaxe

```
ares linenr xamp, irise, idec, iatdec
```

```
kres linenr kamp, irise, idec, iatdec
```

## Initialisation

*irise* -- durée de l'attaque en secondes. Une valeur nulle ou négative signifie pas d'attaque.

*idec* -- durée de la chute en secondes. Si *idec* > *idur* la chute sera tronquée.

*iatdec* -- facteur d'atténuation par lequel la dernière valeur de l'entretien diminue exponentiellement pendant la chute. Cette valeur doit être positive et elle est normalement de l'ordre de 0,01. Une valeur trop longue ou excessivement courte peut produire une coupure audible. Les valeurs nulle ou négatives sont interdites.

## Exécution

*kamp*, *xamp* -- signal d'amplitude en entrée.

Ce qui rend unique *linenr* dans Csound c'est qu'il contient un *détecteur de note-off* et un *allongement de la durée de relâchement*. Lorsqu'il détecte la fin d'un évènement de partition ou un note-off MIDI, il allonge immédiatement la durée d'exécution de l'instrument courant de *idec* secondes, puis il exécute une chute exponentielle vers le facteur *iatdec*. S'il y a plusieurs unités dans un instrument, l'allongement est défini par le plus grand *idec*.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note off, comme *linsegr* et *expsegr*, ou bien l'on peut construire des enveloppes plus complexes au moyen de *xtratim* et de *release*. Noter qu'il n'est pas nécessaire d'utiliser *xtratim* avec *linenr*, car la durée est allongée automatiquement.

Ces unités « r » peuvent être modifiées également par des évènements MIDI note-off provoqués par une vélocité nulle.

## Exemples

Voici un exemple de l'opcode *linenr*. Il utilise le fichier *linenr.csd* [examples/linenr.csd].

### Exemple 474. Exemple de l'opcode *linenr*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc          -M0 ;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

; Example by Jonathan Murphy and Charles Gran 2007
sr      = 44100
ksmps   = 10
nchnls  = 2

; new, and important. Make sure that midi note events are only
; received by instruments that actually need them.

; turn default midi routing off
massign 0, 0
; route note events on channel 1 to instr 1
massign 1, 1

; Define your midi controllers
#define C1 #21#
#define C2 #22#
#define C3 #23#

; Initialize MIDI controllers
initc7   1, 21, 0.5      ;delay send
initc7   1, 22, 0.5      ;delay: time to zero
initc7   1, 23, 0.5      ;delay: rate

gaosc    init           0

; Define an opcode to "smooth" the MIDI controller signal
opcode smooth, k, k
kin      xin
kport    linseg         0, 0.0001, 0.01, 1, 0.01
kin      portk          kin, kport
xout     kin
endop

instr 1
; Generate a sine wave at the frequency of the MIDI note that triggered the instrument
ifqc     cpsmidi
iamp     ampmidi        10000
aenv     linenr         iamp, .01, .1, .01      ;envelope
a1       oscil          aenv, ifqc, 1
; All sound goes to the global variable gaosc
gaosc    = gaosc + a1
endin

instr 198 ; ECHO
kcbsnd   ctrl7          1, 21, 0, 1      ;delay send
ktime    ctrl7          1, 22, 0.01, 6    ;time loop fades out
kloop    ctrl7          1, 23, 0.01, 1    ;loop speed
; Receive MIDI controller values and then smooth them
kcbsnd   smooth         kcbsnd
ktime    smooth         ktime
kloop    smooth         kloop

imaxlpt  = 1            ;max loop time
; Create a variable reverberation (delay) of the gaosc signal
acomb    vcomb          gaosc, ktime, kloop, imaxlpt, 1
aout     = (acomb * kcbsnd) + gaosc * (1 - kcbsnd)
outs     aout, aout

```

```
gaosc      = 0
    endin

</CsInstruments>

<CsScore>
f1 0 16384 10 1
i198 0 10000
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*linsegr, expsegr, envlpxr, mxadsr, madsr, envlpx, linen, xtratim*

# lineto

lineto — Génère un glissando à partir d'un signal de contrôle.

## Description

Génère un glissando à partir d'un signal de contrôle.

## Syntaxe

```
kres lineto ksig, ktime
```

## Exécution

*kres* -- Signal de sortie.

*ksig* -- Signal d'entrée.

*ktime* -- Durée du glissando en secondes.

*lineto* ajoute un glissando (c-à-d des segments de droite) à un signal d'entrée en escalier (produit par exemple par *randh* ou par *lpshold*). Il génère un segment de droite allant d'un degré à l'autre en *ktime* secondes. Lorsque le degré suivant est atteint, cette valeur est maintenue jusqu'à ce qu'un nouveau degré apparaisse. Il faut s'assurer que la valeur de l'argument *ktime* est inférieure à l'intervalle de temps entre deux degrés consécutifs du signal original, sinon des discontinuités apparaîtront dans le signal de sortie.

Lorsqu'on l'utilise avec la sortie de *lpshold*, on obtient une simulation de l'effet de glissando des vieux synthétiseurs analogiques.



### Note

Une nouvelle valeur de *ksig* ou de *ktime* n'aura d'effet qu'après que la valeur précédente de *ktime* se soit écoulée.

## Exemples

Voici un exemple de l'opcode *lineto*. Il utilise le fichier *lineto.csd* [examples/lineto.csd].

### Exemple 475. Exemple de l'opcode *lineto*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o lineto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 2^10, 10, 1

instr 1

kfreq      randh      1000, 20, 2, 1, 2000 ;generates ten random number between 100 and 300 per second
kpan       randh      .5, 1, 2, 1, .5      ;panning between 0 and 1
kp         lineto      kpan, .5            ;smoothing pan transition
aout       poscil      .4, kfreq, giSine
aL, aR     pan2        aout, kp
           outs        aL, aR

endin
</CsInstruments>
<CsScore>

i 1 0 10
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*tlineto*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la Version 4.13

# linlin

linlin — Correspondance linéaire.

## Description

Opcode du greffon emugens.

Envoie un intervalle linéaires de valeurs vers un autre intervalle linéaire. Supporte les scalaires ainsi que les tableaux, aux taux-i et -k.

## Syntaxe

```
ky linlin kx, ky0, ky1 [, kx0, kx1 ]  
iy linlin ix, iy0, iy1 [, ix0, ix1 ]  
kys[] linlin kxs[], ky0, ky1 [, kx0, kx1 ]  
iys[] linlin ixs[], ky0, ky1, [ kx0, kx1 ]  
kC[] linlin kx, kA[], kB[] [, kx0, kx1 ]
```

## Exécution

**kx** -- Signal en entrée.

**kx0** -- Limite inférieure de l'intervalle d'entrée. 0 par défaut.

**kx1** -- Limite supérieure de l'intervalle d'entrée. 1 par défaut.

**ky0** -- Limite inférieure de l'intervalle de sortie.

**ky1** -- imite supérieure de l'intervalle de sortie.

$$y = (x - x_0) / (x_1 - x_0) * (y_1 - y_0) + y_0$$

## Exemples

Voici un exemple de l'opcode linlin. Il utilise le fichier *linlin.csd* [examples/linlin.csd].

### Exemple 476. Exemple de l'opcode linlin.

```
<CsoundSynthesizer>  
<CsOptions>  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 128  
nchnls = 2  
0dbfs = 1.0
```



```

; Example file for linlin.csd

/*

linlin

linear to linear interpolation between two ranges

ky    linlin kx, ky0, ky1, kx0=0, kx1=1
kys[] linlin kxs[], ky0, ky1, kx0=0, kx1=1
iys[] linlin ixs[], iy0, iy1, ix0=0, ix1=1

linlin can be also used to blend between two arrays:

kC[]  linlin kx, kA[], kB[], kx0=0, kx1=1

if kx==0.5, kC[] will hold the avg of kA[] and kB[] for each value
(assumes that kA[] and kB[] are arrays of the same size)

*/

; Map a value within the range 1-3 to the range 0-10.
instr 1
  kx line 1, p3, 3
  ky linlin kx, 0, 10, 1, 3
  printks "kx: %f   ky: %f \n", 1/kr, kx, ky
endin

; Map an array of values
instr 2
  kX[] fillarray 0, 0.5, 1, 1.5, 2
  trim kX, 4
  kY[] linlin kX, 0, 10, 0, 2
  printarray kY
  turnoff
endin

; Blend between two arrays
instr 3
  kA[] fillarray 0, 1, 2, 3, 4, 5
  kB[] fillarray 0, 2, 4, 6, 8, 10
  kx line 0, p3, 1
  kC[] linlin kx, kA, kB
  printarray kC, -1, "", "blend"
endin

</CsInstruments>
<CsScore>
; i 1 0 0.2
i 2 0.5 0.2
; i 3 1 0.5

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*lincos, bpf bpfcos scale, ntrpol,*

## Crédits

Par : Eduardo Moguillansky 2017

Nouveau greffon dans la version 6.11

# lincos

lincos — Interpolation linéaire vers cosinus.

## Description

Opcodes du greffon emugens.

Distribue un intervalle de valeurs linéaires vers un intervalle de valeurs cosinus (transitions douces). Ne supporte pour le moment que les scalaires, de taux-i ou -k (pas de tableaux).

## Syntaxe

```
ky lincos kx, ky0, ky1 [, kx0, kx1 ]
```

```
iy lincos ix, iy0, iy1 [, ix0, ix1 ]
```

## Exécution

**kx** -- Signal en entrée.

**ky0** -- Limite inférieure de l'intervalle en sortie.

**ky1** -- Limite supérieure de l'intervalle en sortie.

**kx0** -- Limite inférieure de l'intervalle en entrée (0 par défaut).

**kx1** -- Limite supérieure de l'intervalle en entrée (1 par défaut).

```
dx = ((x-x0) / (x1-x0)) * PI + PI  
y  = y0 + ((y1 - y0) * (1 + cos(dx)) / 2.0);
```

## Exemples

Voici un exemple de l'opcode lincos. Il utilise le fichier *lincos.csd* [examples/lincos.csd].

### Exemple 477. Exemple de l'opcode lincos.

```
<CsoundSynthesizer>  
<CsOptions>  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 128  
nchnls = 2  
0dbfs = 1.0  
  
; Example file for lincos.csd  
  
/*  
  lincos
```

```

similar to cosseg, but with an explicit input for time
lincos can be used ease-in / out any linear ramp

ky lincos kx, ky0, ky1, kx0=0, kx1=1
iy lincos ix, iy0, iy1, ix0=0, ix1=1

*/

seed 0

instr 1
; Map a value within the range 1-3 to the range 0-10.
iy lincos 1.5, 0, 10, 1, 3
print iy
kx line 1, p3, 3
ky lincos kx, 0, 10, 1, 3
printks "kx: %f   ky: %f \n", 1/kr, kx, ky
endin

instr 2
; lincos can be used to create amplitude or pitch envelopes
ktrig init 0
krnd dust 1, 1
ktrig = lineto(sc_trig(krnd & ~ktrig, 0.5), 1)
kpitch = lincos:k(ktrig, 60, 61)
a0 oscili 0.7, mtof(kpitch)

kfade lincos linsegr(0, 1.5, 1, 1.5, 0), 0, 1
kcresc lincos ktrig, 0.25, 1
outch 1, a0 * interp(kfade * kcresc)
endin

</CsInstruments>
<CsScore>

i 1 0 0.2
i 2 0 20

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*linlin, cosseg, scale, ntrpol, bpf*

## Crédits

Auteur : Eduardo Moguillansky 2018

Nouveau greffon dans la version 6.12

# link\_beat\_force

`link_beat_force` — Force la session du réseau global Ableton Link à adopter une pulsation et un temps spécifiques.

## Description

Opcodes du greffon `ableton_link_opcodes`.

Force la session du réseau global Ableton Link à adopter une pulsation et un temps spécifiques, comme si un chef stoppait son orchestre et le relançait immédiatement.

## Syntaxe

```
link_beat_force i_peer, k_beat [, k_at_time_seconds [, k_quantum ]]
```

## Initialisation

`i_peer` -- L'objet pair Ableton Link.

## Exécution

`k_beat` -- Le nombre requis pour la pulsation.

`k_at_time_seconds` -- Le temps requis pour la pulsation ; la valeur par défaut est le temps courant.

`k_quantum` -- La quantification locale de la pulsation ; la valeur par défaut est 1.

## Voir aussi

*Ableton Link Opcodes*

## Crédits

Par : Michael Gogins 2017

# link\_beat\_get

`link_beat_get` — Retourne la pulsation, la phase en fonction de la quantification locale et le temps courant de la session.

## Description

Opcodes du greffon `ableton_link_opcodes`.

Retourne la pulsation, la phase en fonction de la quantification locale et le temps courant de la session du réseau global Ableton Link.

## Syntaxe

```
k_beat_number, k_phase, k_current_time_seconds link_beat_get i_peer [, k_quantum]
```

## Initialisation

*i\_peer* -- L'objet pair Ableton Link.

## Exécution

*k\_quantum* -- La quantification locale de la pulsation ; la valeur par défaut est 1.

*k\_beat\_number* -- Le numéro de la pulsation courante.

*k\_phase* -- La phase courante de la pulsation en fonction de la quantification locale de la pulsation.

*k\_current\_time\_seconds* -- Le temps courant en fonction du moment où ce pair a été activé.

Penser à utiliser *link\_metro* si l'on a besoin d'un déclencheur périodique piloté par la pulsation de la session.

## Voir aussi

*Ableton Link Opcodes*

## Crédits

Par : Michael Gogins 2017

# link\_beat\_request

`link_beat_request` — Demande à la session du réseau global Ableton Link à adopter une pulsation et un temps spécifiques.

## Description

Opcodes du greffon `ableton_link_opcodes`.

Demande à la session du réseau global Ableton Link à adopter une pulsation et un temps spécifiques.

## Syntaxe

```
link_beat_request i_peer, k_beat [, k_at_time_seconds [, k_quantum ]]
```

## Initialisation

*i\_peer* -- L'objet pair Ableton Link.

## Performance

*k\_beat* -- Le nombre requis pour la pulsation.

*k\_at\_time\_seconds* -- Le temps requis pour la pulsation ; la valeur par défaut est le temps courant.

*k\_quantum* -- La quantification locale de la pulsation ; la valeur par défaut est 1.

## Voir aussi

*Ableton Link Opcodes*

## Crédits

Par : Michael Gogins 2017

# link\_create

link\_create — Crée un pair dans une session réseau Ableton Link.

## Description

Opcodes du greffon `ableton_link_opcodes`.

Crée un pair dans une session réseau Ableton Link. Le premier membre dans une session détermine le tempo initial de la session. La valeur retournée doit être passée comme premier paramètre à chaque appel consécutif d'un opcode Ableton Link pour ce pair.

## Syntaxe

```
i_peer link_create [i_bpm]
```

## Initialisation

*i\_bpm* -- Tempo initial de la session, en pulsations par minute. N'a aucun effet sauf si c'est le premier pair de la session. La valeur par défaut est 60.

*i\_peer* -- L'objet pair Ableton Link retourné pour être utilisé par les autres opcodes Ableton Link.

## Exemples

Voici un exemple de l'opcode `link_create`. Il utilise le fichier *ableton\_link.csd* [examples/ableton\_link.csd].

### Exemple 478. Exemple de l'opcode link\_create.

```
<CsoundSynthesizer>
<CsLicense>
Run the Ableton Link "LinkHut" example application, or some other
Ableton Link peer, while you run this CSD to see what happens.
</CsLicense>
<CsOptions>
-m0 -d -odac
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 10
nchnls = 2

alwayson "LinkMonitor"

gi_peer link_create 72
gk_beat init 0

instr Beep
asignal oscils 20000, p4, 0
outs asignal, asignal
endin

instr TempoChange
link_tempo_set gi_peer, 80
endin
```

```

instr LinkEnable
i_enabled = p4
link_enable gi_peer, i_enabled
endin

instr LinkMonitor
i_kperiod_seconds = ksmps / sr
printf_i "kperiod: %9.6f seconds.\n", 1, i_kperiod_seconds
printf_i "gi_peer: %g\n", 1, gi_peer
link_enable gi_peer, 1
k_trigger, gk_beat, k_phase, k_time link_metro gi_peer, 1
k_peers link_peers gi_peer
k_tempo link_tempo_get gi_peer
k_enabled link_is_enabled gi_peer
k_hz = 1000
if floor(gk_beat % 4) == 0 then
k_hz = 3000
else
k_hz = 2000
endif
schedkwhen k_trigger, 0, 1, "Beep", 0, 0.01, k_hz
printf "LinkMonitor: gi_peer: %g k_enabled: %9.4f k_trigger: %9.4f beat: %9.4f k_phase: %9.4f time: %9.4f\n", 1, gi_peer, k_enabled, k_trigger, gk_beat, k_phase, k_time
endin
</CsInstruments>
<CsScore>
f 0 360
i "TempoChange" 10 80
i "LinkEnable" 20 1 0
i "LinkEnable" 30 1 1
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

Voir la section *Ableton Link Opcodes* pour plus d'information.

## Crédits

Par : Michael Gogins 2017



# link\_enable

link\_enable — Active ou désactive la synchronisation avec la session Ableton Link.

## Description

Opcodes du greffon `ableton_link_opcodes`.

Active ou désactive la synchronisation avec le tempo et la pulsation de la session globale du réseau Ableton Link.

## Syntaxe

```
ableton_link_enable i_peer [, k_enable]
```

## Initialisation

*i\_peer* -- L'objet pair Ableton Link.

## Exécution

*k\_enable* -- Active (valeur non nulle) ou désactive (valeur nulle) la synchronisation de ce pair avec la session Ableton Link.

## Exemples

Voici un exemple de l'opcode `link_enable`. Il utilise le fichier `ableton_link.csd` [examples/ableton\_link.csd].

### Exemple 479. Exemple de l'opcode `link_enable`.

```
<CsoundSynthesizer>
<CsLicense>
Run the Ableton Link "LinkHut" example application, or some other
Ableton Link peer, while you run this CSD to see what happens.
</CsLicense>
<CsOptions>
-m0 -d -odac
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 10
nchnls = 2

alwayson "LinkMonitor"

gi_peer link_create 72
gk_beat init 0

instr Beep
asignal oscils 20000, p4, 0
outs asignal, asignal
endin
```

```

instr TempoChange
link_tempo_set gi_peer, 80
endin

instr LinkEnable
i_enabled = p4
link_enable gi_peer, i_enabled
endin

instr LinkMonitor
i_kperiod_seconds = kamps / sr
printf_i "kperiod: %9.6f seconds.\n", 1, i_kperiod_seconds
printf_i "gi_peer: %g\n", 1, gi_peer
link_enable gi_peer, 1
k_trigger, gk_beat, k_phase, k_time link_metro gi_peer, 1
k_peers link_peers gi_peer
k_tempo link_tempo_get gi_peer
k_enabled link_is_enabled gi_peer
k_hz = 1000
if floor(gk_beat % 4) == 0 then
k_hz = 3000
else
k_hz = 2000
endif
schedkwhen k_trigger, 0, 1, "Beep", 0, 0.01, k_hz
printf "LinkMonitor: gi_peer: %g k_enabled: %9.4f k_trigger: %9.4f beat: %9.4f k_phase: %9.4f time: %9.4f\n", 1, k_enabled, k_trigger, gk_beat, k_phase, k_time
endin
</CsInstruments>
<CsScore>
f 0 360
i "TempoChange" 10 80
i "LinkEnable" 20 1 0
i "LinkEnable" 30 1 1
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

Voir la section *Ableton Link Opcodes* pour plus d'information.

## Crédits

Par : Michael Gogins 2017

# link\_is\_enabled

`link_is_enabled` — Indique si ce pair est synchronisé avec la session du réseau global Ableton Link.

## Description

Opcodes du greffon `ableton_link_opcodes`.

Indique si la pulsation et le temps de ce pair sont synchronisés avec la session du réseau global Ableton Link.

## Syntaxe

```
k_is_enabled link_is_enabled i_peer
```

## Initialisation

`i_peer` -- L'objet pair Ableton Link.

## Exécution

`k_is_enabled` -- Retourne 1 si ce pair est activé, 0 sinon.

## Exemples

Voici un exemple de l'opcode `link_is_enabled`. Il utilise le fichier `ableton_link.csd` [examples/ableton\_link.csd].

### Exemple 480. Exemple de l'opcode `link_is_enabled`.

```
<CsoundSynthesizer>
<CsLicense>
Run the Ableton Link "LinkHut" example application, or some other
Ableton Link peer, while you run this CSD to see what happens.
</CsLicense>
<CsOptions>
-m0 -d -odac
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 10
nchnls = 2

alwayson "LinkMonitor"

gi_peer link_create 72
gk_beat init 0

instr Beep
asignal oscils 20000, p4, 0
outs asignal, asignal
endin

instr TempoChange
```

```

link_tempo_set gi_peer, 80
endin

instr LinkEnable
i_enabled = p4
link_enable gi_peer, i_enabled
endin

instr LinkMonitor
i_kperiod_seconds = ksmps / sr
printf_i "kperiod: %9.6f seconds.\n", 1, i_kperiod_seconds
printf_i "gi_peer: %g\n", 1, gi_peer
link_enable gi_peer, 1
k_trigger, gk_beat, k_phase, k_time link_metro gi_peer, 1
k_peers link_peers gi_peer
k_tempo link_tempo_get gi_peer
k_enabled link_is_enabled gi_peer
k_hz = 1000
if floor(gk_beat % 4) == 0 then
k_hz = 3000
else
k_hz = 2000
endif
schedkwhen k_trigger, 0, 1, "Beep", 0, 0.01, k_hz
printf "LinkMonitor: gi_peer: %g k_enabled: %9.4f k_trigger: %9.4f beat: %9.4f k_phase: %9.4f time: %9.4f\n", 1, k_enabled, k_trigger, gk_beat, k_phase, k_time
endin
</CsInstruments>
<CsScore>
f 0 360
i "TempoChange" 10 80
i "LinkEnable" 20 1 0
i "LinkEnable" 30 1 1
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

Voir la section *Ableton Link Opcodes* pour plus d'information.

## Crédits

Par : Michael Gogins 2017

# link\_metro

`link_metro` — Retourne un déclencheur valant 1 sur la pulsation et 0 sinon, ainsi que la pulsation, la phase et le temps d'Ableton Link pour cette session.

## Description

Opcodes du greffon `ableton_link_opcodes`.

Retourne un déclencheur valant 1 sur la pulsation et 0 sinon, ainsi que la pulsation, la phase et le temps d'Ableton Link pour cette session pour une quantification donnée.

## Syntaxe

```
k_trigger, k_beat, k_phase, k_current_time_seconds link_metro i_peer [, k_quantum]
```

## Initialisation

*i\_peer* -- L'objet pair Ableton Link.

## Exécution

*k\_trigger* -- Déclencheur, 1 sur la première pulsation dans chaque quantification locale de la pulsation et 0 sinon.

*k\_beat* -- Le nombre de pulsations compté depuis que ce pair est activé.

*k\_phase* -- La phase de cette pulsation en fonction de la quantification.

*k\_current\_time\_seconds* -- Le temps de la pulsation compté depuis que ce pair est activé.

*k\_quantum* -- La quantification locale de la pulsation, qui peut être un multiple ou une fraction de la pulsation. La valeur par défaut est 1.

Cet opcode permet l'utilisation d'une session Ableton Link pour déclencher des événements dans Csound. On peut aussi l'utiliser pour obtenir la pulsation et le temps de la session ainsi que la phase en fonction de la quantification locale de la pulsation. Par exemple, une quantification de 4 peut impliquer une mesure à 4/4, tandis qu'une quantification de 0,25 divisera la pulsation par 4.

## Exemples

Voici un exemple de l'opcode `link_metro`. Il utilise le fichier *ableton\_link.csd* [examples/ableton\_link.csd].

### Exemple 481. Exemple de l'opcode `link_metro`.

```
<CsoundSynthesizer>
<CsLicense>
Run the Ableton Link "LinkHut" example application, or some other
Ableton Link peer, while you run this CSD to see what happens.
</CsLicense>
<CsOptions>
```

```

-m0 -d -odac
</CsOptions>
<CsInstruments>
  sr = 44100
  ksmpps = 10
  nchnls = 2

  alwayson "LinkMonitor"

  gi_peer link_create 72
  gk_beat init 0

  instr Beep
  asignal oscils 20000, p4, 0
  outs asignal, asignal
  endin

  instr TempoChange
  link_tempo_set gi_peer, 80
  endin

  instr LinkEnable
  i_enabled = p4
  link_enable gi_peer, i_enabled
  endin

  instr LinkMonitor
  i_kperiod_seconds = ksmpps / sr
  printf_i "kperiod: %9.6f seconds.\n", 1, i_kperiod_seconds
  printf_i "gi_peer: %g\n", 1, gi_peer
  link_enable gi_peer, 1
  k_trigger, gk_beat, k_phase, k_time link_metro gi_peer, 1
  k_peers link_peers gi_peer
  k_tempo link_tempo_get gi_peer
  k_enabled link_is_enabled gi_peer
  k_hz = 1000
  if floor(gk_beat % 4) == 0 then
    k_hz = 3000
  else
    k_hz = 2000
  endif
  schedkwhen k_trigger, 0, 1, "Beep", 0, 0.01, k_hz
  printf "LinkMonitor: gi_peer: %g k_enabled: %9.4f k_trigger: %9.4f beat: %9.4f k_phase: %9.4f time: %9.4f\n", 1, gi_peer, k_enabled, k_trigger, gk_beat, k_phase, k_time
  endin
</CsInstruments>
<CsScore>
  f 0 360
  i "TempoChange" 10 80
  i "LinkEnable" 20 1 0
  i "LinkEnable" 30 1 1
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

Voir la section *Ableton Link Opcodes* pour plus d'information.

## Crédits

Par : Michael Gogins 2017

# link\_peers

link\_peers — Retourne le nombre de pairs dans la session.

## Description

Opcodes du greffon `ableton_link_opcodes`.

Retourne le nombre de pairs dans la session du réseau global Ableton Link.

## Syntaxe

```
k_count link_peers i_peer
```

## Initialisation

*i\_peer* -- L'objet pair Ableton Link.

## Exécution

*k\_count* -- Le nombre courant de pairs dans la session du réseau global Ableton Link.

## Exemples

Voici un exemple de l'opcode `link_peers`. Il utilise le fichier *ableton\_link.csd* [examples/ableton\_link.csd].

### Exemple 482. Exemple de l'opcode link\_peers.

```
<CsoundSynthesizer>
<CsLicense>
Run the Ableton Link "LinkHut" example application, or some other
Ableton Link peer, while you run this CSD to see what happens.
</CsLicense>
<CsOptions>
-m0 -d -odac
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 10
nchnls = 2

alwayson "LinkMonitor"

gi_peer link_create 72
gk_beat init 0

instr Beep
asignal oscils 20000, p4, 0
outs asignal, asignal
endin

instr TempoChange
link_tempo_set gi_peer, 80
endin
```

```

instr LinkEnable
i_enabled = p4
link_enable gi_peer, i_enabled
endin

instr LinkMonitor
i_kperiod_seconds = ksmps / sr
printf_i "kperiod: %9.6f seconds.\n", 1, i_kperiod_seconds
printf_i "gi_peer: %g\n", 1, gi_peer
link_enable gi_peer, 1
k_trigger, gk_beat, k_phase, k_time link_metro gi_peer, 1
k_peers link_peers gi_peer
k_tempo link_tempo_get gi_peer
k_enabled link_is_enabled gi_peer
k_hz = 1000
if floor(gk_beat % 4) == 0 then
k_hz = 3000
else
k_hz = 2000
endif
schedkwhen k_trigger, 0, 1, "Beep", 0, 0.01, k_hz
printf "LinkMonitor: gi_peer: %g k_enabled: %9.4f k_trigger: %9.4f beat: %9.4f k_phase: %9.4f time: %9.4f\n", 1, gi_peer, k_enabled, k_trigger, gk_beat, k_phase, k_time
endin
</CsInstruments>
<CsScore>
f 0 360
i "TempoChange" 10 80
i "LinkEnable" 20 1 0
i "LinkEnable" 30 1 1
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

Voir la section *Ableton Link Opcodes* pour plus d'information.

## Crédits

Par : Michael Gogins 2017



# link\_tempo\_get

link\_tempo\_get — Retourne le tempo courant de la session du réseau global Ableton Link.

## Description

Opcodes du greffon `ableton_link_opcodes`.

Retourne le tempo courant de la session du réseau global Ableton Link.

## Syntaxe

```
k_bpm link_tempo_get i_peer
```

## Initialisation

*i\_peer* -- L'objet pair Ableton Link.

## Exécution

*k\_bpm* -- Le tempo courant de la session du réseau global Ableton Link.

## Exemples

Voici un exemple de l'opcode `link_tempo_get`. Il utilise le fichier `ableton_link.csd` [examples/ableton\_link.csd].

### Exemple 483. Exemple de l'opcode link\_tempo\_get.

```
<CsoundSynthesizer>
<CsLicense>
Run the Ableton Link "LinkHut" example application, or some other
Ableton Link peer, while you run this CSD to see what happens.
</CsLicense>
<CsOptions>
-m0 -d -odac
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 10
nchnls = 2

alwayson "LinkMonitor"

gi_peer link_create 72
gk_beat init 0

instr Beep
asignal oscils 20000, p4, 0
outs asignal, asignal
endin

instr TempoChange
link_tempo_set gi_peer, 80
```

```
    endin

    instr LinkEnable
    i_enabled = p4
    link_enable gi_peer, i_enabled
    endin

    instr LinkMonitor
    i_kperiod_seconds = ksmps / sr
    printf_i "kperiod: %9.6f seconds.\n", 1, i_kperiod_seconds
    printf_i "gi_peer: %g\n", 1, gi_peer
    link_enable gi_peer, 1
    k_trigger, gk_beat, k_phase, k_time link_metro gi_peer, 1
    k_peers link_peers gi_peer
    k_tempo link_tempo_get gi_peer
    k_enabled link_is_enabled gi_peer
    k_hz = 1000
    if floor(gk_beat % 4) == 0 then
    k_hz = 3000
    else
    k_hz = 2000
    endif
    schedkwhen k_trigger, 0, 1, "Beep", 0, 0.01, k_hz
    printf "LinkMonitor: gi_peer: %g k_enabled: %9.4f k_trigger: %9.4f beat: %9.4f k_phase: %9.4f time: %9.4f\n", 1, k_enabled, k_trigger, gk_beat, k_phase, k_time
    endin
</CsInstruments>
<CsScore>
f 0 360
i "TempoChange" 10 80
i "LinkEnable" 20 1 0
i "LinkEnable" 30 1 1
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

Voir la section *Ableton Link Opcodes* pour plus d'information.

## Crédits

Par : Michael Gogins 2017

# link\_tempo\_set

link\_tempo\_set — Fixe le tempo.

## Description

Opcodes du greffon `ableton_link_opcodes`.

Fixe le tempo local si ce pair n'est pas activé ; fixe le tempo de la session du réseau global Ableton Link si ce pair est activé.

## Syntaxe

```
link_tempo_set i_peer, k_bpm [, k_at_time_seconds]
```

## Initialisation

*i\_peer* -- L'objet pair Ableton Link.

*k\_bpm* -- Le tempo requis en pulsations par minute.

*k\_at\_time\_seconds* -- Le temps de la session auquel le tempo indiqué prendra effet. La valeur par défaut est le temps courant de la session.

## Exemples

Voici un exemple de l'opcode `link_tempo_set`. Il utilise le fichier *ableton\_link.csd* [examples/ableton\_link.csd].

### Exemple 484. Exemple de l'opcode link\_tempo\_set.

```
<CsoundSynthesizer>
<CsLicense>
Run the Ableton Link "LinkHut" example application, or some other
Ableton Link peer, while you run this CSD to see what happens.
</CsLicense>
<CsOptions>
-m0 -d -odac
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 10
nchnls = 2

alwayson "LinkMonitor"

gi_peer link_create 72
gk_beat init 0

instr Beep
asignal oscils 20000, p4, 0
outs asignal, asignal
endin
```

```
instr TempoChange
link_tempo_set gi_peer, 80
endin

instr LinkEnable
i_enabled = p4
link_enable gi_peer, i_enabled
endin

instr LinkMonitor
i_kperiod_seconds = ksmpr / sr
printf_i "kperiod: %9.6f seconds.\n", 1, i_kperiod_seconds
printf_i "gi_peer: %g\n", 1, gi_peer
link_enable gi_peer, 1
k_trigger, gk_beat, k_phase, k_time link_metro gi_peer, 1
k_peers link_peers gi_peer
k_tempo link_tempo_get gi_peer
k_enabled link_is_enabled gi_peer
k_hz = 1000
if floor(gk_beat % 4) == 0 then
k_hz = 3000
else
k_hz = 2000
endif
schedkwhen k_trigger, 0, 1, "Beep", 0, 0.01, k_hz
printf "LinkMonitor: gi_peer: %g k_enabled: %9.4f k_trigger: %9.4f beat: %9.4f k_phase: %9.4f time: %9.4f\n", 1, k_enabled, k_trigger, gk_beat, k_phase, k_time
endin
</CsInstruments>
<CsScore>
f 0 360
i "TempoChange" 10 80
i "LinkEnable" 20 1 0
i "LinkEnable" 30 1 1
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

Voir la section *Ableton Link Opcodes* pour plus d'information.

## Crédits

Par : Michael Gogins 2017

# linrand

linrand — Générateur de nombres aléatoires de distribution linéaire (valeurs positives seulement).

## Description

Générateur de nombres aléatoires de distribution linéaire (valeurs positives seulement). C'est un générateur de bruit de classe x.

## Syntaxe

```
ares linrand krange
```

```
ires linrand krange
```

```
kres linrand krange
```

## Exécution

*krange* -- l'intervalle des nombres aléatoires (0 - *krange*). Ne produit que des nombres positifs.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode linrand. Il utilise le fichier *linrand.csd* [examples/linrand.csd].

### Exemple 485. Exemple de l'opcode linrand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o linrand.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1    ; every run time same values

klin linrand 100
```

```

    printk .2, klin    ; look
    aout oscili 0.8, 440+klin, 1  ; & listen
    outs aout, aout
    endin

    instr 2  ; every run time different values

    seed 0
    klin linrand 100
    printk .2, klin    ; look
    aout oscili 0.8, 440+klin, 1  ; & listen
    outs aout, aout
    endin

</CsInstruments>
<CsScore>
; sine wave
f 1 0 16384 10 1

i 1 0 2
i 2 3 2
e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

i 1 1 time      0.00033:    13.54770
i 1 1 time      0.20033:    32.38746
i 1 1 time      0.40033:    47.69304
i 1 1 time      0.60033:    19.82218
i 1 1 time      0.80033:    42.98293
i 1 1 time      1.00000:    81.13174
i 1 1 time      1.20033:    47.39585
i 1 1 time      1.40033:    12.53248
i 1 1 time      1.60033:    35.70722
i 1 1 time      1.80000:    65.25774
i 1 1 time      2.00000:    23.24811
Seeding from current time 392575384
i 2 2 time      3.00033:    23.05609
i 2 2 time      3.20033:    76.15114
i 2 2 time      3.40033:    22.78861
i 2 2 time      3.60000:     0.79064
i 2 2 time      3.80033:    43.49438
i 2 2 time      4.00000:    34.10963
i 2 2 time      4.20000:    31.88702
i 2 2 time      4.40033:    59.78054
i 2 2 time      4.60033:     4.96821
i 2 2 time      4.80033:    24.69674
i 2 2 time      5.00000:    21.88815

```

## Voir aussi

*seed, betarand, bexprnd, cauchy, exprand, gauss, pcauchy, poisson, trirand, unirand, weibull*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

# linseg

linseg — Trace une suite de segments de droite entre les points spécifiés.

## Description

Trace une suite de segments de droite entre les points spécifiés.

## Syntaxe

```
ares linseg ia, idur1, ib [, idur2] [, ic] [...]  
kres linseg ia, idur1, ib [, idur2] [, ic] [...]
```

## Initialisation

*ia* -- valeur initiale.

*ib*, *ic*, etc. -- valeur après *dur1* secondes, etc.

*idur1* -- durée en secondes du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

*idur2*, *idur3*, etc. -- durée en secondes des segments suivants. Une valeur nulle ou négative terminera la phase d'initialisation avec le point précédent, permettant au dernier segment défini de continuer durant toute l'exécution. La valeur par défaut est zéro.

## Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par 2 ou plus points spécifiés. La somme des valeurs *dur* peut égaier ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, la dernière valeur sera répétée jusqu'à la fin de la note.

## Exemples

Voici un exemple de l'opcode linseg. Il utilise le fichier *linseg.csd* [examples/linseg.csd].

### Exemple 486. Exemple de l'opcode linseg.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac      ;;realtime audio out  
;-iadc     ;;uncomment -iadc if realtime audio input is needed too  
; For Non-realtime ouput leave only the line below:  
;-o linseg.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>
```

```

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 2^10, 10, 1

instr 1

kcps = cspch(p4)
kenv linseg 0, 0.25, 1, 0.75, 0 ; together = 1 sec
asig poscil kenv, kcps, giSine
outs asig, asig

endin

instr 2 ; scaling to duration

kcps = cspch(p4)
kenv linseg 0, p3*0.25, 1, p3*0.75, 0
asig poscil kenv, kcps, giSine
outs asig, asig

endin

instr 3 ; with negative value

kcps = cspch(p4)
aenv linseg 0, 0.1, 1, 0.5, -0.9, 0.4, 0
asig poscil aenv, kcps, giSine
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 1 7.00 ; = 1 sec, p3 fits exactly
i 1 2 2 7.00 ; = 2 sec, p3 truncated at 1 sec

i 2 4 1 7.00 ; scales to duration
i 2 6 2 7.00 ; of p3

i 3 9 2 7.00
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*expon, expseg, expsegb, expsegr, cosseg, line, linsegr transeg transegb*

## Crédits

Auteur : Barry L. Vercoe



# linsegb

linsegb — Trace une suite de segments de droite entre les points absolus spécifiés.

## Description

Trace une suite de segments de droite entre les points absolus spécifiés.

## Syntaxe

```
ares linsegb ia, itim1, ib [, itim2] [, ic] [...]  
kres linsegb ia, itim1, ib [, itim2] [, ic] [...]
```

## Initialisation

*ia* -- valeur initiale.

*ib*, *ic*, etc. -- valeur à *tim1* secondes, etc.

*itim1* -- date en secondes de la fin du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

*itim2*, *itim3*, etc. -- date en secondes de la fin des segments suivants.

## Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par 2 ou plus points spécifiés. La dernière valeur *tim* peut égal ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, la dernière valeur sera répétée jusqu'à la fin de la note.

## Exemples

Voici un exemple de l'opcode linsegb. Il utilise le fichier *linsegb.csd* [exemples/linsegb.csd].

### Exemple 487. Exemple de l'opcode linsegb.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac      ;;realtime audio out  
;-iadc      ;;uncomment -iadc if realtime audio input is needed too  
; For Non-realtime ouput leave only the line below:  
;-o linseg.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 32
```

```

nchnls = 2
odbfs = 1

giSine ftgen 0, 0, 2^10, 10, 1

instr 1

kcps = cpspch(p4)
kenv linsegb 0, 0.25, 1, 1, 0
asig poscil kenv, kcps, giSine
outs asig, asig

endin

instr 2 ; scaling to duration

kcps = cpspch(p4)
kenv linseg 0, p3*0.25, 1, p3, 0
asig poscil kenv, kcps, giSine
outs asig, asig

endin

</CsInstruments>
<CsScore>

i 1 0 1 7.00 ; = 1 sec, p3 fits exactly
i 1 2 2 7.00 ; = 2 sec, p3 truncated at 1 sec

i 2 4 1 7.00 ; scales to duration
i 2 6 2 7.00 ; of p3

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*expon, expseg, expsegr, line, linseg linsegr transeg*

## Crédits

Auteur : Victor Lazzarini  
Juin 2011

Nouveau dans la version 5.14

# linsegr

linsegr — Trace une suite de segments de droite entre les points spécifiés avec un segment de relâchement.

## Description

Trace une suite de segments de droite entre les points spécifiés avec un segment de relâchement.

## Syntaxe

```
ares linsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz  
kres linsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
```

## Initialisation

*ia* -- valeur initiale.

*ib*, *ic*, etc. -- valeur après *dur1* secondes, etc.

*idur1* -- durée en secondes du premier segment. Avec une valeur nulle ou négative l'initialisation sera ignorée.

*idur2*, *idur3*, etc. -- durée en secondes des segments suivants. Une valeur nulle ou négative terminera la phase d'initialisation avec le point précédent, permettant au dernier segment défini de continuer durant toute l'exécution. La valeur par défaut est zéro.

*irel*, *iz* -- durée en secondes et valeur finale du segment de relâchement de la note.

Pour les versions de Csound antérieures à la 5.00, le temps de relâchement ne peut pas dépasser 32767/*kr* secondes. Cette limite a été étendue à  $(2^{31}-1)/kr$ .

## Exécution

Ces unités génèrent des signaux de contrôle ou audio dont les valeurs passent par 2 ou plus points spécifiés. La somme des valeurs *dur* peut évaluer ou non la durée d'exécution de l'instrument : avec une exécution plus courte, la courbe sera tronquée alors qu'avec une exécution plus longue, le dernier segment défini continuera dans la même direction.

*linsegr* fait partie des unités « r » de Csound qui contiennent un détecteur de fin de note et une extension de durée pour le relâchement. Quand la fin d'un événement ou MIDI noteoff est détectée, la durée d'exécution de l'instrument courant est immédiatement allongée de *irel* secondes, de façon à ce que la valeur *iz* soit atteinte à la fin de cette période (quelque soit le segment dans lequel se trouvait l'unité). Les unités « r » peuvent aussi être modifiées par les vitesses nulles provoquant un message MIDI noteoff. S'il y a plusieurs extensions de durée dans un instrument, c'est la plus longue qui sera choisie.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note off, comme *linenr* et *expsegr*, ou bien l'on peut construire des enveloppes plus complexes au moyen de *xtratim* et de *release*. Noter que qu'il n'est pas nécessaire d'utiliser *xtratim* avec *linsegr*, car la durée est allongée automatiquement.

## Exemples

Voici un exemple de l'opcode *linsegr*. Il utilise le fichier *linsegr.csd* [examples/linsegr.csd].

### Exemple 488. Exemple de l'opcode `linsegr`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0   ;;realtime audio out and realtime midi in
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o linsegr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

icps cpsmidi
iamp ampmidi .3

kenv linsegr 1, .05, 0.5, 1, 0
asig pluck kenv, icps, 200, 1, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 4096 10 1 ;sine wave

f0 30 ;runs 30 seconds
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*linenr, expsegr, envlpxr, mxadsr, madsr expon, expseg, expsega line, linseg, xtratim, transegr*

## Crédits

Auteur : Barry L. Vercoe

Décembre 2002, Décembre 2006. Merci à Istvan Varga pour l'ajout de la documentation sur le temps de relâchement maximum.

Nouveau dans Csound 3.47

# liveconv

liveconv — Convolution partitionnée avec réponse impulsionnelle rechargeable dynamiquement.

## Description

Opcode du greffon liveconv.

Convolution partitionnée, calculée efficacement, utilisant une table de fonction comme source de réponse impulsionnelle (RI), semblable à l'opcode *ftconv*. L'opcode *liveconv* permet de recharger dynamiquement les données de RI à n'importe quel moment durant la convolution a lieu, contrôlé par le paramètre *kupdate*. Grâce à la façon dont la RI est mise à jour, l'opération se fait sans artéfact audio dans la sortie de la convolution.

La table de RI est lue partition par partition après le signalement d'une mise à jour. Les premiers *ksmps* échantillons doivent être disponibles quand l'indicateur de mise à jour est positionné. Le reste de la table peut être rempli en continu lors des passages suivants. Il peut y avoir une nouvelle mise à jour pour chaque partition, permettant jusqu'à *flen/iplen* mises à jour simultanées (*flen* est la longueur de la table de RI).

Le comportement dynamique à faible latence de *liveconv* le rend idéal pour la convolution avec des réponses impulsionnelles échantillonnées en direct, et/ou les transformations et les modifications en temps réel de la réponse impulsionnelle.

## Syntaxe

```
ares liveconv ain, ift, iplen, kupdate, kclear
```

## Initialisation

*ift* -- numéro de la table de stockage de la réponse impulsionnelle (RI) pour la convolution. La table peut être remplie avec de nouvelles données à n'importe quel moment durant la convolution.

*iplen* -- longueur en échantillons de la partition de la réponse impulsionnelle ; doit être une puissance entière de deux. Les faibles valeurs donnent un délai plus court en sortie au prix d'une utilisation plus intensive du CPU.

## Exécution

*ain* -- signal d'entrée.

*ares* -- signal de sortie.

*kupdate* -- indicateur de mise à jour de la table de RI. Si *kupdate*=1 la table de RI *ift* est chargée partition par partition, en commençant par la prochaine partition. Si *kupdate*=-1 la table de RI *ift* est déchargée (mise à zéro) partition par partition, en commençant par la prochaine partition. Les autres valeurs n'ont aucun effet.

*kclear* -- indicateur pour effacer tous les tampons internes. Si *kclear* prend n'importe quelle valeur différente de zéro, les tampons internes sont effacés immédiatement. L'opération n'est pas sans artéfacts.

## Exemple

Voici un exemple de l'opcode liveconv. Il utilise le fichier *liveconv.csd* [examples/liveconv.csd].

### Exemple 489. Exemple de l'opcode liveconv.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac ;realtime audio out
</CsOptions>
<CsInstruments>

  sr = 44100
  nchnls = 2
  odbfs = 1

  ; empty IR table
  giIR_record ftgen 0, 0, 131072, 2, 0

  ; Record impulse response
  instr 13

    p3 = ftlen(giIR_record)/sr
    iskip = p4
    irlen = p5
    a1 diskin2 "fox.wav", 1, iskip

    ; Fill IR table with segment from audio file
    amp linseg 0, 0.1, 1, irlen, 1, 0.1, 0, 1, 0
    andx_IR line 0, 1, 1/(ftlen(giIR_record)/sr)
    tablew a1*amp, andx_IR, giIR_record, 1
    outch 1, a1*amp ; output the IR
    ktrig init 1
    if ktrig > -1 then
      chnset ktrig, "conv_update"
      ktrig -= 1
    endif

  endin

  ; The convolver
  instr 14

    ain diskin2 "beats.wav", 1, 0, 1
    kupdate chnget "conv_update"
    aconv liveconv ain, giIR_record, 2048, kupdate, 0
    outch 2, aconv*0.009 ; output the convolution response
  endin

</CsInstruments>
<CsScore>
; record impulse response
;      skip IR_dur
i13 0 1 0.0 0.5
i13 2 1 0.5 0.5
i13 4 1 1.0 0.5
i13 6 1 1.5 0.5
i13 8 1 2.0 0.75
i13 10 1 2.38 0.25

; convolve
i14 0.0 11.65

e

</CsScore>
```

`</CsoundSynthesizer>`

## Voir aussi

*ftconv tvconv pconvolve, convolve,*

## Crédits

Auteur : Sigurd Saue, Oeyvind Brandtsegg  
2017

# locsend

locsend — Distribue les signaux audio d'un opcode *locsig* précédent.

## Description

*locsend* dépend de l'existence d'un *locsig* précédemment défini. Le nombre de signaux de sortie doit correspondre à celui du *locsig* précédent. Les signaux de sortie de *locsend* sont dérivés des valeurs de distance et de réverbération données dans le *locsig* et sont prêts à être envoyés à des unités de réverbération locale ou globale (voir l'exemple ci-dessous). Le taux de réverbération et la balance entre les 2 ou les 4 canaux sont calculés selon la description dans le livre de Dodge (un texte essentiel !).

## Syntaxe

```
a1, a2 locsend
```

```
a1, a2, a3, a4 locsend
```

## Exemples

```
asig ;some audio signal
kdegree      line    0, p3, 360
kdistance     line    1, p3, 10
a1, a2, a3, a4 locsig asig, kdegree, kdistance, .1
ar1, ar2, ar3, ar4 locsend
ga1 = ga1+ar1
ga2 = ga2+ar2
ga3 = ga3+ar3
ga4 = ga4+ar4
                                outq    a1, a2, a3, a4
endin

instr 99 ; reverb instrument
a1      reverb2 ga1, 2.5, .5
a2      reverb2 ga2, 2.5, .5
a3      reverb2 ga3, 2.5, .5
a4      reverb2 ga4, 2.5, .5
                                outq    a1, a2, a3, a4

ga1=0
ga2=0
ga3=0
ga4=0
```

Dans l'exemple ci-dessus, le signal *asig* fait un tour complet sur un cercle pendant la durée de la note tout en « s'éloignant » de la position de l'auditeur. *locsig* envoie en interne à *locsend* la quantité de signal appropriée. Les sorties de *locsend* sont ajoutées à des accumulateurs globaux selon la manière habituelle dans Csound et les signaux globaux servent d'entrée aux unités de réverbération dans un instrument séparé. Pour un exemple, voir *locsig*.

*locsig* est utile pour les panoramiques quadro et stéréo ainsi que pour le placement fixe des sons n'importe où entre deux haut-parleurs. Ci-dessous un exemple de placement fixe de sons dans un champ stéréo. Il utilise le fichier *locsend\_stereo.csd* [examples/locsend\_stereo.csd].

### Exemple 490. Exemple de l'opcode locsend.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.



```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o locsend_stereo.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

ga1 init 0
ga2 init 0

instr 1

krevsend = p4
aout diskin2 "beats.wav", 1, 0, 1
kdegree line 0, p3, 180 ;left to right
kdistance line 1, p3, 30
a1, a2 locsig aout, kdegree, kdistance, p4
ar1, ar2 locsend
ga1 = ga1+ar1
ga2 = ga2+ar2
      outs a1, a2

endin

instr 99 ; reverb instrument
a1 reverb2 ga1, 2.5, .5
a2 reverb2 ga2, 2.5, .5
      outs a1, a2
ga1 = 0
ga2 = 0

endin
</CsInstruments>
<CsScore>
; sine wave.
f 1 0 16384 10 1

i 1 0 4 .1 ;with reverb
i 1 + 4 0 ;no reverb
i99 0 7
e
</CsScore>
</CsoundSynthesizer>

```

Quelques notes:

```

; place le son dans le haut-parleur gauche et au premier plan :
i1 0 1 0 1

; place le son dans le haut-parleur droit et à l'arrière plan :
i1 1 1 90 25

; place le son au milieu gauche-droite et à mi-distance en profondeur :
i1 2 1 45 12
e

```

L'exemple suivant montre une utilisation intuitive simple de la valeur de distance pour simuler un effet Doppler. La même valeur est utilisée comme diviseur pour la fréquence et comme paramètre de distance pour *locsig*.

```
kdistance      line    1, p3, 10
kfreq = (ifreq * 340) / (340 + kdistance)
asig           oscili  iamp, kfreq, 1
kdegree        line    0, p3, 360
a1, a2, a3, a4  locsig  asig, kdegree, kdistance, .1
ar1, ar2, ar3, ar4 locsend
```

## Voir aussi

*locsig*

## Crédits

Auteur : Richard Karpen  
Seattle, WA USA  
1998

Nouveau dans la version 3.48 de Csound

# locsig

locsig — Distribue le signal d'entrée entre 2 ou 4 canaux.

## Description

*locsig* distribue le signal d'entrée entre 2 ou 4 canaux en utilisant des valeurs en degrés pour calculer la balance entre les canaux adjacents. Il y a aussi des arguments pour la distance (pour atténuer les signaux qui doivent être perçus comme s'ils étaient plus éloignés que les haut-parleurs) et pour la quantité de signal qui sera envoyée aux unités de réverbération. Basé sur l'exemple du livre de Charles Dodge/Thomas Jerse, *Computer Music*, page 320.

## Syntaxe

```
a1, a2 locsig asig, kdegree, kdistance, kreverbsend
```

```
a1, a2, a3, a4 locsig asig, kdegree, kdistance, kreverbsend
```

## Exécution

*kdegree* -- valeur entre 0 et 360 pour le placement du signal dans un espace à 2 ou 4 canaux configuré comme ceci : a1=0, a2=90, a3=180, a4=270 (kdegree=45 répartira également le signal entre a1 et a2). *locsig* applique *kdegree* à des fonctions sin et cos pour obtenir les balances du signal (par exemple asig=1, kdegree=45, a1=a2=0.707).

*kdistance* -- valeur >= 1 pour atténuer le signal et calculer le niveau de réverbération pour simuler l'éloignement. Plus *kdistance* est important et plus le son sera adouci et quelque peu réverbéré (assumant l'utilisation de *locsend* dans ce cas).

*kreverbsend* -- le pourcentage de signal direct qui sera combiné avec les valeurs de distance et de degrés pour obtenir la quantité de signal qui sera envoyée à une unité de réverbération telle que *reverb* ou *reverb2*.

## Exemples

Voici un exemple de l'opcode *locsig*. Il utilise le fichier *locsig\_quad.csd* [examples/locsig\_quad.csd].

### Exemple 491. Exemple de l'opcode *locsig*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o locsig_quad.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
```

```

nchnls = 4
0dbfs = 1

ga1 init 0
ga2 init 0
ga3 init 0
ga4 init 0

instr 1

krevsend = p4
aout diskint "beats.wav", 1, 0, 1
kdegree line 0, p3, 360 ;full circle
kdistance line 1, p3, 1
a1, a2, a3, a4 locsig aout, kdegree, kdistance, krevsend
ar1, ar2, ar3, ar4 locsend

ga1 = ga1+ar1
ga2 = ga2+ar2
ga3 = ga3+ar3
ga4 = ga4+ar4
outq a1, a2, a3, a4

endin

instr 99 ; reverb instrument
a1 reverb2 ga1, 3.5, .5
a2 reverb2 ga2, 3.5, .5
a3 reverb2 ga3, 3.5, .5
a4 reverb2 ga4, 3.5, .5
outq a1, a2, a3, a4

ga1 = 0
ga2 = 0
ga3 = 0
ga4 = 0

endin
</CsInstruments>
<CsScore>
; sine wave.
f 1 0 16384 10 1

i 1 0 14 .1 ;with reverb
i 1 14 14 0 ;no reverb
i99 0 36
e
</CsScore>
</CsoundSynthesizer>

```

Dans l'exemple ci-dessus, le signal *aout* fait un tour complet sur un cercle pendant la durée de la note tout en « s'éloignant » de la position de l'auditeur. *locsigs* envoie en interne à *locsend* la quantité de signal appropriée. Les sorties de *locsend* sont ajoutées à des accumulateurs globaux selon la manière habituelle dans Csound et les signaux globaux servent d'entrée aux unités de réverbération dans un instrument séparé.

*locsigs* est utile pour les panoramiques quadro et stéréo ainsi que pour le placement fixe des sons n'importe où entre deux haut-parleurs. Ci-dessous un exemple de placement fixe de sons dans un champ stéréo.

```

instr 1
a1, a2          locsig asig, p4, p5, .1
ar1, ar2        locsend
ga1 = ga1+ar1
ga2 = ga2+ar2
outs a1, a2
endin

```

```
instr 99
  ; reverb...
endin
```

A few notes:

```
; place le son dans le haut-parleur gauche et au premier plan :
i1 0 1 0 1

; place le son dans le haut-parleur droit et à l'arrière plan :
i1 1 1 90 25

; place le son au milieu gauche-droite et à mi-distance en profondeur :
i1 2 1 45 12
e
```

L'exemple suivant montre une utilisation intuitive simple de la valeur de distance pour simuler un effet Doppler. La même valeur est utilisée comme diviseur pour la fréquence et comme paramètre de distance pour *locsig*.

```
kdistance      line    1, p3, 10
kfreq = (ifreq * 340) / (340 + kdistance)
asig           oscili  iamp, kfreq, 1
kdegree        line    0, p3, 360
a1, a2, a3, a4  locsig  asig, kdegree, kdistance, .1
ar1, ar2, ar3, ar4 locsend
```

## Voir aussi

*locsend*

## Crédits

Auteur : Richard Karpen  
Seattle, WA USA  
1998

Nouveau dans la version 3.48 de Csound

# log

`log` — Retourne le logarithme naturel d'un nombre, ou d'un tableau (avec facultativement une base arbitraire).

## Description

Retourne le logarithme naturel de  $x$  ( $x$  strictement positif). Dans le cas d'un tableau en entrée, l'opération peut avoir facultativement une base arbitraire.

Les valeurs de l'argument sont restreintes pour *log*, *log10* et *sqrt*.

## Syntaxe

`log(x)` (pas de restriction de taux)

`log(k/i[])` (k- ou i-tableau)

`kout[]log kin[],ibas`

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression. Dans le cas d'un tableau en entrée, *ibas* est la base arbitraire facultative, qui vaut par défaut *e* (base des logarithmes naturels).

## Exemples

Voici un exemple de l'opcode `log`. Il utilise le fichier *log.csd* [examples/log.csd].

### Exemple 492. Exemple de l'opcode `log`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o log.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = log(8)
  print i1
endin
```

```
</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e
```

```
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme :

```
instr 1: i1 = 2.079
```

## Voir aussi

*abs, exp, frac, int, log10, i, sqrt*

## Crédits

Ecrit par John ffitch.

Nouveau dans la version 3.47

Exemple écrit par Kevin Conder.

# log10

log10 — Retourne un logarithme en base 10.

## Description

Retourne le logarithme en base 10 de  $x$  ( $x$  strictement positif).

Les valeurs de l'argument sont restreintes pour *log*, *log10* et *sqrt*.

## Syntaxe

`log10(x)` (pas de restriction de taux)

`log10(k/i[])` (k- ou i-tableau)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode log10. Il utilise le fichier *log10.csd* [examples/log10.csd].

### Exemple 493. Exemple de l'opcode log10.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o log10.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = log10(8)
  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
```



e

```
</CsScore>  
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  i1 = 0.903
```

## Voir aussi

*abs, exp, frac, int, log, log, i, sqrt*

## Crédits

Ecrit par John ffitich.

Nouveau dans la version 3.47

Exemple écrit par Kevin Conder.

# log2

log2 — Retourne un logarithme en base 2.

## Description

Retourne le logarithme en base 2 de  $x$  ( $x$  strictement positif).

Les valeurs de l'argument sont restreintes pour *log*, *log2* et *sqrt*.

## Syntaxe

`log2(x)` (pas de restriction de taux)

`log2(k/i[])` (k- ou i-tableau)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode log2. Il utilise le fichier *log2.csd* [examples/log2.csd].

### Exemple 494. Exemple de l'opcode log2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o log10.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = log2(8)
  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
```

e

```
</CsScore>  
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1:  i1 = 3.000
```

## Voir aussi

*abs, exp, frac, int, log, log, i, sqrt*

## Crédits

Ecrit John ffitch.

Nouveau dans la version 5.17.12

# logbtwo

logbtwo — Calcule le logarithme en base deux.

## Description

Calcule le logarithme en base deux.

## Syntaxe

`logbtwo(x)` (argument au taux d'initialisation ou de contrôle seulement)

## Exécution

`logbtwo()` retourne le logarithme en base deux de *x*. L'intervalle des valeurs permises en argument va de 0,25 à 4 (c-à-d une réponse comprise entre -2 et +2 octaves). Cette fonction est l'inverse de `powoftwo()`.

Ces fonctions sont rapides, car elles lisent des valeurs stockées dans des tables. Elles sont très utiles lorsque l'on travaille avec des rapports de hauteurs. Elles travaillent au taux-i et au taux-k.

## Exemples

Voici un exemple de l'opcode `logbtwo`. Il utilise le fichier *logbtwo.csd* [examples/logbtwo.csd].

### Exemple 495. Exemple de l'opcode `logbtwo`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o logbtwo.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  i1 = logbtwo(3)
  print i1
endin

</CsInstruments>
<CsScore>
```

```
; Play Instrument #1 for one second.  
i 1 0 1  
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 1.585
```

## Voir aussi

*powoftwo*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Juin 1998

Auteur : John ffitch  
Université de Bath, Codemist, Ltd.  
Bath, UK  
Juillet 1999

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.57 de Csound

# logcurve

logcurve — Cet opcode implémente une formule qui génère une courbe logarithmique normalisée dans l'intervalle 0 - 1. Il est basé sur le travail dans Max / MSP de Eric Singer (c) 1994.

## Description

Génère une courbe logarithmique dans l'intervalle de 0 à 1 avec une raideur de pente arbitraire. Une raideur de pente inférieure ou égale à 1,0 lévera des erreurs NaN (Not-a-Number) et provoquera un comportement instable.

La formule utilisée pour le calcul de la courbe est :

$$\log(x * (y-1)+1) / (\log(y))$$

où x est égal à *kindex* et y est égal à *ksteepness*.

## Syntaxe

kout **logcurve** kindex, ksteepness

## Exécution

*kindex* -- Valeur d'indice. Attendue dans l'intervalle de 0 à 1.

*ksteepness* -- Raideur de la courbe générée. Avec des valeurs proches de 1,0 on obtient une courbe plus rectiligne alors qu'avec des valeurs plus grandes la courbe est plus raide.

*kout* -- Sortie pondérée.

## Exemples

Voici un exemple de l'opcode logcurve. Il utilise le fichier *logcurve.csd* [examples/logcurve.csd].

### Exemple 496. Exemple de l'opcode logcurve.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  Silent
-odac          -iadc     -d      ;;realtime output
</CsOptions>
<CsInstruments>

sr = 48000
ksmps = 100
nchnls = 2

instr 1 ; logcurve test

kmod phasor 1/p3
```

```
kout logcurve kmod, p4

printks "kmod = %f  kout = %f\\n", 0.1, kmod, kout

endin

</CsInstruments>
<CsScore>

i1 0 10 2
i1 10 10 30
i1 20 10 0.5

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*scale, gainslider, expcurve*

## Crédits

Auteur : David Akbari  
Octobre  
2006

# loop\_ge

loop\_ge — Constructions de boucle.

## Description

Construction d'opérations de boucle.

## Syntaxe

```
loop_ge    indx, idecr, imin, label
```

```
loop_ge    kndx, kdecr, kmin, label
```

## Initialisation

*indx* -- variable de taux-i, compteur de la boucle.

*idecr* -- valeur de décrémentation de la boucle.

*imin* -- valeur minimale de l'index de la boucle.

## Exécution

*kndx* -- variable de taux-k, compteur de la boucle.

*kdecr* -- valeur de décrémentation de la boucle.

*kmin* -- valeur minimale de l'index de la boucle.

L'action de **loop\_ge** est équivalente à

```
indx = indx - idecr
if (indx >= imin) igoto label
```

ou à

```
kndx = kndx - kdecr
if (kndx >= kmin) kgoto label
```

## Voir aussi

*loop\_gt*, *loop\_le* et *loop\_lt*.

Plus d'information sur cet opcode : [http://www.csoundjournal.com/2006summer/controlFlow\\_part2.html](http://www.csoundjournal.com/2006summer/controlFlow_part2.html), écrit par Steven Yi. Et dans les Floss Manuals : <http://write.flossmanuals.net/csound/c-control-structures>

## Crédits

Istvan Varga. 2006

Nouveau dans la version 5.01 de Csound



# loop\_gt

loop\_gt — Constructions de boucle..

## Description

Construction d'opérations de boucle.

## Syntaxe

```
loop_gt    indx, idecr, imin, label
```

```
loop_gt    kndx, kdecr, kmin, label
```

## Initialisation

*indx* -- variable de taux-i, compteur de la boucle.

*idecr* -- valeur de décrémentation de la boucle.

*imin* -- valeur minimale de l'index de la boucle.

## Exécution

*kndx* -- variable de taux-k, compteur de la boucle.

*kdecr* -- valeur de décrémentation de la boucle.

*kmin* -- valeur minimale de l'index de la boucle.

L'action de **loop\_gt** est équivalente à

```
indx = indx - idecr
if (indx > imin) igoto label
```

ou à

```
kndx = kndx - kdecr
if (kndx > kmin) kgoto label
```

## Voir aussi

*loop\_ge*, *loop\_le* et *loop\_lt*.

Plus d'information sur cet opcode : [http://www.csoundjournal.com/2006summer/controlFlow\\_part2.html](http://www.csoundjournal.com/2006summer/controlFlow_part2.html), écrit par Steven Yi. Et dans les Floss Manuals : <http://write.flossmanuals.net/csound/c-control-structures>

## Crédits

Istvan Varga.

Nouveau dans la version 5.01 de Csound

# loop\_le

loop\_le — Constructions de boucle.

## Description

Construction d'opérations de boucle.

## Syntaxe

```
loop_le    indx, incr, imax, label
```

```
loop_le    kndx, kncr, kmax, label
```

## Initialisation

*indx* -- variable de taux-i, compteur de la boucle.

*incr* -- valeur d'incrément de la boucle.

*imax* -- valeur maximale de l'index de la boucle.

## Exécution

*kndx* -- variable de taux-k, compteur de la boucle.

*knrc* -- valeur d'incrément de la boucle.

*kmax* -- valeur maximale de l'index de la boucle.

L'action de **loop\_le** est équivalente à

```
indx = indx + incr
if (indx <= imax) igoto label
```

ou à

```
kndx = kndx + knrc
if (kndx <= kmax) kgoto label
```

## Exemples

Voici un exemple de l'opcode loop\_le. Il utilise le fichier *loop\_le.csd* [examples/loop\_le.csd].

### Exemple 497. Exemple de l'opcode loop\_le.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
```

```

; For Non-realtime ouput leave only the line below:
; -o loop_le.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

seed 0
gisine ftgen 0, 0, 2^10, 10, 1

instr 1 ;master instrument

ininstr = 5 ;number of called instances
indx = 0
loop:
    prints "play instance %d\\n", indx
    ipan random 0, 1
    ifreq random 100, 1000
    iamp = 1/ininstr
    event_i "i", 10, 0, p3, iamp, ifreq, ipan
    loop_le indx, 1, ininstr, loop

endin

instr 10

ipeak random 0, 1 ;where is the envelope peak
asig poscil3 p4, p5, gisine
aenv transeg 0, p3*ipeak, 6, 1, p3-p3*ipeak, -6, 0
aL,aR pan2 asig*aenv, p6
outs aL, aR

endin

</CsInstruments>
<CsScore>
i1 0 10
e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

play instance 0
play instance 1
play instance 2
play instance 3
play instance 4
play instance 5

```

## Voir aussi

*loop\_ge*, *loop\_gt* et *loop\_lt*.

Plus d'information sur cet opcode : [http://www.csoundjournal.com/2006summer/controlFlow\\_part2.html](http://www.csoundjournal.com/2006summer/controlFlow_part2.html), écrit par Steven Yi. Et dans les Floss Manuals : <http://write.flossmanuals.net/csound/c-control-structures>

## Crédits

Istvan Varga.

Nouveau dans la version 5.01 de Csound

# loop\_lt

loop\_lt — Constructions de boucle.

## Description

Construction d'opérations de boucle.

## Syntaxe

```
loop_lt    indx, incr, imax, label
loop_lt    kndx, kncr, kmax, label
```

## Initialisation

*indx* -- variable de taux-i, compteur de la boucle.

*incr* -- valeur d'incrément de la boucle.

*imax* -- valeur maximale de l'index de la boucle.

## Exécution

*kndx* -- variable de taux-k, compteur de la boucle.

*knrc* -- valeur d'incrément de la boucle.

*kmax* -- valeur maximale de l'index de la boucle.

L'action de **loop\_lt** est équivalente à

```
indx = indx + incr
if (indx < imax) igoto label
```

ou à

```
kndx = kndx + knrc
if (kndx < kmax) kgoto label
```

## Exemples

Voici un exemple de l'opcode loop\_lt. Il utilise le fichier *loop\_lt.csd* [examples/loop\_lt.csd].

### Exemple 498. Exemple de l'opcode loop\_lt.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;;realtime audio out
```

```

;-iadc    ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o loop_lt.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

seed 0
gisine ftgen 0, 0, 2^10, 10, 1

instr 1 ;master instrument

ininstr = 5 ;number of called instances
indx = 0
loop:
  prints "play instance %d\\n", indx
  ipan random 0, 1
  ifreq random 100, 1000
  iamp = 1/ininstr
  event_i "i", 10, 0, p3, iamp, ifreq, ipan
  loop_lt indx, 1, ininstr, loop

endin

instr 10

ipeak random 0, 1 ;where is the envelope peak
asig poscil3 p4, p5, gisine
aenv transeg 0, p3*ipeak, 6, 1, p3-p3*ipeak, -6, 0
aL,aR pan2 asig*aenv, p6
outs aL, aR

endin

</CsInstruments>
<CsScore>
i1 0 10
e
</CsScore>
</CsSoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

play instance 0
play instance 1
play instance 2
play instance 3
play instance 4

```

## Voir aussi

*loop\_ge*, *loop\_gt* et *loop\_le*.

Plus d'information sur cet opcode : [http://www.csoundjournal.com/2006summer/controlFlow\\_part2.html](http://www.csoundjournal.com/2006summer/controlFlow_part2.html), écrit par Steven Yi. Et dans les Floss Manuals : <http://write.flossmanuals.net/csound/c-control-structures>

## Crédits

Istvan Varga.

Nouveau dans la version 5.01 de Csound

# loopseg

**loopseg** — Génère un signal de contrôle constitué de segments de droite délimités par deux ou plus points spécifiés.

## Description

Génère un signal de contrôle constitué de segments de droite délimités par deux ou plus points spécifiés. L'enveloppe entière est parcourue en boucle au taux *kfreq*. Chaque paramètre peut varier au taux-k.

## Syntaxe

```
ksig loopseg kfreq, ktrig, iphase, kvalue0, ktime0 [, kvalue1] [, ktime1] \  
      [, kvalue2] [, ktime2] [...]
```

## Initialisation

*iphase* -- Une valeur comprise entre 0 et 1 pour indiquer où commence la boucle. Zéro, la valeur la plus usuelle, indique le début du signal.

## Exécution

*ksig* -- Signal de sortie.

*kfreq* -- Taux de répétition en Hz ou en fraction de Hz.

*ktrig* -- S'il est non nul, redéclanche l'enveloppe depuis le début (voir l'opcode *trigger*), avant que le cycle de l'enveloppe ne soit complet.

*kvalue0...kvalueN* -- Valeurs des points.

*ktime0...ktimeN* -- Durées entre les points ; exprimées en fraction d'une période (voir ci-dessous). La dernière durée indique une ligne entre la valeur finale et la première valeur.

L'opcode *loopseg* est semblable à *linseg*, mais l'enveloppe entière est parcourue en boucle au taux *kfreq*. Noter que les valeurs temporelles ne sont pas exprimées en secondes mais en fractions d'une période. Concrètement chaque durée est proportionnelle aux autres, et la durée du cycle complet est proportionnelle à la somme de toutes les valeurs de durée.

La somme de toutes les durées est ensuite pondérée en fonction de l'argument *kfreq*. Par exemple, considérant une enveloppe faite de 3 segments, chaque segment ayant une valeur de durée de 100, leur somme sera 300. Cette valeur représente la durée totale de l'enveloppe, et elle est divisée en 3 parties égales, une partie pour chaque segment.

Concrètement, la durée réelle de l'enveloppe en secondes est déterminée par *kfreq*. Si l'enveloppe est à nouveau constituée de 3 segments, mais cette fois-ci le premier et le dernier segments ayant une durée de 50, tandis que le segment central a une durée de 100, leur somme sera 200. Ici 200 représente la durée totale des 3 segments, et ainsi le segment central sera deux fois plus long que les autres segments.

Tous les paramètres peuvent varier au taux-k. Si les valeurs de fréquence sont négatives, l'enveloppe est lue à l'envers. *ktime0* doit toujours valoir 0, sauf si l'on désire un effet spécial.



## Exemples

Voici un exemple de l'opcode `loopseg`. Il utilise le fichier `loopseg.csd` [examples/loopseg.csd].

### Exemple 499. Exemple de l'opcode `loopseg`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o loopseg.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
kfreq init      p4 ; frequency of loop repetition
ifrac =         p5 ; frequency ratio of restart triggers
ktrig metro     kfreq * ifrac ; triggers to restart loop
iphase =        0 ; initial phase
; loop of note values (some glissandi)
;                               val dur val dur etc...
knote loopseg kfreq, ktrig, iphase, 40, 1, 40, 0, 43,1,43,0, 49,2,48,0, \
  47,1,47,0, 46,1,46,0, 46,1,47,0, 49,1,49,0, 43,1,43,0, 46,1,46,0, 40,1,39,0
; loop of filter cutoff values (oct format). This loop, half speed of note loop.
kcfoct loopseg kfreq*0.5, ktrig, iphase, 11,2,4,0, 12,1,4,0, 13,1,4,0, \
  11.5,3,4,0, 12.5,1,4,0, 13,2,4,0, 12.5,1,4,0
kenv linseg 0,0.01,1,p3-5.01,1,5,0
ioct =      int((rnd(0.999)*4)-2) ; random value either -1, 0 or 1
asig vco2 0.2*kenv,cpsmidinn(knote)*octave(ioct),0 ; sawtooth
asig moogladder asig,cpsoct(kcfoct),rnd(0.6) ; filter sawtooth
aL,aR pan2 outsig,rnd(1) ; random static pan location
outs aL, aR
endin

</CsInstruments>
<CsScore>

; 4 layers, each with a different frequency of loop repetition (p4),
; frequency ratio of restart triggers (p5) and pan position (p6).
i 1 0 30 0.5 [11/19]
i 1 6 30 0.25 [11/13]
i 1 12 30 0.125 [11/16]
i 1 18 30 1 [11/12]
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*lpshold loopxseg*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la Version 4.13

# loopsegp

loopsegp — Signaux de contrôle basés sur des segments de droite.

## Description

Génère un signal de contrôle constitué de segments de droite délimités par deux ou plus points spécifiés. L'enveloppe entière peut être parcourue en boucle à une vitesse variable. Chaque coordonnée de segment peut aussi varier au taux-k.

## Syntaxe

```
ksig loopsegp kphase, kvalue0, kdur0, kvalue1 \  
[, kdur1, ... , kdurN-1, kvalueN]
```

## Exécution

*ksig* - signal de sortie.

*kphase* - point de la séquence lu, exprimé en fraction d'un cycle (de 0 à 1)

*kvalue0 ...kvalueN* - valeurs des points.

*kdur0 ...kdurN-1* - durée des points exprimée en fractions d'un cycle.

L'opcode *loopsegp* est semblable à *loopseg* ; la seule différence étant que, à la place de la fréquence, une phase variable est utilisée. Si l'on utilise un *phaseur* pour obtenir la valeur de la phase, on aura un comportement identique à celui de *loopseg*, mais on peut obtenir des résultats intéressants avec des phases à l'évolution non linéaire, ce qui rend *loopsegp* plus puissant et plus général que *loopseg*.

## Exemples

Voici un exemple de l'opcode *loopsegp*. Il utilise le fichier *loopsegp.csd* [examples/loopsegp.csd].

### Exemple 500. Exemple de l'opcode loopsegp.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac      ;;realtime audio out  
;-iadc     ;;uncomment -iadc if realtime audio input is needed too  
; For Non-realtime ouput leave only the line below:  
; -o loopsegp.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 32  
nchnls = 2  
0dbfs = 1  
  
instr 1  
ktrig metro 22/8 ; triggers used to generate new direction values  
kdir trandom ktrig,-2.99,2.99
```

```

kdir =      0.5*int(kdir) ; kdir will either -1, -0.5, 0, 0.5 or 1
; kphase - looping pointer
kphase phasor kdir
; a loop sequence of midi note numbers and durations
knote loopsegg kphase, 40,1,40,0, 43,1,43,0, 49,2,48,0, \
  47,1,47,0, 46,1,46,0, 46,1,47,0, 49,1,49,0, 43,1,43,0, 46,1,46,0, 40,1,39,0
kmul rspline 0.1,0.8,0.5,5 ; modulation of buzz tone
asig gbuzz 0.2, cpsmidinn(knote), 30, 3, kmul, 1 ; buzz tone
outs asig, asig

      schedkwhen ktrig,0,0,2,0,0.1 ; play metronome
endin

instr 2 ; metronome
acps expon 180+rnd(40),p3,50
aamp expon 0.05+rnd(0.05),p3,0.001
asig poscil aamp-0.001,acps,2
outs asig,asig
endin

</CsInstruments>
<CsScore>
; cosine wave.
f 1 0 16384 11 1
; sine wave.
f 2 0 16384 10 1

i 1 0 360 0.25

e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5. (Auparavant, disponible seulement dans CsoundAV)

# looptseg

**looptseg** — Génère un signal de contrôle constitué de segments linéaires ou exponentiels délimités par deux ou plus points spécifiés.

## Description

Génère un signal de contrôle constitué de segments linéaires ou exponentiels contrôlables délimités par deux ou plus points spécifiés. L'enveloppe entière est parcourue en boucle au taux *kfreq*. Chaque paramètre peut varier au taux-k.

## Syntaxe

```
ksig looptseg kfreq, ktrig, iphase, kvalue0, ktype0, ktime0, [, kvalue1] [,ktype1] [, ktime1] \  
[, kvalue2] [,ktype2] [, ktime2] [...] [, kvalueN] [,ktypeN] [, ktimeN]
```

## Initialisation

*iphase* -- Une valeur comprise entre 0 et 1 pour indiquer où commence la boucle. Zéro, la valeur la plus usuelle, indique le début du signal.

## Exécution

*k<sub>sig</sub>* -- Signal de sortie.

*kfreq* -- Taux de répétition en Hz ou en fraction de Hz.

*ktrig* -- S'il est non nul, redéclanche l'enveloppe depuis le début (voir l'opcode *trigger*), avant que le cycle de l'enveloppe ne soit complet.

*kvalue0...kvalueN* -- Valeurs des points.

*ktime0...ktimeN* -- Durées entre les points ; exprimées en fraction d'une période (voir ci-dessous). La dernière durée indique une ligne entre la valeur finale et la première valeur.

*ktype0...ktypeN* -- forme de l'enveloppe. Si la valeur est 0, la forme est linéaire ; sinon c'est une exponentielle concave (type positif) ou une exponentielle convexe (type négatif).

L'opcode *looptseg* est semblable à *transeg*, mais l'enveloppe entière est parcourue en boucle au taux *kfreq*. Noter que les valeurs temporelles ne sont pas exprimées en secondes mais en fractions d'une période. Concrètement chaque durée est proportionnelle aux autres, et la durée du cycle complet est proportionnelle à la somme de toutes les valeurs de durée.

La somme de toutes les durées est ensuite pondérée en fonction de l'argument *kfreq*. Par exemple, considérant une enveloppe faite de 3 segments, chaque segment ayant une valeur de durée de 100, leur somme sera 300. Cette valeur représente la durée totale de l'enveloppe, et elle est divisée en 3 parties égales, une partie pour chaque segment.

Concrètement, la durée réelle de l'enveloppe en secondes est déterminée par *kfreq*. Si l'enveloppe est à nouveau constituée de 3 segments, mais cette fois-ci le premier et le dernier segments ayant une durée de 50, tandis que le segment central a une durée de 100, leur somme sera 200. Ici 200 représente la durée totale des 3 segments, et ainsi le segment central sera deux fois plus long que les autres segments.

Tous les paramètres peuvent varier au taux-k. Si les valeurs de fréquence sont négatives, l'enveloppe est lue à l'envers. *ktime0* doit toujours valoir 0, sauf si l'on désire un effet spécial.

## Exemples

Voici un exemple de l'opcode *looptseg*. Il utilise le fichier *looptseg.csd* [examples/looptseg.csd].

### Exemple 501. Exemple de l'opcode *looptseg*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-o dac      ;;realtime audio out
-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o looptseg.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
kfreq = 1          ; frequency of loop repetition
ktrig init 0        ; loop restart trigger (not used)
iphase = 0          ; initial phase
ktyp line 6,p3,-6    ; explore the useful range of curve types
; loop of filter cutoff values (oct format)
;                                     value curve dur.
kcfoct looptseg kfreq, ktrig, iphase,13, ktyp, 1, \
                                     4, ktyp, 0, \
                                     11, ktyp, 1, \
                                     4

asig vco2 0.2,cpsmidinn(48),0          ; a sawtooth
asig moogladder asig,cpsoct(kcfoct),rnd(0.6) ; filter sawtooth
outs asig, asig
endin

</CsInstruments>
<CsScore>
i 1 0 12
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*lpshold looptseg*

## Crédits

Auteur : John ffitich

Nouveau dans la version 5.12

# loopxseg

loopxseg — Génère un signal de contrôle constitué de segments exponentiels délimités par deux ou plus points spécifiés.

## Description

Génère un signal de contrôle constitué de segments exponentiels délimités par deux ou plus points spécifiés. L'enveloppe entière est parcourue en boucle au taux *kfreq*. Chaque paramètre peut varier au taux-k.

## Syntaxe

```
ksig loopxseg kfreq, ktrig, iphase, ktime0, kvalue0 [, ktime1] [, kvalue1] \  
      [, ktime2] [, kvalue2] [...]
```

## Exécution

*ksig* -- Signal de sortie.

*kfreq* -- Taux de répétition en Hz ou en fraction de Hz.

*ktrig* -- S'il est non nul, redéclanche l'enveloppe depuis le début (voir l'opcode *trigger*), avant que le cycle de l'enveloppe ne soit complet.

*iphase* -- Une valeur comprise entre 0 et 1 pour indiquer où commence la boucle. Zéro, la valeur la plus usuelle, indique le début du signal.

*ktime0...ktimeN* -- Dates des points ; exprimées en fraction d'une période.

*kvalue0...kvalueN* -- Valeurs des points.

L'opcode *loopxseg* est semblable à *expseg*, mais l'enveloppe entière est parcourue en boucle au taux *kfreq*. Noter que les valeurs temporelles ne sont pas exprimées en secondes mais en fractions d'une période. Concrètement chaque durée est proportionnelle aux autres, et la durée du cycle complet est proportionnelle à la somme de toutes les valeurs de durée.

La somme de toutes les durées est ensuite pondérée en fonction de l'argument *kfreq*. Par exemple, considérant une enveloppe faite de 3 segments, chaque segment ayant une valeur de durée de 100, leur somme sera 300. Cette valeur représente la durée totale de l'enveloppe, et elle est divisée en 3 parties égales, une partie pour chaque segment.

Concrètement, la durée réelle de l'enveloppe en secondes est déterminée par *kfreq*. Si l'enveloppe est à nouveau constituée de 3 segments, mais cette fois-ci le premier et le dernier segments ayant une durée de 50, tandis que le segment central a une durée de 100, leur somme sera 200. Ici 200 représente la durée totale des 3 segments, et ainsi le segment central sera deux fois plus long que les autres segments.

Tous les paramètres peuvent varier au taux-k. Si les valeurs de fréquence sont négatives, l'enveloppe est lue à l'envers. *ktime0* doit toujours valoir 0, sauf si l'on désire un effet spécial.

## Exemples

Voici un exemple de l'opcode *loopxseg*. Il utilise le fichier *loopxseg.csd* [examples/loopxseg.csd].

## Exemple 502. Exemple de l'opcode loopxseg.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o loopxseg.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
kfreq rspline 0.01,20,0.2,1 ; freq. of loop repetition created by random spline
ktrig init 0 ; loop restart trigger (not used)
iphase = 0 ; initial phase
; loop of filter cutoff values (oct format). Rescaled further down.
kcfoct loopxseg kfreq, ktrig, iphase, 1,1,0,0
kenv linseg 0,0.01,1,p3-5.01,1,5,0
asig vco2 0.2*kenv,cpsmidinn(48),0
kdep rspline 5,8,0.2,1 ; filter depth created by a random spline
kcf port cpsoct((kcfoct*kdep)+4), 0.001 ; smooth filter changes
asig moogladder asig,kcf,rnd(0.6)
aL,aR pan2 asig,rnd(1)
outs aL, aR
endin

</CsInstruments>
<CsScore>
i 1 0 60
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*lpshold loopseg*

## Crédits

Auteur : John ffitch

Nouveau dans la version 5.12



# lorenz

lorenz — Implémente le système d'équations de Lorenz.

## Description

Implémente le système d'équations de Lorenz. Le système de Lorenz est un système dynamique chaotique qui fut utilisé à l'origine pour simuler le mouvement d'une particule dans des courants de convection et des systèmes météorologiques simplifiés. De petites différences dans les conditions initiales conduisent rapidement à des valeurs divergentes. C'est ce qu'on appelle parfois l'effet papillon. Si un papillon bat des ailes en Australie, cela aura des conséquences sur le temps en Alaska. Ce système est l'un des éléments fondateurs du développement de la théorie du chaos. Il est utile comme source audio chaotique ou comme source de modulation basse fréquence.

## Syntaxe

```
ax, ay, az lorenz ksv, krsv, kbv, kh, ix, iy, iz, iskip [, iskipinit]
```

## Initialisation

*ix, iy, iz* -- les coordonnées initiales de la particule.

*iskip* -- utilisé pour sauter des valeurs générées. Si *iskip* vaut 5, seulement une valeur sur cinq sera retournée. Utile pour générer des sons de hauteur plus élevée.

*iskipinit* (facultatif, 0 par défaut) -- s'il est non nul, l'initialisation du filtre sera ignorée. (Nouveau dans les versions 4.23f13 et 5.0 de Csound)

## Exécution

*ksv* -- le nombre de Prandtl ou sigma

*krv* -- le nombre de Rayleigh

*kbv* -- le rapport entre la longueur et la largeur de la boîte dans laquelle les courants de convection sont générés

*kh* -- le pas de progression utilisé pour le calcul approché de l'équation différentielle. On peut l'utiliser pour contrôler la hauteur du système. Des valeurs comprises entre 0,1 et 0,001 sont typiques.

Le calcul approché des équations se fait comme suit :

$$\begin{aligned}x &= x + h*(s*(y - x)) \\ y &= y + h*(-x*z + r*x - y) \\ z &= z + h*(x*y - b*z)\end{aligned}$$

Les valeurs historiques des paramètres sont :

$ks = 10$

kr = 28  
kb = 8/3



## Note

Cet algorithme utilise des boucles de rétroaction internes non linéaires ce qui fait dépendre le résultat audio du taux d'échantillonnage de l'orchestre. Par exemple, si l'on développe un projet avec  $sr=48000\text{Hz}$  et si l'on veut produire un CD audio de ce projet, il faut enregistrer un fichier avec  $sr=48000\text{Hz}$ , puis sous-échantillonner ce fichier à  $44100\text{Hz}$  avec l'utilitaire *src\_conv*.

## Exemples

Voici un exemple de l'opcode *lorenz*. Il utilise le fichier *lorenz.csd* [examples/lorenz.csd].

### Exemple 503. Exemple de l'opcode *lorenz*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lorenz.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 2

; Instrument #1 - a lorenz system in 3D space.
instr 1
; Create a basic tone.
kamp init 25000
kcps init 1000
ifn = 1
asnd oscil kamp, kcps, ifn

; Figure out its X, Y, Z coordinates.
ksv init 10
krv init 28
kbv init 2.667
kh init 0.0003
ix = 0.6
iy = 0.6
iz = 0.6
iskip = 1
ax1, ay1, az1 lorenz ksv, krv, kbv, kh, ix, iy, iz, iskip

; Place the basic tone within 3D space.
kx downsamp ax1
ky downsamp ay1
kz downsamp az1
idist = 1
ift = 0
```

```
imode = 1
imdel = 1.018853416
iovr = 2
aw2, ax2, ay2, az2 spat3d asnd, kx, ky, kz, idist, \
                                ift, imode, imdel, iovr

; Convert the 3D sound to stereo.
aleft = aw2 + ay2
aright = aw2 - ay2

outs aleft, aright
endin

</CsInstruments>
<CsScore>

; Table #1 a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 5 seconds.
i 1 0 5
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Hans Mikelson  
Février 1999

Nouveau dans la version 3.53 de Csound

Note ajoutée par François Pinot, août 2009

# lorisread

**lorisread** — Importe un ensemble de partiels à largeur de bande améliorée depuis un fichier de données au format SDIF, leur applique des enveloppes de mise en forme de fréquence, d'amplitude et de largeur de bande au taux-k et stocke les partiels modifiés en mémoire.

## Syntaxe

```
lorisread ktimpt, ifilcod, istoreidx, kfreqenv, kampenv, kbwenv[, ifadetime]
```

## Description

*lorisread* importe un ensemble de partiels à largeur de bande améliorée depuis un fichier de données au format SDIF, leur applique des enveloppes de mise en forme de fréquence, d'amplitude et de largeur de bande au taux-k et stocke les partiels modifiés en mémoire.

## Initialisation

*ifilcod* - nombre entier ou chaîne de caractères dénotant un fichier de contrôle dérivé de l'analyse d'un signal audio par réassignation de largeur de bande améliorée. Un nombre entier indique le suffixe d'un fichier loris.sdif (par exemple loris.sdif.1) ; une chaîne de caractères (entre guillemets) donne un nom de fichier, qui peut être un nom de chemin complet. Si ce n'est pas un nom de chemin complet, le fichier est d'abord cherché dans le répertoire courant, puis dans celui qui est donné par la variable d'environnement SADIR (si elle est définie). Le fichier de données de réassignation de largeur de bande améliorée contient des valeurs de points charnières de fréquence, d'amplitude, de caractère bruiteux et d'enveloppe de phase organisés pour une synthèse additive à largeur de bande améliorée. Les données de contrôle doivent être conformes à l'un des formats SDIF existant.

Loris stocke les partiels dans des trames SDIF RBEP. Chaque trame RBEP contient une matrice RBEP et chaque ligne d'une matrice RBEP décrit un point charnière d'un partiel de Loris. Une trame RBEL contenant une matrice RBEL qui décrit l'étiquetage des partiels peut précéder la première trame RBEP dans le fichier SDIF. Les définitions des trames et des matrices RBEP et RBEL sont incluses dans l'en-tête du fichier SDIF. En plus des trames RBEP, Loris peut également lire et écrire des trames SDIF ITRC. Comme les trames ITRC ne représentent pas une amélioration de largeur de bande ou les coordonnées temporelles exactes de points charnière de Loris, leur utilisation est déconseillée. Les possibilités ITRC sont fournies pour pouvoir échanger des données avec des programmes qui ne peuvent pas traiter les trames RBEP.

*istoreidx*, *ireadidx*, *isrcidx*, *itgtidx* sont des étiquettes qui identifient un ensemble stocké de partiels à largeur de bande améliorée. *lorisread* importe les partiels depuis un fichier SDIF et les stocke avec l'étiquette entière *istoreidx*. *lorismorph* réalise un morphing des ensembles de partiels étiquetés *isrcidx* et *itgtidx*, et stocke les partiels résultants avec l'étiquette entière *istoreidx*. *lorisplay* restitue les partiels stockés avec l'étiquette *ireadidx*. Les étiquettes ne sont utilisées qu'à l'initialisation de la note, et l'on peut les réutiliser sans coût supplémentaire ou bénéfice en efficacité, et sans introduire d'interaction entre les instruments ou leurs instances.

*ifadetime* (*facultatif*) - En général, les partiels exportés depuis Loris commencent et se terminent avec une amplitude non nulle. Afin d'éviter les artefacts, il est très souvent nécessaire de d'introduire et de terminer progressivement les partiels au lieu de les commencer et de les terminer de façon abrupte. Si *ifadetime* est différent de zéro, les partiels ont une attaque progressive et une chute dégressive. Ceci est réalisé en ajoutant deux points charnière à chaque partiel : un *ifadetime* secondes avant le début et un autre *ifadetime* secondes après la fin. (Cependant, aucun point charnière n'est ajouté à une date inférieure à zéro.

Si nécessaire, la durée de l'introduction progressive est raccourcie.) Les points charnière supplémentaires au début et à la fin du partiel auront respectivement la même fréquence et la même largeur de bande que celle du premier et du dernier point charnière du partiel mais leurs amplitudes seront nulles. La phase des nouveaux points charnière sera extrapolée pour préserver la cohérence de phase. Si aucune valeur n'est spécifiée, *ifadetime* vaut par défaut zéro. Notez que la valeur de *ifadetime* peut être approchée, car les enveloppes des paramètres des partiels sont échantillonnées au taux de contrôle (taux-k) et indexées par *ktimpnt* (voir ci-dessous), et non pas en temps réel.

## Exécution

*lorisread* lit des données pré-calculées d'analyse par réassignation de largeur de bande améliorée depuis un fichier stocké au format SDIF, comme décrit ci-dessus. L'écoulement du temps dans ce fichier est spécifié par *ktimpnt* qui représente le temps en secondes. *ktimpnt* doit toujours être positif mais peut avancer ou reculer dans le temps, être stationnaire ou discontinu, comme un pointeur dans le fichier d'analyse. *kfreqenv* est un facteur de transposition au taux de contrôle, 1.5 transposant vers l'aigu d'une quinte juste, et 0.5 transposant vers le grave d'une octave. *kampenv* est un facteur de mise à l'échelle au taux de contrôle qui est appliqué à toutes les enveloppes d'amplitude des partiels. *kbwenv* est un facteur de mise à l'échelle au taux de contrôle qui est appliqué à toutes les enveloppes de largeur de bande ou de caractère bruiteux des partiels. Les données de partiel à largeur de bande améliorée sont stockées en mémoire avec une étiquette spécifiée pour un accès ultérieur par un autre générateur.

## Exemples

Voici un exemple de l'opcode *lorisread*. Il utilise le fichier *lorisread.csd* [examples/lorisread.csd].

### Exemple 504. Exemple de l'opcode *lorisread*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o lorisread.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

; Play the partials in clarinet.sdif from 0 to 3 sec with 1 ms fadetime
; and no frequency , amplitude, or bandwidth modification.

instr 1

ktime linseg 0, p3, 3 ; linear time function from 0 to 3 seconds
lorisread ktime, "clarinet.sdif", 1, 1, 1, 1, .001
asig lorisplay 1, 1, 1, 1
outs asig, asig
endin

; Play the partials in clarinet.sdif from 0 to 3 sec with 1 ms fadetime
```

```

; adding tuning and vibrato, increasing the "breathiness" (noisiness) and overall
; amplitude, and adding a highpass filter.

instr 2

ktime linseg 0, p3, 3 ; linear time function from 0 to 3 seconds
; compute frequency scale for tuning
ifscale = cpspch(p4)/cpspch(8.08) ; (original pitch was G#4)
; make a vibrato envelope
kvenv linseg 0, p3/6, 0, p3/6, .02, p3/3, .02, p3/6, 0, p3/6, 0
kvib oscil kvenv, 4, 1 ; table 1, sinusoid
kbwenv linseg 1, p3/6, 1, p3/6, 2, 2*p3/3, 2 ;lots of noise
lorisread ktime, "clarinet.sdif", 1, 1, 1, 1, .001
a1 lorisplay 1, ifscale*kvib, 2, kbwenv
asig atone a1, 1000 ; highpass filter, cutoff 1000 Hz
outs asig, asig

endin

</CsInstruments>
<CsScore>
; a sinus
f 1 0 4096 10 1

i 1 0 3
i 1 + 1
i 1 + 6
s

;
i 2 1 3 8.08
i 2 3.5 1 8.04
i 2 4 6 8.00
i 2 4 6 8.07
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Cette implémentation des générateurs unitaires Loris a été écrite par Kelly Fitz ([loris@cerlsoundgroup.org](mailto:loris@cerlsoundgroup.org) [<mailto:loris@cerlsoundgroup.org>]). Elle est modélisée sur un prototype d'implémentation du générateur unitaire *lorisplay* écrit par Corbin Champion, et basé sur la méthode de synthèse additive à largeur de bande améliorée et sur les algorithmes de métamorphose sonore implémentés dans la bibliothèque Loris pour la modélisation et la manipulation du son. Les opcodes ont été ensuite adaptés sous la forme d'un greffon de Csound 5 par Michael Gogins.

# lorismorph

*lorismorph* — réalise un morphing de deux ensembles stockés de partiels à largeur de bande améliorée et stocke un nouvel ensemble de partiels représentant le son transformé. Le morphing est réalisé par interpolation linéaire des enveloppes des paramètres (fréquence, amplitude et largeur de bande ou caractère bruiteux) des partiels à largeur de bande améliorée selon des fonctions de transformation de fréquence, d'amplitude et de largeur de bande au taux de contrôle.

## Syntaxe

`lorismorph isrcidx, itgtidx, istoreidx, kfreqmorphenv, kampmorphenv, kbwmorphenv`

## Description

*lorismorph* réalise le morphing de deux ensembles stockés de partiels à largeur de bande améliorée et stocke un nouvel ensemble de partiels représentant le son transformé. Le morphing est réalisé par interpolation linéaire des enveloppes des paramètres (fréquence, amplitude et largeur de bande ou caractère bruiteux) des partiels à largeur de bande améliorée selon des fonctions de transformation de fréquence, d'amplitude et de largeur de bande au taux de contrôle.

## Initialisation

*istoreidx*, *ireadidx*, *isrcidx*, *itgtidx* sont des étiquettes qui identifient un ensemble stocké de partiels à largeur de bande améliorée. *lorisread* importe les partiels depuis un fichier SDIF et les stocke avec l'étiquette entière *istoreidx*. *lorismorph* réalise un morphing des ensembles de partiels étiquetés *isrcidx* et *itgtidx*, et stocke les partiels résultants avec l'étiquette entière *istoreidx*. *lorisplay* restitue les partiels stockés avec l'étiquette *ireadidx*. Les étiquettes ne sont utilisées qu'à l'initialisation de la note, et l'on peut les réutiliser sans coût supplémentaire ou bénéfice en efficacité, et sans introduire d'interaction entre les instruments ou leurs instances.

## Exécution

*lorismorph* génère un ensemble de partiels à largeur de bande améliorée en effectuant le morphing de deux ensembles de partiels stockés, les partiels sources et cibles, qui peuvent avoir été importés au moyen de *lorisread*, ou générés par un autre générateur unitaire, y compris une autre instance de *lorismorph*. Le morphing est réalisé en interpolant les paramètres (étiquetés) des partiels correspondants dans les deux sources sonores. Le morphing sonore est décrit par trois enveloppes de morphing au taux de contrôle. *kfreqmorphenv* décrit l'interpolation des valeurs de fréquence des partiels dans les deux sources sonores. Lorsque *kfreqmorphenv* vaut 0, les fréquences des partiels sont obtenues à partir des partiels stockés à *isrcidx*. Lorsque *kfreqmorphenv* vaut 1, les fréquences des partiels sont obtenues à partir des partiels à *itgtidx*. Lorsque *kfreqmorphenv* est compris entre 0 et 1, les fréquences des partiels sont interpolées entre les partiels sources et cibles correspondants. Les interpolations de l'amplitude et de la largeur de bande (caractère bruiteux) des partiels sont décrites de manière similaire par *kampmorphenv* et par *kbwmorphenv*.

## Exemples

Voici un exemple de l'opcode *lorismorph*. Il utilise le fichier *lorismorph.csd* [exemples/lorismorph.csd].

### Exemple 505. Exemple de l'opcode *lorismorph*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o lorismorph.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
; clarinet.sdif and meow.sdif can be found in /manual/examples

; Morph the partials in meow.sdif into the partials in clarinet.sdif over the duration of
; the sustained portion of the two tones (from .2 to 2.0 seconds in the meow, and from
; .5 to 2.1 seconds in the clarinet). The onset and decay portions in the morphed sound are
; specified by parameters p4 and p5, respectively. The morphing time is the time between the
; onset and the decay. The meow partials are shifted in pitch to match the pitch of the clarinet
; tone (D above middle C).

instr 1

ionset = p4
idecay = p5
itmorph = p3 - (ionset + idecay)
ipshift = cpspch(8.02)/cpspch(8.08)

ktme linseg 0, ionset, .2, itmorph, 2.0, idecay, 2.1 ; meow time function, morph from .2 to 2.0 seconds
ktcl linseg 0, ionset, .5, itmorph, 2.1, idecay, 2.3 ; clarinet time function, morph from .5 to 2.1 seconds
kmurph linseg 0, ionset, 0, itmorph, 1, idecay, 1
  lorisread ktme, "meow.sdif", 1, ipshift, 2, 1, .001
  lorisread ktcl, "clarinet.sdif", 2, 1, 1, 1, .001
  lormorph 1, 2, 3, kmurph, kmurph, kmurph
asig lorisplay 3, 1, 1, 1
  outs asig, asig
endin

; Morph the partials in clarinet.sdif into the partials in meow.sdif. The start and end times
; for the morph are specified by parameters p4 and p5, respectively. The morph occurs over the
; second of four pitches in each of the sounds, from .75 to 1.2 seconds in the flutter-tongued
; clarinet tone, and from 1.7 to 2.2 seconds in the cat's meow. Different morphing functions are
; used for the frequency and amplitude envelopes, so that the partial amplitudes make a faster
; transition from clarinet to cat than the frequencies. (The bandwidth envelopes use the same
; morphing function as the amplitudes.)

instr 2

ionset = p4
imorph = p5 - p4
irelease = p3 - p5

ktclar linseg 0, ionset, .75, imorph, 1.2, irelease, 2.4
ktmeow linseg 0, ionset, 1.7, imorph, 2.2, irelease, 3.4

kmfreq linseg 0, ionset, 0, .75*imorph, .25, .25*imorph, 1, irelease, 1
kmamp linseg 0, ionset, 0, .75*imorph, .9, .25*imorph, 1, irelease, 1

  lorisread ktclar, "clarinet.sdif", 1, 1, 1, 1, .001
  lorisread ktmeow, "meow.sdif", 2, 1, 1, 1, .001
  lormorph 1, 2, 3, kmfreq, kmamp, kmamp
asig lorisplay 3, 1, 1, 1
  outs asig, asig

```



```
endin

</CsInstruments>
<CsScore>
;      strt  dur  onset  decay
i 1    0     3    .25    .15
i 1    +     1    .10    .10
i 1    +     6    1.     1.

;      strt  dur  morph_start  morph_end
i 2    9     4    .75          2.75

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Cette implémentation des générateurs unitaires Loris a été écrite par Kelly Fitz ([loris@cerlsoundgroup.org](mailto:loris@cerlsoundgroup.org) [<mailto:loris@cerlsoundgroup.org>]). Elle est modélisée sur un prototype d'implémentation du générateur unitaire *lorisplay* écrit par Corbin Champion, et basé sur la méthode de synthèse additive à largeur de bande améliorée et sur les algorithmes de métamorphose sonore implémentés dans la bibliothèque Loris pour la modélisation et la manipulation du son. Les opcodes ont été ensuite adaptés sous la forme d'un greffon de Csound 5 par Michael Gogins.

# lorisplay

*lorisplay* — restitue un ensemble stocké de partiels à largeur de bande améliorée en utilisant la méthode de synthèse additive à largeur de bande améliorée implémentée dans le logiciel Loris, et en appliquant au taux de contrôle des enveloppes de mise en forme de la fréquence, de l'amplitude et de la largeur de bande.

## Syntaxe

ar **lorisplay** ireadidx, kfreqenv, kampenv, kbwenv

## Description

*lorisplay* restitue un ensemble stocké de partiels à largeur de bande améliorée en utilisant la méthode de synthèse additive à largeur de bande améliorée implémentée dans le logiciel Loris, et en appliquant au taux de contrôle des enveloppes de mise en forme de la fréquence, de l'amplitude et de la largeur de bande.

## Initialisation

*istoreidx*, *ireadidx*, *isrcidx*, *itgtidx* sont des étiquettes qui identifient un ensemble stocké de partiels à largeur de bande améliorée. *lorisread* importe les partiels depuis un fichier SDIF et les stocke avec l'étiquette entière *istoreidx*. *lorismorph* réalise un morphing des ensembles de partiels étiquetés *isrcidx* et *itgtidx*, et stocke les partiels résultants avec l'étiquette entière *istoreidx*. *lorisplay* restitue les partiels stockés avec l'étiquette *ireadidx*. Les étiquettes ne sont utilisées qu'à l'initialisation de la note, et l'on peut les réutiliser sans coût supplémentaire ou bénéfice en efficacité, et sans introduire d'interaction entre les instruments ou leurs instances.

## Exécution

*lorisplay* implémente la reconstruction de signal au moyen de la synthèse additive à largeur de bande améliorée. Les données de contrôle sont obtenues à partir d'un ensemble stocké de partiels à largeur de bande améliorée importé depuis un fichier SDIF en utilisant *lorisread* ou construit par un autre générateur unitaire tel que *lorismorph*. *kfreqenv* est un facteur de transposition au taux de contrôle : une valeur de 1 signifie pas de transposition, 1.5 transpose vers l'aigu d'une quinte juste, et 0.5 transpose vers le grave d'une octave. *kampenv* est un facteur de mise à l'échelle au taux de contrôle qui est appliqué à toutes les enveloppes d'amplitude des partiels. *kbwenv* est un facteur de mise à l'échelle au taux de contrôle qui est appliqué à toutes les enveloppes de largeur de bande ou de caractère bruiteux des partiels. Les données de partiel à largeur de bande améliorée sont stockées en mémoire avec une étiquette spécifiée pour un accès ultérieur par un autre générateur.

## Exemples

Voici un exemple de l'opcode *lorisplay*. Il utilise le fichier *lorisplay.csd* [examples/lorisplay.csd].

### Exemple 506. Exemple de l'opcode *lorisplay*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;;realtime audio out
```

```

;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o lorisplay.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
; clarinet.sdif can be found in /manual/examples

; Play the partials in clarinet.sdif from 0 to 3 sec with 1 ms fadetime
; and no frequency , amplitude, or bandwidth modification.

instr 1

ktime linseg 0, p3, 3.0 ; linear time function from 0 to 3 seconds
lorisread ktime, "clarinet.sdif", 1, 1, 1, 1, .001
kfrq = p4 ; pitch shifting
asig lorisplay 1, kfrq, 1, 1
outs asig, asig
endin

; Play the partials in clarinet.sdif from 0 to 3 sec with 1 ms fadetime
; adding tuning and vibrato, increasing the "breathiness" (noisiness) and overall
; amplitude, and adding a highpass filter.

instr 2

ktime linseg 0, p3, 3.0 ; linear time function from 0 to 3 seconds
; compute frequency scale for tuning

ifscale = cpspch(p4)/cpspch(8.08) ; (original pitch was G#4)
; make a vibrato envelope
kvenv linseg 0, p3/6, 0, p3/6, .02, p3/3, .02, p3/6, 0, p3/6, 0
kvib oscil kvenv, 4, 1 ; table 1, sinusoid
kbwenv linseg 1, p3/6, 1, p3/6, 100, 2*p3/3, 100 ;lots of noise
lorisread ktime, "clarinet.sdif", 1, 1, 1, 1, .001
a1 lorisplay 1, ifscale*kvib, 2, kbwenv
asig atone a1, 1000 ; highpass filter, cutoff 1000 Hz
outs asig, asig
endin

</CsInstruments>
<CsScore>
; a sinusoid
f 1 0 4096 10 1

i 1 0 3 1.2 ; shifted up
i 1 + 1 1.5
i 1 + 6 .5 ; shifted down
s

;      strt   dur   ptch
i 2      1     3    8.08
i 2      3.5   1    8.04
i 2      4     6    8.00
i 2      4     6    8.07
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Cette implémentation des générateurs unitaires Loris a été écrite par Kelly Fitz ([loris@cerlsoundgroup.org](mailto:loris@cerlsoundgroup.org) [<mailto:loris@cerlsoundgroup.org>]). Elle est modélisée sur un prototype d'implémentation du générateur unitaire *lorisplay* écrit par Corbin Champion, et basé sur la méthode de synthèse additive à largeur de bande améliorée et sur les algorithmes de métamorphose sonore implémentés dans la bibliothèque Loris pour la modélisation et la manipulation du son. Les opcodes ont été ensuite adaptés sous la forme d'un greffon de Csound 5 par Michael Gogins.

# loscil

loscil — Lit un son échantillonné depuis une table.

## Description

Lit un son échantillonné (mono ou stéréo) depuis une table, avec des boucles facultatives d'entretien et de relâchement. Une version donnant la position exacte dans la table (phase) correspondant à l'échantillon en sortie est fournie comme opcode alternatif.

## Syntaxe

```
ar1 [,ar2] loscil xamp, kcps, ifn [, ibas] [, imod1] [, ibeg1] [, iend1] \  
    [, imod2] [, ibeg2] [, iend2]  
  
aph, ar1 [,ar2] loscilphs xamp, kcps, ifn [, ibas] [, imod1] [, ibeg1] [, iend1] \  
    [, imod2] [, ibeg2] [, iend2]
```

## Initialisation

*ifn* -- numéro de table de fonction, contenant typiquement un son échantillonné avec des points de boucle précisés, remplie au moyen de *GEN01*. Le fichier source peut être mono ou stéréo.

*ibas* (facultatif) -- fréquence de base en Hz du son enregistré. Elle remplace éventuellement la fréquence donnée dans le fichier audio, mais devient nécessaire si le fichier n'en contient pas. La valeur par défaut est 261,626 Hz, c-à-d le do médian. (Nouveau dans Csound 4.03). Si la valeur est inconnue ou absente il faut utiliser 1 ici et dans *kcps*.

*imod1*, *imod2* (facultatif, -1 par défaut) -- modes d'interprétation des boucles d'entretien et de relâchement. Une valeur de 1 signifie une boucle normale, 2 signifie une boucle à l'endroit et à l'envers, 0 signifie pas de boucle. La valeur par défaut (-1) s'en remet au mode et aux points de boucle définis dans le fichier source. Il faut s'assurer de choisir un mode approprié si le fichier ne contient pas cette information.

*ibeg1*, *iend1*, *ibeg2*, *iend2* (facultatifs, dépendants de *mod1*, *mod2*) -- début et fin des boucles d'entretien et de relâchement. Ils sont mesurés en *trames d'échantillon* depuis le début du fichier, et auront ainsi la même valeur que le son soit mono ou stéréo. Si aucun point de boucle n'est spécifié et qu'un mode de boucle est donné (*imod1*, *imod2*, le fichier sera lu en boucle sur toute sa longueur.

## Exécution

*aph* -- la position normalisée dans la table correspondant à l'échantillon en sortie (seulement pour *loscilphs*).

*ar1*, *ar2* -- la sortie de taux audio. Il n'y a que *ar1* pour une sortie mono, alors qu'il y a *ar1* et *ar2* pour une sortie stéréo.

*xamp* -- l'amplitude du signal de sortie.

*kcps* -- la fréquence du signal de sortie en Hz.

*loscil* parcourt la ftable audio à un taux déterminé par *kcps*, en multipliant le résultat par *xamp*. L'incrément de lecture pour *kcps* dépend de la fréquence de base de la table *ibas*, et il est automatiquement ajusté si

le taux d'échantillonnage *sr* de l'orchestre diffère de celui auquel la source a été enregistrée. Dans cette unité, *f*table est toujours lue avec interpolation.

Si la lecture atteint la fin de la *boucle d'entretien* et que la boucle est active, le point de lecture sera modifié et *loscil* continuera sa lecture depuis l'intérieur de la boucle. Une fois que l'instrument reçoit un signal *turnoff* (depuis la partition ou depuis un évènement MIDI noteoff), la fin de la boucle est ignorée et la lecture continue vers la fin de la *boucle de relâchement*, ou vers le dernier échantillon (dorénavant vers zéro).

*loscil* est l'unité de base pour bâtir un échantillonneur. Avec un ensemble suffisamment conséquent de sons de piano échantillonnés, par exemple, cette unité peut les rééchantillonner pour simuler les hauteurs manquantes. La détection de la source de son la plus proche d'une hauteur donnée peut être réalisée par la consultation d'une table. Une fois qu'un instrument échantillonneur est actif, son point de *turnoff* peut être imprévisible et nécessiter une enveloppe de *relâchement* externe ; on réalise souvent cela en munissant le son échantillonné d'un détecteur *linenr*, qui allonge la durée d'un instrument à la fin de la note d'une durée spécifique car il implémente une chute.

Si l'on veut boucler sur tout le fichier, il faut spécifier un mode de boucle dans *imodl* et ne donner aucune valeur pour *ibeg* et *iend*.



## Note pour les utilisateurs de Windows

Les utilisateurs de Windows utilisent normalement l'antislash, « \ », lorsqu'ils écrivent les chemins de leurs fichiers. Par exemple, un utilisateur de Windows pourra utiliser le chemin « c:\music\samples\loop001.wav ». Ceci pose problème car les l'antislash est normalement utilisé pour spécifier des caractères spéciaux.

Pour écrire correctement ce chemin dans Csound, on peut :

- Soit utiliser le slash : c:/music/samples/loop001.wav
- Soit utiliser le caractère spécial d'antislash, « \\ » : c:\\music\\samples\\loop001.wav



## Note

Voici *loscil* en mono :

```
a1 loscil 10000, 1, 1, 1 ,1
```

... et *loscil* en stéréo :

```
a1, a2 loscil 10000, 1, 1, 1 ,1
```

## Exemples

Voici un exemple de l'opcode *loscil*. Il utilise les fichiers *loscil.csd* [examples/loscil.csd] *mary.wav* [examples/mary.wav] et *kickroll.wav* [examples/kickroll.wav].

### Exemple 507. Exemple de l'opcode *loscil*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o loscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ichnls = fchnls(p4)
print ichnls

if (ichnls == 1) then
  asigL loscil .8, 1, p4, 1
  asigR = asigL
elseif (ichnls == 2) then
  asigL, asigR loscil .8, 1, p4, 1
; safety precaution if not mono or stereo
else
  asigL = 0
  asigR = 0
endif
outs asigL, asigR

endin
</CsInstruments>
<CsScore>
f 1 0 0 1 "mary.wav" 0 0 0
f 2 0 0 1 "kickroll.wav" 0 0 0

i 1 0 3 1 ;mono file
i 1 + 4 2 ;stereo file
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*loscil3* et *GEN01*

## Crédits

La note au sujet de la différence mono/stéréo est due à Rasmus Ekman.

# loscil3

loscil3 — Lit un son échantillonné depuis une table avec interpolation cubique.

## Description

Lit un son échantillonné (mono ou stéréo) depuis une table, avec des boucles facultatives d'entretien et de relâchement, et interpolation cubique. Une version donnant la position exacte dans la table (phase) correspondant à l'échantillon en sortie est fournie comme opcode alternatif.

## Syntaxe

```
ar1 [,ar2] loscil3 xamp, kcps, ifn [, ibas] [, imod1] [, ibeg1] [, iend1] \  
    [, imod2] [, ibeg2] [, iend2]  
  
aph, ar1 [,ar2] loscil3phs xamp, kcps, ifn [, ibas] [, imod1] [, ibeg1] [, iend1] \  
    [, imod2] [, ibeg2] [, iend2]
```

## Initialisation

*ifn* -- numéro de table de fonction, contenant typiquement un son échantillonné avec des points de boucle précisés, remplie au moyen de *GEN01*. Le fichier source peut être mono ou stéréo.

*ibas* (facultatif) -- fréquence de base en Hz du son enregistré. Elle remplace éventuellement la fréquence donnée dans le fichier audio, mais devient nécessaire si le fichier n'en contient pas. La valeur par défaut est 261,626 Hz, c-à-d le do médian. (Nouveau dans Csound 4.03). Si la valeur est inconnue ou absente il faut utiliser 1 ici et dans *kcps*.

*imod1*, *imod2* (facultatif, -1 par défaut) -- modes d'interprétation des boucles d'entretien et de relâchement. Une valeur de 1 signifie une boucle normale, 2 signifie une boucle à l'endroit et à l'envers, 0 signifie pas de boucle. La valeur par défaut (-1) s'en remet au mode et aux points de boucle définis dans le fichier source. Il faut s'assurer de choisir un mode approprié si le fichier ne contient pas cette information.

*ibeg1*, *iend1*, *ibeg2*, *iend2* (facultatifs, dépendants de *mod1*, *mod2*) -- début et fin des boucles d'entretien et de relâchement. Ils sont mesurés en *trames d'échantillon* depuis le début du fichier, et auront ainsi la même valeur que le son soit mono ou stéréo. Si aucun point de boucle n'est spécifié et qu'un mode de boucle est donné (*imod1*, *imod2*, le fichier sera lu en boucle sur toute sa longueur.

## Exécution

*aph* -- la position normalisée dans la table correspondant à l'échantillon en sortie (seulement pour *loscilphs3*).

*ar1*, *ar2* -- la sortie de taux audio. Il n'y a que *ar1* pour une sortie mono, alors qu'il y a *ar1* et *ar2* pour une sortie stéréo.

*xamp* -- l'amplitude du signal de sortie.

*kcps* -- la fréquence du signal de sortie en Hz.

*loscil3* est identique à *loscil* sauf qu'il utilise l'interpolation cubique. Nouveau dans la version 3.50 de Csound.





## Note pour les utilisateurs de Windows

Les utilisateurs de Windows utilisent normalement l'antislash, « \ », lorsqu'ils écrivent les chemins de leurs fichiers. Par exemple, un utilisateur de Windows pourra utiliser le chemin « c:\music\samples\loop001.wav ». Ceci pose problème car les l'antislash est normalement utilisé pour spécifier des caractères spéciaux.

Pour écrire correctement ce chemin dans Csound, on peut :

- Soit *utiliser le slash* : c:/music/samples/loop001.wav
- Soit *utiliser le caractère spécial d'antislash*, « \\ » : c:\\music\\samples\\loop001.wav



## Note

Voici *loscil3* en mono :

```
a1 loscil3 10000, 1, 1, 1, 1
```

... et *loscil3* en stéréo :

```
a1, a2 loscil3 10000, 1, 1, 1, 1
```

## Exemples

Voici un exemple de l'opcode *loscil3*. Il utilise les fichiers *loscil3.csd* [examples/loscil3.csd], *mary.wav* [examples/mary.wav] et *kickroll.wav* [examples/kickroll.wav].

### Exemple 508. Exemple de l'opcode *loscil3*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o loscil3.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ichnls = ftchnls(p4)
print ichnls

if (ichnls == 1) then
  asigL loscil3 .8, 1, p4, 1
  asigR = asigL
elseif (ichnls == 2) then
  asigL, asigR loscil3 .8, 1, p4, 1
```

```

; safety precaution if not mono or stereo
else
    asigL = 0
    asigR = 0
endif
    outs asigL, asigR

endin
</CsInstruments>
<CsScore>
f 1 0 0 1 "mary.wav" 0 0 0
f 2 0 0 1 "kickroll.wav" 0 0 0

i 1 0 3 1 ;mono file
i 1 + 2 2 ;stereo file
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*loscil* et *GEN01*

## Crédits

La note au sujet de la différence mono/stéréo est due à Rasmus Ekman.

# loscilx

loscilx — Lit un son échantillonné multi-canaux depuis une table.

## Description

Lit un son échantillonné (jusqu'à 16 canaux) depuis une table, avec avec bouclage d'entretien et de relâchement facultatif.

## Syntaxe

```
ar1 [, ar2, ar3, ar4, ar5, ar6, ar7, ar8, ar9, ar10, ar11, ar12, ar13, ar14, \
    ar15, ar16] loscilx xamp, kcps, ifn \
    [, iwsiz, ibas, istr, imod, ibeg, iend]

ar[] loscilx xamp, kcps, ifn \
    [, iwsiz, ibas, istr, imod, ibeg, iend]
```

## Initialisation

*ifn* -- numéro de table de fonction, désignant typiquement un segment de son échantillonné avec prescription de points de bouclage au moyen de *GENOI*. Le fichier source peut avoir jusqu'à 16 canaux.

*iwsiz* (facultatif) -- taille de la fenêtre utilisée dans l'interpolation. *iwsiz* (facultatif, zéro par défaut) -- taille de la fenêtre d'interpolation en échantillons. Peut prendre une des valeurs suivantes :

- 1 : arrondi à l'échantillon le plus proche (pas d'interpolation, pour *kpitch*=1)
- 2 : interpolation linéaire
- 4 : interpolation cubique
- >=8 : *iwsiz* interpolation sinc par point avec anti-repliement (lente)

Zéro ou des valeurs négatives sélectionnent l'action par défaut qui est l'interpolation cubique.

*ibas* (facultatif) -- fréquence de base en *Hz* du son enregistré. Permet de redéfinir la fréquence donnée dans le fichier audio, mais est requis si le fichier n'en contient pas. La valeur par défaut est 261.626 Hz, le do médian. (Nouveau dans Csound 4.03). Si la valeur est inconnue ou absente, utiliser 1 ici et dans *kcps*.

*istr* (facultatif, 0 par défaut) -- Trame du début de la lecture des données. Si ce n'est pas un nombre entier, les données sont interpolées (voir *iwsiz*).

*imod* (facultatif, -1 par défaut) -- mode de jeu des boucles d'entretien et de relâchement. 1 signifie un bouclage normal, 2 un bouclage à l'endroit et à l'envers, 0 pas de boucle. La valeur par défaut (-1) choisit le mode et les points de bouclage donnés dans le fichier source. Il ne faut pas oublier de choisir un mode approprié si le fichier ne contient pas cette information.

*ibeg*, *iend* (facultatif, dépendant d'*imod*) -- points de début et de fin des boucles d'entretien et de relâchement. Ils sont mesurés en *trames d'échantillon* depuis le début du fichier. Si aucun point de boucle n'est spécifié alors qu'un mode de bouclage (*imod*) est donné, le fichier sera bouclé sur toute sa longueur.

## Exécution

*ar1*, *ar2*, ... *ar[]* -- la sortie au taux audio. Le nombre de sorties doit correspondre au nombres de canaux dans le fichier échantillonné.

*xamp* -- l'amplitude du signal de sortie.

*kcps* -- le facteur de lecture du fichier. Par exemple, une valeur de 1 ne provoque pas de transposition, 1.5 transpose d'une quinte vers le haut et 2 d'une octave.

*loscilx* échantillonne la ftable audio au taux déterminé par *kcps*, puis multiplie le résultat par *xamp*. L'incrément d'échantillonnage pour *kcps* dépend de la fréquence de la note de base de la table *ibas*, et est automatiquement ajusté si la valeur du *sr* de l'orchestre diffère de celle utilisée lors de l'enregistrement de la source. Dans cette unité la ftable est toujours échantillonnée avec interpolation.

Si l'échantillonnage atteint le point final de la *boucle d'entretien* et que le bouclage est actif, le point d'échantillonnage est modifié et *loscil* continue sa lecture depuis ce segment de boucle. Si l'instrument reçoit un signal *turnoff* (depuis la partition ou depuis un évènement MIDI noteoff), le prochain point final d'entretien est ignoré et l'échantillonnage continue vers le point final de la *boucle de relâchement* ou vers le dernier échantillon (des valeurs nulles).

Si l'on veut boucler sur tout le fichier, spécifier un mode de bouclage dans *imod* et n'entrer aucune valeur dans *ibeg* et dans *iend*.

## Exemples

Voici un exemple de l'opcode *loscilx*. Il utilise les fichiers *loscil.csd* [examples/loscil.csd] et *kickroll.wav* [examples/kickroll.wav].

### Exemple 509. Exemple de l'opcode *loscilx*.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o loscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 44100
ksmps = 1
nchnls = 2
0dbfs = 1.0

gS_loop = "kickroll.wav"
gisr filesr gS_loop
gilength filelen gS_loop
gibeats = 16
gi_sampleleft ftgen 1, 0, 0, 1, gS_loop, 0, 4, 0

instr 1
  idur = p3
  iamp = p4
  ibeat = p5
  itune = p6
  ipos = ibeat / gibeats * gilength * gisr

  aenv linseg iamp, idur - 0.01, iamp, 0.01, 0
  a1, a2 loscilx aenv, itune, 1, 0, 1, ipos, 0
```

```

    outs a1, a2
endin

</CsInstruments>
<CsScore>

t 0 170

; Measure 1
i 1 0.0 0.5 0.707 0 1
i 1 1.0 0.5 0.707 1 1
i 1 2.5 0.5 0.707 0 1
i 1 3.0 0.5 0.707 1 1

; Measure 2
i 1 4.0 0.5 0.707 0 1
i 1 5.0 0.5 0.707 1 1
i 1 6.5 0.5 0.707 0 1
i 1 7.0 0.5 0.707 1 1

; Measure 3
i 1 8.0 0.5 0.707 0 1
i 1 9.0 0.5 0.707 1 1
i 1 10.5 0.5 0.707 0 1
i 1 11.0 0.5 0.707 1 1

; Measure 4
i 1 12.0 0.5 0.707 0 1
i 1 13.0 0.5 0.707 1 1
i 1 14.5 0.5 0.707 0 1
i 1 15.0 0.5 0.707 1 1

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*loscil*

## Crédits

Ecrit par Istvan Varga.

2006

Nouveau dans Csound 5.03

Nouvelle version avec tableau dans Csound 6.13 (Février 2019)

# lowpass2

lowpass2 — Un filtre passe-bas résonant.

## Description

Implémentation d'un filtre passe-bas résonant du second ordre.

## Syntaxe

```
ares lowpass2 asig, kcf, kq [, iskip]
```

## Initialisation

*iskip* -- disposition initiale de la mémoire interne. Une valeur de zéro efface la mémoire ; une valeur différente de zéro permet de maintenir l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal d'entrée à filtrer

*kcf* -- fréquence de coupure ou de résonance du filtre, mesurée en Hz

*kq* -- Q du filtre, défini, pour les filtres passe-bande, comme le rapport (largeur de bande)/(fréquence de coupure). *kq* doit être compris entre 1 et 500.

*lowpass2* est un filtre RII passe-bas du second ordre, avec contrôle au taux-k de la fréquence de coupure (*kcf*) et de Q (*kq*). Lorsque *kq* augmente un pic de résonance se forme autour de la fréquence de coupure, transformant la réponse du filtre passe-bas en une réponse semblable à celle d'un filtre passe-bande, mais avec plus d'énergie dans les basses fréquences. Cela correspond à un accroissement de la magnitude et de la "raideur" du pic de résonance. On peut avoir besoin d'une fonction comme *balance* pour les grandes valeurs de *kq*. En pratique, cela permet la simulation des filtres contrôlés en tension des synthétiseurs analogiques, ou bien la création d'une hauteur d'amplitude constante lorsque l'on filtre un bruit blanc.

## Exemples

Voici un exemple de l'opcode *lowpass2*. Il utilise le fichier *lowpass2.csd* [exemples/lowpass2.csd].

### Exemple 510. Exemple de l'opcoce lowpass2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lowpass2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

/* Written by Sean Costello */
; Orchestra file for resonant filter sweep of a sawtooth-like waveform.
sr = 44100
kr = 2205
ksmps = 20
nchnls = 1

instr 1

idur    =      p3
ifreq   =      p4
iamp    =      p5 * .5
iharms  =      (sr*.4) / ifreq

; Sawtooth-like waveform
asig    gbuzz 1, ifreq, iharms, 1, .9, 1

; Envelope to control filter cutoff
kfreq   linseg 1, idur * 0.5, 5000, idur * 0.5, 1

afilt    lowpass2 asig,kfreq, 30

; Simple amplitude envelope
kenv     linseg 0, .1, iamp, idur -.2, iamp, .1, 0
out      afilt * kenv

endin

</CsInstruments>
<CsScore>

/* Written by Sean Costello */
f1 0 8192 9 1 1 .25

i1 0 5 100 1000
i1 5 5 200 1000
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Sean Costello  
 Seattle, Washington  
 Août 1999

Nouveau dans la version 4.0 de Csound.

# lowres

lowres — Un autre filtre passe-bas à résonance.

## Description

*lowres* est un filtre passe-bas à résonance.

## Syntaxe

```
ares lowres asig, xcutoff, xresonance [, iskip]
```

## Initialisation

*iskip* -- disposition initiale de la mémoire interne. Une valeur de zéro efface la mémoire ; une valeur différente de zéro permet de maintenir l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal d'entrée

*xcutoff* -- fréquence de coupure du filtre

*xresonance* -- quantité de résonance

*lowres* est un filtre passe-bas à résonance dérivé d'un orchestre écrit par Hans Mikelson. Cette implémentation est bien plus rapide que celle écrite dans le langage de Csound, et elle permet d'avoir un *kr* inférieur à *sr*. *xcutoff* n'étant pas en Hz et *xresonance* pas en dB, il faut expérimenter pour obtenir les meilleurs résultats.

## Exemples

Voici un exemple de l'opcode *lowres*. Il utilise le fichier *lowres.csd* [examples/lowres.csd].

### Exemple 511. Exemple de l'opcode *lowres*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o lowres.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
```



```
instr 1

kres = p4
asig vco .2, 220, 1 ;sawtooth

kcut line 1000, p3, 10 ;note: kcut is not in Hz
as lowres asig, kcut, kres ;note: kres is not in dB
aout balance as, asig ;avoid very loud sounds
outs aout, aout

endin
</CsInstruments>
<CsScore>
; a sine
f 1 0 16384 10 1

i 1 0 4 3
i 1 + 4 30
i 1 + 4 60
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*lowresx*

## Crédits

Auteur : Gabriel Maldonado (adapté par John ffitich)  
 Italie

Nouveau dans la version 3.49 de Csound.

# lowresx

lowresx — Simule des couches de filtres passe-bas à résonance connectés en série.

## Description

*lowresx* est équivalent à plusieurs étages de *lowres* connectés en série, avec les mêmes arguments.

## Syntaxe

```
ares lowresx asig, xcutoff, xresonance [, inumlayer] [, iskip]
```

## Initialisation

*inumlayer* -- nombre d'éléments dans une aggrégation *lowresx*. La valeur par défaut est 4. Il n'y a pas de maximum.

*iskip* -- disposition initiale de la mémoire interne. Une valeur de zéro efface la mémoire ; une valeur différente de zéro permet de maintenir l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal d'entrée

*xcutoff* -- fréquence de coupure du filtre

*xresonance* -- quantité de résonance

*lowresx* est équivalent à plusieurs étages de *lowres* connectés en série, avec les mêmes arguments. Plus il y a de filtres dans l'aggrégation et plus la coupure est raide. C'est plus rapide que d'utiliser un plus grand nombre d'instances de *lowres* dans un orchestre de Csound parce que ne sont nécessaires qu'une seule initialisation et qu'un cycle *k* à la fois, et que la boucle audio est entièrement contenue dans la mémoire cache du processeur. Basé sur un orchestre par Hans Mikelson.

## Exemples

Voici un exemple de l'opcode *lowresx*. Il utilise le fichier *lowresx.csd* [examples/lowresx.csd].

### Exemple 512. Exemple de l'opcode *lowresx*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o lowresx.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kres = p4
inumlayer = 4

kenv linseg 0, p3*.1, 1, p3*.8, 1, p3*.1, 0 ;envelope
asig vco .3 * kenv, 220, 1 ;sawtooth
kcut line 30, p3, 1000 ;note: kcut is not in Hz
alx lowresx asig, kcut, kres, inumlayer ;note: kres is not in dB
aout balance alx, asig ;avoid very loud sounds
outs aout, aout

endin
</CsInstruments>
<CsScore>
;sine wave
f 1 0 16384 10 1

i 1 0 5 1
i 1 + 5 3
i 1 + 5 20
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*lowres*

## Crédits

Auteur : Gabriel Maldonado (adapté par John ffitich)

Italie

Nouveau dans la version 3.49 de Csound.

Paramètres de taux audio introduits dans la version 6.02

Octobre 2013.

# lpf18

lpf18 — Un filtre passe-bas résonant glissant à 3 pôles.

## Description

Implémentation d'un filtre passe-bas résonant glissant à 3 pôles.

## Syntaxe

```
ares lpf18 asig, xfco, xres, xdist [, iskip]
```

## Initialisation

*iskip* (facultatif, à par défaut) -- S'il est présent et différent de zéro, l'initialisation est ignorée.

## Exécution

*xfco* -- fréquence de coupure du filtre en Hz. Doit être comprise entre 0 et *sr*/2.

*xres* -- quantité de résonance. Il y a des auto-oscillations lorsque *kres* est proche de 1. Doit être habituellement compris entre 0 et 1, mais des valeurs légèrement supérieures à 1 sont possibles pour obtenir des oscillations plus soutenues et un effet de « saturation ».

*xdist* -- quantité de distorsion. *kdist* = 0 donne une sortie propre. *kdist* > 0 ajoute une distorsion de type *tanh()* contrôlée par les paramètres du filtre, de façon à ce qu'une faible fréquence de coupure et qu'une résonance importante augmentent le taux de distorsion. Il est conseillé d'expérimenter.

*lpf18* est une simulation numérique d'un filtre passe-bas à 3 pôles (18 dB/oct) capable d'auto-oscillations avec une unité de distorsion intégrée. C'est vraiment une version 3 pôles de *moogvcf*, révisée, recalibrée et comportant quelques améliorations. Les tables de réglage et de rétroaction n'utilisent pas plus de 6 additions et 6 multiplications par cycle de contrôle. L'unité de distorsion est basée sur une fonction *tanh* modifiée, pilotée par les contrôles du filtre.



### Note

Avant la version 6.04, ce filtre nécessitait un signal d'entrée normalisé à un.

## Exemples

Voici un exemple de l'opcode lpf18. Il utilise le fichier *lpf18.csd* [examples/lpf18.csd].

### Exemple 513. Exemple de l'opcode lpf18.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in    No messages
```

```
-odac          -iadc      -d      ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lpf18.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Generate a sine waveform.
; Note that its amplitude (kamp) ranges from 0 to 1.
kamp init 1
kcps init 440
knh init 3
ifn = 1
asine buzz kamp, kcps, knh, ifn

; Filter the sine waveform.
; Vary the cutoff frequency (kfco) from 300 to 3,000 Hz.
kfco line 300, p3, 3000
kres init 0.8
kdist = p4
ivol = p5
aout lpf18 asine, kfco, kres, kdist

out aout * ivol
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; different distortion and volumes to compensate
i 1 0 4 0.2 30000
i 1 4.5 4 0.9 27000
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Josep M Comajuncosas  
 Espagne  
 Décembre 2000

Exemple écrit par Kevin Conder avec l'aide de Iain Duncan. Merci à Iain pour son aide sur cet exemple.

Nouveau dans la version 4.10 de Csound.

Paramètres de taux audio introduits dans la version 6.02

Octobre 2013.

# lpfreson

lpfreson — Resynthétise un signal à partir des données passées en interne par un lpread précédent, en appliquant un décalage de formant.

## Description

Resynthétise un signal à partir des données passées en interne par un *lpread* précédent, en appliquant un décalage de formant.

## Syntaxe

```
ares lpfreson asig, kfrqratio
```

## Exécution

*asig* -- une fonction audio pour piloter la resynthèse.

*kfrqratio* -- rapport de fréquence. Doit être supérieur à 0.

*lpreson* reçoit en interne des valeurs produites par un *lpread* conducteur. *lpread* reçoit ses valeurs du fichier de contrôle en fonction de la valeur d'entrée *ktimpnt* (en secondes). Si *ktimpnt* évolue au taux de l'analyse, il en résulte une synthèse à déroulement temporel normal ; si l'évolution est plus rapide, plus lente ou à taux variable, le déroulement temporel de la synthèse est déformé. A chaque période-k, *lpread* fait une interpolation entre trames adjacentes pour déterminer plus précisément les valeurs du paramètre présenté en sortie et le réglage des coefficients du filtre (passés en interne à un *lpreson*) qui le suit.

Le signal d'erreur *kerr* (entre 0 et 1) provenant de l'analyse prédictive reflète la nature déterministe/aléatoire de la source analysée. Il paraîtra bas pour un matériau tonal (périodique) et plus important pour un matériau bruiteux. La transition de la parole voisée à la parole non-voisée, par exemple, produit une valeur du signal d'erreur d'environ 0.3. Pendant la synthèse, la valeur du signal d'erreur peut être utilisée pour déterminer la nature de la fonction pilotant *lpreson* : par exemple en arbitrant entre entrée tonale et non-tonale, ou même en déterminant un mélange des deux. Normalement, dans la resynthèse de la parole, l'entrée tonale de *lpreson* est un signal périodique à large bande ou un train d'impulsions dérivé d'une unité telle que *buzz*, et la source non-tonale est habituellement dérivée de *rand*. Cependant, on peut utiliser n'importe quel signal audio comme fonction pilote, la seule exigence de l'analyse étant qu'il ait une réponse plate.

*lpfreson* est un *lpreson* dans lequel *kfrqratio* est le rapport de décalage de formant par rapport à l'original. Cela permet une synthèse dans laquelle l'objet source modifie sa taille acoustique apparente. *lpfreson* avec *kfrqratio* = 1 est équivalent à *lpreson*.

Généralement, *lpreson* fournit un moyen de contrôler l'évolution du contenu et de la forme spectrale d'un signal audio composite par le contenu spectral dynamique d'un autre signal. Il peut y avoir n'importe quel nombre de paires *lpread/lpreson* (ou *lpfreson*) dans un instrument ou dans un orchestre ; ils peuvent lire depuis le même ou depuis différents fichiers de contrôle indépendamment.

## Exemples

Voici un exemple de l'opcode *lpfreson*. Il utilise le fichier *lpfreson.csd* [examples/lpfreson.csd].

**Exemple 514. Exemple de l'opcode lpfreson.**

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o lpfreson.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
;do not use -a option when analyzing "fox.wav" with lpfreson,
;it needs a filter coefficient type of file
instr 1

ilen filelen "fox.wav" ; length of soundfile
prints "fox.wav = %f seconds\\n",ilen

ktime line 0, p3, p4
krmsr,krms0,kerr,kcps lpread ktime,"fox_nopoles.lpc"
krms0 = krms0*.00001 ; low volume
asig buzz krms0, kcps, int(sr/2/kcps), 1 ; max harmonics without aliasing
aout lpfreson asig, 1.2
asig clip aout, 0, 1 ; prevents distortion
outs asig, asig

endin
</CsInstruments>
<CsScore>
; sine
f1 0 4096 10 1

i 1 0 2.8 1 ; first words only
i 1 4 2.8 2.8 ; whole sentence
e
</CsScore>
</CsoundSynthesizer>

```

Le fichier audio « fox.wav » dure 2.8 secondes. Ainsi la sortie contiendra une ligne comme celle-ci :

```
fox.wav = 2.756667 seconds
```

**Voir aussi**

*lpread, lpreson*

# lphasor

lphasor — Génère un indice de table pour la lecture d'échantillons.

## Description

Cet opcode peut être utilisé pour générer un indice de table pour la lecture d'échantillons (par exemple avec `tablexkt`).

## Syntaxe

```
ares lphasor xtrns [, ilps] [, ilpe] [, imode] [, istrtr] [, istor]
```

## Initialisation

*ilps* -- début de la boucle.

*ilpe* -- fin de la boucle (doit être supérieur à *ilps* pour que la boucle soit possible). La valeur par défaut de *ilps* et de *ilpe* est zéro.

*imode* (facultatif : 0 par défaut) -- mode de boucle. Les valeurs permises sont :

- 0 : pas de boucle
- 1 : boucle à l'endroit
- 2 : boucle à l'envers
- 3 : boucle à l'endroit et à l'envers

*istrtr* (facultatif : 0 par défaut) -- La valeur de sortie initiale (phase). Elle doit être inférieure à *ilpe* si la boucle est active, mais elle peut être supérieure à *ilps* (c-à-d que l'on peut démarrer la lecture au milieu de la boucle).

*istor* (facultatif : 0 par défaut) -- s'il a une valeur différente de zéro l'initialisation est ignorée.

## Exécution

*ares* -- un indice brut de table en échantillons (même unité pour les points de boucle). Peut être utilisé comme indice de table avec les opcodes de table.

*xtrns* -- facteur de transposition, exprimé comme un rapport de pointeur de lecture. *ares* est incrémenté de cette valeur, et répète les valeurs comprises entre les points de boucle. Par exemple, 1.5 signifie une quinte ascendante, 0.75 signifie une quarte descendante. Il ne peut pas être négatif.

## Exemples

Voici un exemple de l'opcode `lphasor`. Il utilise le fichier *lphasor.csd* [exemples/lphasor.csd].

### Exemple 515. Exemple de l'opcode `lphasor`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.



```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o lphashor.wav -W ;; for file output any platform
</CsOptions>

<CsInstruments>
; Example by Jonathan Murphy Dec 2006

sr      = 44100
ksmps   = 10
nchnls  = 1

instr 1

ifn      = 1 ; table number
ilen     = nsamp(ifn) ; return actual number of samples in table
itrns    = 1 ; no transposition
ilps     = 0 ; loop starts at index 0
ilpe     = ilen ; ends at value returned by nsamp above
imode    = 3 ; loop forwards & backwards
istrt    = 10000 ; commence playback at index 10000 samples
; lphasor provides index into f1
alphs    lphasor itrns, ilps, ilpe, imode, istrt
atab     tablei alphs, ifn
; amplify signal
atab     = atab * 10000

out      atab

endin

</CsInstruments>

<CsScore>
f 1 0 262144 1 "beats.wav" 0 4 1
i1 0 60
e
</CsScore>

</CsoundSynthesizer>

```

## Crédits

Auteur : Istvan Varga  
 Janvier 2002  
 Exemple écrit par Jonathan Murphy

Nouveau dans la version 4.18

Mise à jour en avril 2002 et en novembre 2002 par Istvan Varga

# lpinterp

lpslot, lpinterp — Calcule un nouvel ensemble de pôles à partir de l'interpolation entre deux analyses.

## Description

Calcule un nouvel ensemble de pôles à partir de l'interpolation entre deux analyses.

## Syntaxe

```
lpinterp islot1, islot2, kmix
```

## Initialisation

*islot1* -- slot dans lequel la première analyse est stockée

*islot2* -- slot dans lequel la seconde analyse est stockée

*kmix* -- valeur du mélange entre les deux analyses. Doit être comprise entre 0 et 1. 0 pour l'analyse 1 seulement. 1 pour l'analyse 2 seulement. Toute valeur intermédiaire produit une interpolation entre les filtres.

*lpinterp* calcule un nouvel ensemble de pôles à partir de l'interpolation entre deux analyses. Les pôles sont stockés dans le *lpslot* courant pour être utilisés par l'opcode *lpreson* suivant.

## Exemples

Voici un orchestre typique utilisant les opcodes :

```
ipower init 50000 ; Define sound generator
ifreq  init 440
asrc  buzz ipower, ifreq, 10, 1

ktime  line 0, p3, p3          ; Define time lin
      lpslot 0                ; Read square data poles
krmsr, krms0, kerr, kcps lpread ktime,"square.pol"
      lpslot 1                ; Read triangle data poles
krmsr, krms0, kerr, kcps lpread ktime,"triangle.pol"
kmix   line 0, p3, 1           ; Compute result of mixing
      lpinterp 0, 1, kmix     ; and balance power
ares  lpreson asrc
aout  balance ares, asrc
      out aout
```

## Voir aussi

*lpslot*

## Crédits

Auteur : Gabriel Maldonado

# lposcil

lposcil — Lit un son échantillonné depuis une table avec boucle et haute précision.

## Description

Lit un son échantillonné (mono ou stéréo) depuis une table, avec boucle et haute précision.

## Syntaxe

```
ares lposcil kamp, kfregratio, kloop, kend, ifn [, iphs]
```

## Initialisation

*ifn* -- numéro de la table de fonction

## Exécution

*kamp* -- amplitude

*kfregratio* -- facteur de multiplication de la fréquence de la table (par exemple : 1 = fréquence originale, 1.5 = une quinte ascendante, 0.5 = une octave descendante)

*kloop* -- début de la boucle (en échantillons)

*kend* -- fin de la boucle (en échantillons)

*lposcil* (looping precise oscillator) permet de faire varier au taux-k le début et la fin d'un son échantillonné contenu dans une table (*GEN01*). Peut être utile pour lire une boucle d'échantillons depuis une table d'onde, avec une vitesse de répétition variant durant l'exécution.

## Exemples

Voici un exemple de l'opcode *lposcil*. Il utilise le fichier *lposcil.csd* [examples/lposcil.csd].

### Exemple 516. Exemple de l'opcode *lposcil*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o lposcil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
```

```
Odbfs = 1

instr 1

kcps = 1.5 ; a fifth up
kloop = 0 ; loop start time in samples
kend = 45000 ; loop end time in samples

asig lposcil 1, kcps, kloop, kend, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
; Its table size is deferred,
; and format taken from the soundfile header.
f 1 0 0 1 "beats.wav" 0 0 0

; Play Instrument #1 for 6 seconds.
; This will loop the drum pattern several times.
i 1 0 6

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*lposcil3, lposcila, lposcilsa, lposcilsa2*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.52 de Csound

# lposcil3

lposcil3 — Lit un son échantillonné depuis une table en haute précision avec interpolation cubique.

## Description

Lit un son échantillonné (mono ou stéréo) depuis une table, avec boucle et haute précision. *lposcil3* utilise l'interpolation cubique.

## Syntaxe

```
ares lposcil3 kamp, kfregratio, kloop, kend, ifn [, iphs]
```

## Initialisation

*ifn* -- numéro de la table de fonction

## Exécution

*kamp* -- amplitude

*kfregratio* -- facteur de multiplication de la fréquence de la table (par exemple : 1 = fréquence originale, 1.5 = une quinte ascendante, 0.5 = une octave descendante)

*kloop* -- début de la boucle (en échantillons)

*kend* -- fin de la boucle (en échantillons)

*lposcil3* (looping precise oscillator) permet de faire varier au taux-k le début et la fin d'un son échantillonné contenu dans une table (*GEN01*). Peut être utile pour lire une boucle d'échantillons depuis une table d'onde, avec une vitesse de répétition variant durant l'exécution.

## Exemples

Voici un exemple de l'opcode *lposcil3*. Il utilise le fichier *lposcil3.csd* [examples/lposcil3.csd].

### Exemple 517. Exemple de l'opcode lposcil3.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o lposcil3.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
```

```
nchnls = 2
0dbfs = 1

instr 1

kcps = 1.5 ; a fifth up
kloop = 0 ; loop start time (in samples)
kend line 45000, p3, 10000 ; vary loop end time (in samples)

asig lposcil3 1, kcps, kloop, kend, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
; Its table size is deferred,
; and format taken from the soundfile header.
f 1 0 0 1 "beats.wav" 0 0 0

; Play Instrument #1 for 6 seconds.
; This will loop the drum pattern several times.
i 1 0 6

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*lposcil*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.52 de Csound

# lposcila

*lposcila* — Lit un son échantillonné depuis une table avec boucle et haute précision.

## Description

*lposcila* lit un son échantillonné depuis une table avec boucle et haute précision.

## Syntaxe

```
ar lposcila aamp, kfregratio, kloop, kend, ift [,iphs]
```

## Initialisation

*ift* -- numéro de la table de fonction

*iphs* -- phase initiale (en échantillons)

## Exécution

*ar* -- signal de sortie

*aamp* -- amplitude

*kfregratio* -- facteur de multiplication de la fréquence de la table (par exemple : 1 = fréquence originale, 1.5 = une quinte ascendante, 0.5 = une octave descendante)

*kloop* -- début de la boucle (en échantillons)

*kend* -- fin de la boucle (en échantillons)

*lposcila* est semblable à *lposcil*, mais il a un argument d'amplitude de taux audio (au lieu du taux-k) pour permettre des transitoires d'enveloppe rapides.

## Exemples

Voici un exemple de l'opcode *lposcila*. Il utilise le fichier *lposcila.csd* [examples/lposcila.csd].

### Exemple 518. Exemple de l'opcode *lposcila*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o lposcila.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kcps = 1.3      ;a 3d up
kloop = 0       ;loop start time in samples
kend = 10000    ;loop end time in samples

aenv expsega 0.01, 0.1, 1, 0.1, 0.5, 0.5, 0.01 ;envelope with fast and short segment
asig lposcila aenv, kcps, kloop, kend, 1 ;use it for amplitude
outs asig, asig

endin
</CsInstruments>
<CsScore>
; Its table size is deferred,
; and format taken from the soundfile header.
f 1 0 0 1 "beats.wav" 0 0 0

; Play Instrument #1 for 6 seconds.
; This will loop the drum pattern several times.
i 1 0 2

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*lposcil, lposcilsa, lposcilsa2*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06



# lposcilsa

*lposcilsa* — Lit un son stéréo échantillonné depuis une table avec boucle et haute précision.

## Description

*lposcilsa* lit un son stéréo échantillonné depuis une table avec boucle et haute précision.

## Syntaxe

```
ar1, ar2 lposcilsa aamp, kfregratio, kloop, kend, ift [,iphs]
```

## Initialisation

*ift* -- numéro de la table de fonction

*iphs* -- phase initiale (en échantillons)

## Exécution

*ar1, ar2* -- signal de sortie

*aamp* -- amplitude

*kfregratio* -- facteur de multiplication de la fréquence de la table (par exemple : 1 = fréquence originale, 1.5 = une quinte ascendante, 0.5 = une octave descendante)

*kloop* -- début de la boucle (en échantillons)

*kend* -- fin de la boucle (en échantillons)

*lposcilsa* est semblable à *lposcila*, mais il travaille avec des fichiers stéréo chargés par *GEN01*.

## Exemples

Voici un exemple de l'opcode *lposcilsa*. Il utilise le fichier *lposcilsa.csd* [examples/lposcilsa.csd].

### Exemple 519. Exemple de l'opcode *lposcilsa*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o lposcilsa.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
```

```

ksmps = 32
nchnls = 2
odbfs = 1

instr 1

kcps = 1.3      ;a 3th up
kloop = 0       ;loop start time in samples
kend = 45000    ;loop end time in samples

aenv expsega 0.01, 0.1, 1, 0.1, 0.5, 0.5, 0.01 ;envelope with fast and short segment
aL, aR lposcilsa aenv, kcps, kloop, kend, 1 ;use it for amplitude
outs aL, aR

endin
</CsInstruments>
<CsScore>
; table size of stereo file is deferred,
; and format taken from the soundfile header.
f 1 0 0 1 "kickroll.wav" 0 0 0

; Play Instrument #1 for 6 seconds.
; This will loop the drum pattern several times.
i 1 0 2

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*lposcil, lposcila, lposcilsa2*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# lposcilsa2

*lposcilsa2* — Lit un son stéréo échantillonné depuis une table avec boucle et haute précision.

## Description

*lposcilsa2* lit un son stéréo échantillonné depuis une table avec boucle et haute précision.

## Syntaxe

```
ar1, ar2 lposcilsa2 aamp, kfregratio, kloop, kend, ift [,iphs]
```

## Initialisation

*ift* -- numéro de la table de fonction

*iphs* -- phase initiale (en échantillons)

## Exécution

*ar1, ar2* -- signal de sortie

*aamp* -- amplitude

*kfregratio* -- facteur de multiplication de la fréquence de la table (par exemple : 1 = fréquence originale, 2 = une octave ascendante). Seule les nombres entiers sont permis.

*kloop* -- début de la boucle (en échantillons)

*kend* -- fin de la boucle (en échantillons)

*lposcilsa2* est semblable à *lposcilsa*, mais sans interpolation et il ne travaille qu'avec des valeurs entières de *kfregratio*. Beaucoup plus rapide que *lposcilsa*, il est prévu pour fonctionner principalement avec *kfregratio* = 1, étant dans ce cas un substitut rapide de *soundin*, car le fichier son doit être chargé entièrement en mémoire.

## Exemples

Voici un exemple de l'opcode *lposcilsa2*. Il utilise le fichier *lposcilsa2.csd* [examples/lposcilsa2.csd].

### Exemple 520. Exemple de l'opcode *lposcilsa2*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o lposcilsa2.wav -W ;; for file output any platform
```

```
</CsOptions>
<CsInstruments>

  sr = 44100
  ksmpr = 32
  nchnls = 2
  odbfs = 1

  instr 1

  kcps = 2      ;only integers are allowed
  kloop = 0     ;loop start time in samples
  kend = 45000  ;loop end time in samples

  aenv expsega 0.01, 0.1, 1, 0.1, 0.5, 0.5, 0.01 ;envelope with fast and short segment
  aL, aR lposcilsa2 aenv, kcps, kloop, kend, 1 ;use it for amplitude
    outs aL, aR

  endin
</CsInstruments>
<CsScore>
  ; Its table size is deferred,
  ; and format taken from the soundfile header.
  f 1 0 0 1 "kickroll.wav" 0 0 0

  ; Play Instrument #1 for 6 seconds.
  ; This will loop the drum pattern several times.
  i 1 0 2

e
</CsScore>
</CsSoundSynthesizer>
```

## Voir aussi

*lposcil, lposcila, lposcilsa*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# lpread

lpread — Lit un fichier de contrôle contenant des trames d'information ordonnées dans le temps.

## Description

Lit un fichier de contrôle contenant des trames d'information ordonnées dans le temps.

## Syntaxe

```
krmsr, krms0, kerr, kcps lpread ktmpnt, ifilcod [, inpoles] [, ifrmrate]
```

## Initialisation

*ifilcod* -- entier ou chaîne de caractères dénotant un fichier de contrôle (coefficients de réflexion et quatre valeurs de paramètre) provenant de l'analyse spectrale par prédiction linéaire à *n* pôles d'une source audio. Un entier dénote le suffixe d'un fichier *lp.m* ; une chaîne de caractères (entre guillemets) donne un nom de fichier, optionnellement un nom de chemin complet. Si ce n'est pas un nom de chemin complet, le fichier est d'abord cherché dans le répertoire courant, ensuite dans celui donné par la variable d'environnement *SADIR* (si elle est définie). L'utilisation de la mémoire dépend de la taille du fichier, qui est maintenu entièrement dans la mémoire durant les calculs mais reste partagé par des appels multiples (voir aussi *adsyn* et *pvoc*).

*inpoles* (facultatif, 0 par défaut) -- nombre de pôles dans l'analyse par prédiction linéaire. Nécessaire seulement si le fichier de contrôle n'a pas d'en-tête ; ignoré si un en-tête est détecté.

*ifrmrate* (facultatif, 0 par défaut) -- taux de trames par seconde dans l'analyse par prédiction linéaire. Nécessaire seulement si le fichier de contrôle n'a pas d'en-tête ; ignoré si un en-tête est détecté.

## Exécution

*lpread* accède à un fichier de contrôle constitué de trames d'information ordonnées dans le temps, chacune d'entre elles contenant les coefficients d'un filtre à *n* pôles, provenant de l'analyse par prédiction linéaire d'un signal source à intervalles fixes (par exemple 1/100 de seconde), plus quatre valeurs de paramètres :

*krmsr* -- moyenne quadratique du résidu de l'analyse

*krms0* -- moyenne quadratique du signal original

*kerr* -- le signal d'erreur normalisé

*kcps* -- hauteur en Hz

*ktmpnt* -- l'écoulement du temps en secondes dans le fichier d'analyse. *ktmpnt* doit toujours être positif, mais il peut avancer ou reculer, rester stationnaire ou être discontinu, comme pointeur dans le fichier d'analyse.

*lpread* reçoit ses valeurs du fichier de contrôle en fonction de la valeur d'entrée *ktmpnt* (en secondes). Si *ktmpnt* évolue au taux de l'analyse, il en résulte une synthèse à déroulement temporel normal ; si l'évolution est plus rapide, plus lente ou à taux variable, le déroulement temporel de la synthèse est déformé. A chaque période-*k*, *lpread* fait une interpolation entre trames adjacentes pour déterminer plus précisément

les valeurs des paramètres (présentés en sortie) et le réglage des coefficients du filtre (passés en interne à un *lpreson*) qui le suit.

Le signal d'erreur *kerr* (entre 0 et 1) provenant de l'analyse prédictive reflète la nature déterministe/aléatoire de la source analysée. Il paraîtra bas pour un matériau tonal (périodique) et plus important pour un matériau bruiteux. La transition de la parole voisée à la parole non-voisée, par exemple, produit une valeur du signal d'erreur d'environ 0.3. Pendant la synthèse, la valeur du signal d'erreur peut être utilisée pour déterminer la nature de la fonction pilotant *lpreson* : par exemple en arbitrant entre entrée tonale et non-tonale, ou même en déterminant un mélange des deux. Normalement, dans la resynthèse de la parole, l'entrée tonale de *lpreson* est un signal périodique à large bande ou un train d'impulsions dérivé d'une unité telle que *buzz*, et la source non-tonale est habituellement dérivée de *rand*. Cependant, on peut utiliser n'importe quel signal audio comme fonction pilote, la seule exigence de l'analyse étant qu'il ait une réponse plate.

*lpfreson* est un *lpreson* dont les formants sont décalés. Son argument *kfrqratio* est le rapport de la position des formants décalés par rapport à l'original. Cela permet une synthèse dans laquelle l'objet source modifie sa taille acoustique apparente. *lpfreson* avec *kfrqratio* = 1 est équivalent à *lpreson*.

Généralement, *lpreson* fournit un moyen de contrôler l'évolution du contenu et de la forme spectrale d'un signal audio composite par le contenu spectral dynamique d'un autre signal. Il peut y avoir n'importe quel nombre de paires *lpread/lpreson* (ou *lpfreson*) dans un instrument ou dans un orchestre ; ils peuvent lire depuis le même ou depuis différents fichiers de contrôle indépendamment.

## Exemples

Voici un exemple de l'opcode *lpread*. Il utilise le fichier *lpread.csd* [exemples/*lpread.csd*].

### Exemple 521. Exemple de l'opcode *lpread*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o lpread.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
; works with or without -a option when analyzing "fox.wav" from the manual
ilen filelen "fox.wav" ; length of soundfile
prints "fox.wav = %f seconds\\n",ilen

ktime line 0, p3, p4
krmsr,krms0,kerr,kcps lpread ktime,"fox_poles.lpc"
krms0 = krms0*.00007 ; low volume
aout buzz krms0, kcps, 15, 1
krmsr = krmsr*.0001 ; low volume
asig rand krmsr
outs (aout*2)+asig, (aout*2)+asig ; mix buzz and rand
```

```
    endin
  </CsInstruments>
  <CsScore>
    ; sine
    f1 0 4096 10 1

    i 1 0 2.8 1 ; first words only
    i 1 4 2.8 2.8 ; whole sentence
    e
  </CsScore>
</CsoundSynthesizer>
```

Le fichier audio « fox.wav » dure 2.8 secondes. Ainsi la sortie contiendra une ligne comme celle-ci :

```
fox.wav = 2.756667 seconds
```

## Voir aussi

*lpfreson, lpreson, LPANAL*

# lpreson

lpreson — Resynthétise un signal à partir des données passées en interne par un lpread précédent.

## Description

Resynthétise un signal à partir des données passées en interne par un *lpread* précédent.

## Syntaxe

```
ares lpreson asig
```

## Exécution

*asig* -- une fonction audio pour piloter la resynthèse.

*lpreson* reçoit en interne des valeurs produites par un *lpread* conducteur. *lpread* reçoit ses valeurs du fichier de contrôle en fonction de la valeur d'entrée *ktimpnt* (en secondes). Si *ktimpnt* évolue au taux de l'analyse, il en résulte une synthèse à déroulement temporel normal ; si l'évolution est plus rapide, plus lente ou à taux variable, le déroulement temporel de la synthèse est déformé. A chaque période-k, *lpread* fait une interpolation entre trames adjacentes pour déterminer plus précisément les valeurs du paramètre présenté en sortie et le réglage des coefficients du filtre (passés en interne à un *lpreson*) qui le suit.

Le signal d'erreur *kerr* (entre 0 et 1) provenant de l'analyse prédictive reflète la nature déterministe/aléatoire de la source analysée. Il paraîtra bas pour un matériau tonal (périodique) et plus important pour un matériau bruiteux. La transition de la parole voisée à la parole non-voisée, par exemple, produit une valeur du signal d'erreur d'environ 0.3. Pendant la synthèse, la valeur du signal d'erreur peut être utilisée pour déterminer la nature de la fonction pilotant *lpreson* : par exemple en arbitrant entre entrée tonale et non-tonale, ou même en déterminant un mélange des deux. Normalement, dans la resynthèse de la parole, l'entrée tonale de *lpreson* est un signal périodique à large bande ou un train d'impulsions dérivé d'une unité telle que *buzz*, et la source non-tonale est habituellement dérivée de *rand*. Cependant, on peut utiliser n'importe quel signal audio comme fonction pilote, la seule exigence de l'analyse étant qu'il ait une réponse plate.

*lpfreson* est un *lpreson* dont les formants sont décalés. Son argument *kfrqratio* est le rapport de décalage de formant par rapport à l'original. Cela permet une synthèse dans laquelle l'objet source modifie sa taille acoustique apparente. *lpfreson* avec *kfrqratio* = 1 est équivalent à *lpreson*.

Généralement, *lpreson* fournit un moyen de contrôler l'évolution du contenu et de la forme spectrale d'un signal audio composite par le contenu spectral dynamique d'un autre signal. Il peut y avoir n'importe quel nombre de paires *lpread/lpreson* (ou *lpfreson*) dans un instrument ou dans un orchestre ; ils peuvent lire depuis le même ou depuis différents fichiers de contrôle indépendamment.

## Exemples

Voici un exemple de l'opcode *lpreson*. Il utilise le fichier *lpreson.csd* [exemples/lpreson.csd].

### Exemple 522. Exemple de l'opcode lpreson.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.



```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o lpreson.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
; works with or without -a option when analyzing "fox.wav" from the manual
;both options sound a little different
instr 1

ilen filelen "fox.wav" ; length of soundfile
prints "fox.wav = %f seconds\\n",ilen

ktime line 0, p3, p4
krmsr,krms0,kerr,kcps lpread ktime,"fox_poles.lpc"

krms0 = krms0*.00001 ; low volume
asig buzz krms0, kcps, int(sr/2/kcps), 1 ; max harmonics without aliasing
aout lpreson asig
asig clip aout, 0, 1 ; prevents distortion
outs asig, asig

endin
</CsInstruments>
<CsScore>
; sine
f1 0 4096 10 1

i 1 0 2.8 1 ; first words only
i 1 4 2.8 2.8 ; whole sentence
e
</CsScore>
</CsoundSynthesizer>

```

Le fichier audio « fox.wav » dure 2.8 secondes. Ainsi la sortie contiendra une ligne comme celle-ci :

```
fox.wav = 2.756667 seconds
```

Voici un autre exemple de l'opcode lpreson. Il utilise le fichier *lpreson-2.csd* [examples/lpreson-2.csd].

### Exemple 523. Un autre exemple de l'opcode lpreson.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o lpreson-2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100

```

```
kmps = 32
nchnls = 2
0dbfs = 1
; works with or without -a option when analyzing "fox.wav" from the manual
;both options sound a little different
instr 1

ilen filelen "fox.wav" ;length of soundfile 1
prints "fox.wav = %f seconds\\n",ilen

ktime line 0, p3, ilen
krmsr,krms0,kerr,kcps lpread ktime,"fox_nopoles.lpc"
asig disk2 "flute.aiff", 1
aout lpreson asig
ares balance aout, asig
outs ares, ares

endin
</CsInstruments>
<CsScore>

i 1 0 2.8
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*lpfreson, lpread*

# lpshold

lpshold — Génère un signal de contrôle constitué de segments tenus.

## Description

Génère un signal de contrôle constitué de segments tenus délimités par deux ou plus points spécifiés. L'enveloppe entière est parcourue en boucle au taux *kfreq*. Chaque paramètre peut varier au taux-k.

## Syntaxe

```
ksig lpshold kfreq, ktrig, iphase, ktime0, kvalue0 [, kvalue1] [, ktime1] [, kvalue2] [, ktime2] [...]
```

## Exécution

*ksig* -- Signal de sortie.

*kfreq* -- Taux de répétition en Hz ou en fraction de Hz.

*ktrig* -- S'il est non nul, redéclanche l'enveloppe depuis le début (voir l'opcode *trigger*), avant que le cycle de l'enveloppe ne soit complet.

*kphase* -- point de la séquence lue, exprimé comme une fraction d'un cycle (entre 0 et 1).

*kvalue0...kvalueN* -- Valeurs des points.

*ktime0...ktimeN* -- Durées entre les points ; exprimées en fraction d'une période (voir ci-dessous). La dernière durée indique une ligne entre la valeur finale et la première valeur.

*lpshold* est semblable à *loopseg*, mais il ne peut générer que des segments horizontaux, car il maintient une valeur constante pendant chaque intervalle de temps placé entre *ktimeN* et *ktimeN+1*. Il est utile, entre autres, pour un contrôle mélodique comme celui des vieux séquenceurs analogiques.

## Exemples

Voici un exemple de l'opcode *lpshold*. Il utilise le fichier *lpshold.csd* [examples/lpshold.csd].

### Exemple 524. Exemple de l'opcode lpshold.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o lpshold.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
```

```
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
kfrq init p4 ; frequency of the loop
ifrac = p5 ; fraction of frequency at which to force retrigger
ipan = p6 ; pan position
ktrig metro kfrq * ifrac ; trigger to force restart the loop
iphs = 0 ; initial phase of the loop
; a loop of midi note numbers: note duration etc...
knote lpshold kfrq, ktrig, iphs, 61, 0.0625, 60, 0.9375, 61, 1, 58, 1, \
63, 2, 65, 3
aenv linseg 0,0.01,1,p3-0.11,1,0.1,0 ; amplitude envelope
krnd rspline -0.05,0.05,0.5,1 ; random detune
asig gbuzz 0.2*aenv, cpsmidinn(knote+krnd), 30, 1, 0.5, 1
outs asig*ipan, asig*(1-ipan)
endin

</CsInstruments>
<CsScore>
; cosine wave.
f 1 0 16384 11 1

; 3 layers of the loop are played, each at a different speed,
; - with different retriggering rate, and pan location.
i 1 0 60 0.5 [8/10] 0.5
i 1 0 60 0.375 [8/11] 0.1
i 1 0 60 0.25 [8/13] 0.9
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*loopseg*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la Version 4.13

# lpsholdp

lpsholdp — Signaux de contrôle basés sur des segments tenus.

## Description

Génère un signal de contrôle constitué de segments de droite tenus délimités par deux ou plus points spécifiés. L'enveloppe entière peut être parcourue en boucle à une vitesse variable. Chaque coordonnée de segment peut aussi varier au taux-k.

## Syntaxe

```
ksig lpsholdp kphase, ktrig, ktime0, kvalue0 [, kvalue1] [, ktime1] \  
[, kvalue2] [, ktime2] [...]
```

## Exécution

*ksig* - signal de sortie.

*kphase* -

*kvalue0...valueN* - valeurs des points.

*ktime0...ktimeN* -- Durées entre les points ; exprimées en fraction d'une période (voir ci-dessous). La dernière durée indique une ligne entre la valeur finale et la première valeur.

L'opcode *lpsholdp* est semblable à *lpshold* ; la seule différence étant que, à la place de la fréquence, une phase variable est utilisée. Si l'on utilise un phaseur pour obtenir la valeur de la phase, on aura un comportement identique à *lpshold*, mais on peut obtenir des résultats intéressants avec des phases à l'évolution non linéaire, ce qui rend *lpsholdp* plus puissant et plus général que *lpshold*.

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5. (Auparavant, disponible seulement dans CsoundAV)

# lpslot

*lpslot* — Sélectionne le slot à utiliser par des opcodes *lp* suivants.

## Description

Sélectionne le slot à utiliser par des opcodes *lp* suivants.

## Syntaxe

```
lpslot islot
```

## Initialisation

*islot* -- numéro du slot à sélectionner.

## Exécution

*lpslot* sélectionne le slot à utiliser par des opcodes *lp* suivants. C'est le moyen de charger et de référencer plusieurs analyses en même temps.

## Exemples

Here is a typical orc using the opcodes:

```
ipower init 50000 ; Define sound generator
ifreq init 440
asrc buzz ipower, ifreq, 10, 1

ctime line 0, p3, p3 ; Define time lin
lpslot 0 ; Read square data poles
krmsr, krms0, kerr, kcps lpread ctime, "square.pol"
lpslot 1 ; Read triangle data poles
krmsr, krms0, kerr, kcps lpread ctime, "triangle.pol"
kmix line 0, p3, 1 ; Compute result of mixing
lpinterp 0, 1, kmix ; and balance power
ares lpreson asrc
aout balance ares, asrc
out aout
```

## Voir aussi

*lpinterp*

## Crédits

Auteur : Mark Resibois  
Bruxelles  
1996

Nouveau dans la version 3.44

# mac

mac — Multiplie et accumule des signaux de taux-k et de taux-a.

## Description

Multiplie et accumule des signaux de taux-k et de taux-a.

## Syntaxe

```
ares mac ksig1, asig1 [, ksig2] [, asig2] [, ksig3] [, asig3] [...]
```

## Exécution

*ksig1, etc.* -- signaux d'entrée au taux-k

*asig1, etc.* -- signaux d'entrée au taux-a

*mac* multiplie et accumule des signaux de taux-k et de taux-a. Il est équivalent à :

$$\text{ares} = \text{asig1} * \text{ksig1} + \text{asig2} * \text{ksig2} + \text{asig3} * \text{ksig3} + \dots$$

## Exemples

Voici un exemple de l'opcode *mac*. Il utilise le fichier *mac.csd* [examples/mac.csd]. Il est écrit pour des systèmes \*NIX, et il générera des erreurs sous Windows.

### Exemple 525. Exemple de l'opcode *mac*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if real audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o mac.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;4 band equalizer

klow = p4 ;low gain1
klmid = p5 ;low gain2
kmidh = p6 ;high gain1
khigh = p7 ;high gain2
ifn = p8 ;table
```

```

ilc1 table 0, ifn ;low frequency range
ilc2 table 1, ifn ;low-mid
ihc1 table 2, ifn ;mid-high
ihc2 table 3, ifn ;high

asig disk2 "fox.wav", 1
alow1 butterlp asig, ilc1 ;lowpass 1
almid butterlp asig, ilc2 ;lowpass 2
amidh butterhp asig, ihc1 ;highpass 1
ahigh butterhp asig, ihc2 ;highpass 2
aout mac klow, alow1, klmid, almid, kmidh, amidh, khigh, ahigh
      outs aout, aout

endin
</CsInstruments>
<CsScore>
f1 0 4 -2 150 300 600 5000
f2 0 4 -2 75 500 1000 10000
f3 0 4 -2 200 700 1500 3000

;          low lowmid midhigh high table
i 1 0 2.8 2 1 1 1
i 1 3 2.8 2 3 1 2
i 1 6 2.8 2 1 2 3
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*maca*

## Crédits

Auteur : John ffitch  
 Université de Bath, Codemist, Ltd.  
 Bath, UK  
 Mai 1999

Nouveau dans la version 3.54 de Csound.



# maca

maca — Multiplie et accumule des signaux de taux-a seulement.

## Description

Multiplie et accumule des signaux de taux-a seulement.

## Syntaxe

```
ares maca asig1 , asig2 [, asig3] [, asig4] [, asig5] [...]
```

## Exécution

*asig1*, *asig2*, ... -- signaux d'entrée au taux-a

*maca* multiplie et accumule des signaux de taux-a seulement. Il est équivalent à :

$$\text{ares} = \text{asig1} * \text{asig2} + \text{asig3} * \text{asig4} + \text{asig5} * \text{asig6} + \dots$$

## Exemples

Voici un exemple de l'opcode *maca*. Il utilise le fichier *maca.csd* [examples/maca.csd]. Il est écrit pour les systèmes \*NIX et générera des erreurs sous Windows.

### Exemple 526. Exemple de l'opcode *maca*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if real audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o mac.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;4 band equalizer

klow = p4 ;low gain1
aenv1  oscil  0.2, 1, 4
aenv1 = aenv1 + klow
klmid = p5 ;low gain2
aenv2  oscil  0.21, 1.1, 4
aenv2 = aenv2 + klmid
kmidh = p6 ;high gain1
```

```

aenv3  oscil    0.19, 1.2, 4
aenv3 = aenv3 + kmidh
khigh  = p7 ;high gain2
aenv4  oscil    0.18, 1.3, 4
aenv4 = aenv4 + khigh

ifn    = p8 ;table

ilc1 table 0, ifn ;low frequency range
ilc2 table 1, ifn ;low-mid
ihc1 table 2, ifn ;mid-high
ihc2 table 3, ifn ;high

asig diskin2 "fox.wav", 1
alow1 butterlp asig, ilc1 ;lowpass 1
almid butterlp asig, ilc2 ;lowpass 2
amidh butterhp asig, ihc1 ;highpass 1
ahigh butterhp asig, ihc2 ;highpass 2
aout maca aenv1, alow1, aenv2, almid, aenv3, amidh, aenv4, ahigh
      outs aout, aout

endin
</CsInstruments>
<CsScore>
f1 0 4 -2 150 300 600 5000
f2 0 4 -2 75 500 1000 10000
f3 0 4 -2 200 700 1500 3000
f4 0 4096 10 1

;          low lowmid midhigh high table
i 1 0 2.8 2 1 1 1 1
i 1 3 2.8 2 3 1 1 2
i 1 6 2.8 2 1 2 3 3
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*mac*

## Crédits

Auteur : John ffitch  
 Université de Bath, Codemist, Ltd.  
 Bath, UK  
 Mai 1999

Nouveau dans la version 3.54 de Csound.

## madsr

**madsr** — Calcule l'enveloppe ADSR classique en utilisant le mécanisme de *linsegr*.

## Description

Calcule l'enveloppe ADSR classique en utilisant le mécanisme de *linsegr*.

## Syntaxe

```
ares madsr iatt, idec, islev, irel [, idel] [, ireltim]
```

```
kres madsr iatt, idec, islev, irel [, idel] [, ireltim]
```

## Initialisation

*iatt* -- durée de l'attaque (attack)

*idec* -- durée de la première chute (decay)

*islev* -- niveau d'entretien (sustain)

*irel* -- durée de la chute (release)

*idel* -- délai de niveau zéro avant le démarrage de l'enveloppe

*ireltim* (facultatif, par défaut = -1) -- Contrôle la durée du relâchement après la réception d'un évènement MIDI note-off. S'il est inférieur à zéro, la durée de relâchement la plus longue de l'instrument courant est utilisée. S'il est nul ou positif, la valeur donnée sera utilisée comme durée de relâchement. Sa valeur par défaut est -1. (Nouveau dans Csound 3.59 - pas encore entièrement testé).

Noter que la durée du relâchement ne pouvait pas dépasser  $32767/kr$  secondes dans versions antérieures à la 5.00 ; depuis la limite a été étendue à  $2^{31}/kr$  secondes.

## Exécution

L'enveloppe évolue dans l'intervalle de 0 à 1 et peut être changée d'échelle par la suite. Voici une description de l'enveloppe :

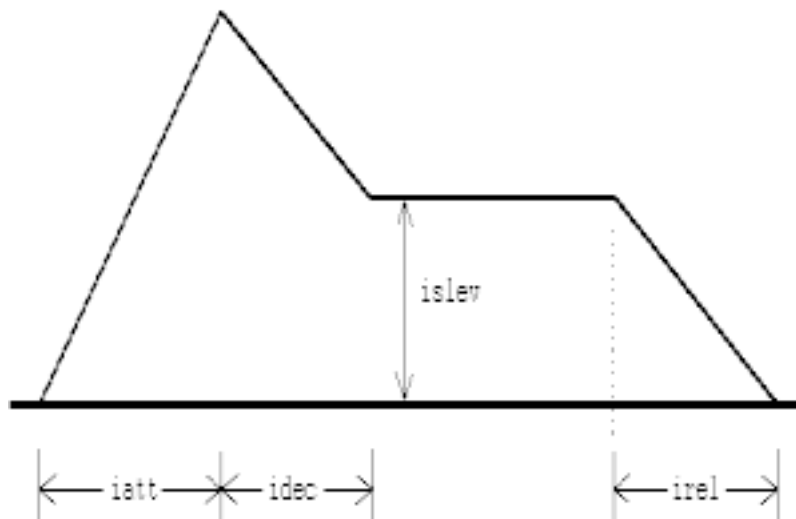


Image d'une enveloppe ADSR.

La longueur de la période d'entretien est calculée à partir de la longueur de la note. C'est pourquoi *adsr* n'est pas adapté au traitement des événements MIDI. L'opcode *madsr* utilise le mécanisme de *linsegr*, et peut donc être utilisé dans les applications MIDI.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note-off, comme *linsegr* et *expsegr*, ou bien l'on peut construire des enveloppes plus complexes au moyen de *xtratim* et de *release*. Noter qu'il n'est pas nécessaire d'utiliser *xtratim* avec *madsr*, car la durée est allongée automatiquement.



## Note

Les durées pour *iatt*, *idec* et *irel* ne peuvent pas être 0. Si l'on utilise 0, aucune enveloppe n'est générée. Utilisez une très petite valeur comme 0.0001 si vous désirez une attaque, une chute ou un relâchement instantanés.

## Exemples

Voici un exemple de l'opcode *madsr*. Il utilise le fichier *madsr.csd* [examples/madsr.csd].

### Exemple 527. Exemple de l'opcode *madsr*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o madsr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Iain McCurdy */
; Initialize the global variables.
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

; Instrument #1.
instr 1
; Attack time.
iattack = 0.5
; Decay time.
idecay = 0
; Sustain level.
isustain = 1
; Release time.
irelease = 0.5
aenv madsr iattack, idecay, isustain, irelease

a1 oscili 10000, 440, 1
out a1*aenv
endin

</CsInstruments>
```

```
<CsScore>

/* Written by Iain McCurdy */
; Table #1, a sine wave.
f 1 0 1024 10 1

; Leave the score running for 6 seconds.
f 0 6

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode `madsr`, utilisant l'entrée midi. Il utilise le fichier `madsr-2.csd` [exemples/madsr-2.csd].

### Exemple 528. Second exemple de l'opcode `madsr`

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0    ;;realtime audio out and realtime midi in
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o madsr-2.wav -W    ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

icps cpsmidi
iamp ampmidi .5

kenv madsr 0.5, 0, 1, 0.5
asig pluck kenv, icps, icps, 2, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 2 0 4096 10 1

f0 30 ;runs 30 seconds
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*linsegr, expsegr, envlpxr, mxadsr, madsr, xadsr expon, expseg, expsega line, linseg, xtratim*

## Crédits

Auteur : John ffitch

Novembre 2002. Merci à Rasmus Ekman pour avoir documenté le paramètre *ireltim*.

Décembre 2002. Merci à Iain McCurdy pour avoir ajouté un exemple.

Décembre 2002. Merci à Istvan Varga pour avoir indiqué la durée maximale de relâchement.

Nouveau dans la version 3.49 de Csound.

# mags

**mags** — Retourne les modules d'un tableau de nombres complexes.

## Description

Cet opcode retourne les modules d'un tableau de nombres complexes (en format rfft) dans un tableau de réels dont la taille est la moitié de celle du tableau d'entrée plus un. Le module pour la fréquence de Nyquist se trouve dans la dernière position du tableau.

## Syntaxe

```
kout[] mags kin[]
```

## Exécution

*kout[]* -- tableau contenant les modules (taille = taille d'entrée / 2). Créé s'il n'existe pas.

*kin[]* -- tableau contenant les valeurs complexes réelles-imaginaires d'entrée.

## Exemples

Voici un exemple de l'opcode **mags**. Il utilise le fichier *mags.csd* [examples/mags.csd].

### Exemple 529. Exemple de l'opcode **mags**.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-d -o dac
</CsOptions>
<CsInstruments>
/* ksmpls needs to be an integer div of
   hopsize */
ksmps = 64
0dbfs=1
nchnls=2

opcode PVA,k[],k[],k,aii
  asig, isize, ihop xin
  iolaps init isize/ihop
  kcmt init 0
  krow init 1
  kIn[] init isize
  kOlph[] init isize/2 + 1
  ifac = (sr/(ihop*2*$M_PI))
  iscal = (2*$M_PI*ihop/isize)
  kfl = 0
  kIn shiftin asig
  if kcmt == ihop then
    kWin[] window kIn, krow*ihop
    kSpec[] rfft kWin
    kMags[] mags kSpec
    kPha[] phas kSpec
    kDelta[] = kPha - kOlph
```

```

kOlpk = kPha
kk = 0
kDelta unwrap kDelta
while kk < isize/2 do
  kPha[kk] = (kDelta[kk] + kk*iscal)*ifac
  kk += 1
od
krow = (krow+1)%iolaps
kcnt = 0
kfl = 1
endif
xout kMags,kPha,kfl
kcnt += ksmps
endop

opcode PVS,a,k[]k[]kii
kMags[],kFr[],kfl,isize,ihop xin
iolaps init isize/ihop
ifac = ihop*2*$M_PI/sr;
iscal = sr/isize
krow init 0
kOla[] init isize
kOut[][] init iolaps,isize
kPhs[] init isize/2+1
if kfl == 1 then
  kk = 0
  while kk < isize/2 do
    kFr[kk] = (kFr[kk] - kk*iscal)*ifac
    kk += 1
  od
  kPhs = kFr + kPhs
  kSpec[] pol2rect kMags,kPhs
  kRow[] rifft kSpec
  kWin[] window kRow, krow*ihop
  kOut setrow kWin, krow
  kOla = 0
  kk = 0
  until kk == iolaps do
    kRow getrow kOut, kk
    kOla = kOla + kRow
    kk += 1
  od
  krow = (krow+1)%iolaps
endif
xout shiftout(kOla)/iolaps
endop

instr 1

ihopsize = 256 ; hopsize
ifftsize = 2048 ; FFT size
kFreqsOut[] init ifftsize/2+1 ; synthesis freqs
kMagsOut[] init ifftsize/2+1 ; synthesis mags

a1 diskin2 "fox.wav",1,0,1

kMags[],kFreqs[],kflg PVA a1,ifftsize,ihopsize

if kflg == 1 then
  ki = 0
  kMagsOut = 0
  kFreqsOut = 0
  iscal = 1.5
  until ki == ifftsize/2 do
    if ki*iscal < ifftsize/2 then
      kFreqsOut[ki*iscal] = kFreqs[ki]*iscal
      kMagsOut[ki*iscal] = kMags[ki]
    endif
    ki += 1
  od

```



```
endif
  ki += 1
od
endif

a2 PVS kMagsOut,kFreqsOut,kflg,ifftsize,ihopsize

a1 delay a1, (ifftsize+ihopsize)/sr
outs a1, a2
endin

</CsInstruments>
<CsScore>
i1 0 10
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux.

## Crédits

Auteur : Victor Lazzarini  
NUI Maynooth  
2014

Nouveau dans la version 6.04

# mandel

mandel — Ensemble de Mandelbrot.

## Description

Retourne le nombre d'itérations correspondant à un point donné du plan complexe auquel on applique les formules de l'ensemble de Mandelbrot.

## Syntaxe

`kiter, koutrig mandel ktrig, kx, ky, kmaxIter`

## Exécution

*kiter* - nombre d'itérations

*koutrig* - signal de déclenchement en sortie

*ktrig* - signal de déclenchement en entrée

*kx, ky* - coordonnées d'un point appartenant au plan complexe

*kmaxIter* - nombre maximum d'itérations autorisé

*mandel* est un opcode qui utilise les formules de l'ensemble de Mandelbrot pour générer une sortie que l'on peut appliquer à n'importe quel paramètre musical (ou non musical). Il comprend deux paramètres de sortie : *kiter*, qui contient le nombre d'itérations d'un point donné, et *koutrig*, qui génère un 'bang' de déclenchement chaque fois que *kiter* change. L'évaluation d'un nouveau nombre d'itérations n'a lieu que lorsque *ktrig* prend une valeur non nulle. Les coordonnées du plan complexe sont fixées dans *kx* et *ky*, tandis que *kmaxIter* contient le nombre maximum d'itérations. Les valeurs de sorties, qui sont des nombres entiers, peuvent être interprétées de n'importe quelle manière par le compositeur.

## Exemples

Voici un exemple de l'opcode *mandel*. Il utilise le fichier *mandel.csd* [examples/mandel.csd].

### Exemple 530. Exemple de l'opcode *mandel*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o mandel.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
;example by Brian Evans
sr = 44100
ksmps = 32
nchnls = 2
```

```

instr 1

ipitchtable = 1      ; pitch table in score
ipitchndx = p5       ; p5=pitch index from table

ipitch table ipitchndx, ipitchtable
kenv expseg 1.0, 1.0, 1.0, 11.5, .0001
asig pluck ampdb(p4)*kenv, cpspch(ipitch), cpspch(ipitch), 0, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>

f1 0 32 -2 6.00 6.02 6.04 6.05 6.07 6.09 6.11 ; f1 is a pitch table defining a four octave C major scale
      7.00 7.02 7.04 7.05 7.07 7.09 7.11 ; on C two octaves below middle C
      8.00 8.02 8.04 8.05 8.07 8.09 8.11
      9.00 9.02 9.04 9.05 9.07 9.09 9.11

;ins start dur ampdb(p4) pitchndx(p5)

i1 0 12.0 75 3
i1 1.5999 12.0 75 4
i1 3.4000 12.0 75 5
i1 4.2000 12.0 75 6
i1 4.4000 12.0 75 7
i1 4.6000 12.0 75 9
i1 4.8000 12.0 75 10
i1 5.0000 12.0 75 5
i1 5.2000 12.0 75 27
i1 5.4000 12.0 75 5
i1 5.6000 12.0 75 20
i1 6.0000 12.0 75 24
i1 6.2000 12.0 75 2
i1 6.4000 12.0 75 27
i1 6.6000 12.0 75 20
i1 6.8000 12.0 75 15
i1 7.0000 12.0 75 3
i1 7.2000 12.0 75 3
i1 7.4000 12.0 75 23
i1 7.6000 12.0 75 9
i1 7.8000 12.0 75 17
i1 8.0000 12.0 75 18
i1 8.2000 12.0 75 3
i1 8.4000 12.0 75 26
i1 8.6000 12.0 75 15
i1 8.8 12.0 75 2
i1 9 12.0 75 26
i1 9.2 12.0 75 8
i1 9.3999 12.0 75 22
i1 9.5999 12.0 75 22
i1 9.7999 12.0 75 20
i1 9.9999 12.0 75 19
i1 10.399 12.0 75 20
i1 10.799 12.0 75 22
i1 10.999 12.0 75 27
i1 11.199 12.0 75 25
i1 11.399 12.0 75 20
i1 11.599 12.0 75 21
i1 11.799 12.0 75 24
i1 11.999 12.0 75 24
i1 12.199 12.0 75 4
i1 12.399 12.0 75 13
i1 12.599 12.0 75 15
i1 12.799 12.0 75 14
i1 12.999 12.0 75 3
i1 13.199 12.0 75 21

```

```

i1 13.399 12.0 75 6
i1 13.599 12.0 75 3
i1 13.799 12.0 75 10
i1 13.999 12.0 75 25
i1 14.199 12.0 75 21
i1 14.399 12.0 75 20
i1 14.599 12.0 75 19
i1 14.799 12.0 75 18
i1 15.199 12.0 75 17
i1 15.599 12.0 75 16
i1 15.999 12.0 75 15
i1 16.599 12.0 75 14
i1 17.199 12.0 75 13
i1 18.399 12.0 75 12
i1 18.599 12.0 75 11
i1 19.199 12.0 75 10
i1 19.799 12.0 75 9
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

Pour plus d'information sur cet opcode : *Composing Fractal Music with Csound* [<http://mymbbs.mbs.net/~pfisher/FOV2-0010016C/FOV2-0010016E/FOV2-001001A3/chapters/35evans/intro.html>], par Brian Evans

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Seulement disponible auparavant dans CsoundAV)

# mandol

mandol — Une simulation de mandoline.

## Description

Une simulation de mandoline.

## Syntaxe

```
ares mandol kamp, kfreq, kpluck, kdetune, kgain, ksize \  
    [, ifn] [, iminfreq]
```

## Initialisation

*ifn* -- numéro d'une table contenant la forme d'onde du pincement de corde. Le fichier *mandpluk.aiff* [examples/mandpluk.aiff] convient pour cela. On peut aussi l'obtenir à <ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/>.

*iminfreq* (facultatif, 0 par défaut) -- Fréquence la plus basse pour une note. Si ce paramètre est omis, il prend la valeur initiale de *kfreq*.

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note.

*kpluck* -- Position du pincement sur la corde, compris entre 0 et 1. Valeur suggérée : 0,4.

*kdetune* -- Proportion de désaccord entre les deux cordes. La valeur suggérée va de 0,9 à 1.

*kgain* -- le gain de la boucle du modèle, compris entre 0,97 et 1.

*ksize* -- La taille du corps de la mandoline. Compris entre 0 et 2.

## Exemples

Voici un exemple de l'opcode mandol. Il utilise les fichiers *mandol.csd* [examples/mandol.csd], et *mandpluk.aiff* [examples/mandpluk.aiff].

### Exemple 531. Exemple de l'opcode mandol.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac      ;;realtime audio out  
;-iadc      ;;uncomment -iadc if realtime audio input is needed too  
; For Non-realtime ouput leave only the line below:
```

```

; -o mandol.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kamp      = p4
ksize     = p5
kdetune   = p6
asig mandol kamp, 880, .4, kdetune, 0.99, ksize, 1, 220
outs asig, asig

endin
</CsInstruments>
<CsScore>
; "mandpluk.aiff" audio file
f 1 0 8192 1 "mandpluk.aiff" 0 0 0

i 1 .5 1 1 2 .99
i 1 + 1 .5 1 .99 ;lower volume to compensate
i 1 + 3 .3 .3 .99 ;lower volume to compensate

i 1 4 1 1 2 .39 ;change detune value
i 1 + 1 .5 1 .39
i 1 + 3 .3 .3 .39
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 3.47 de Csound

# maparray

maparray — Applique une fonction à chaque élément d'un vecteur.

## Description

Applique une fonction d'un argument à chaque élément d'un vecteur (tableau unidimensionnel de taux-k).

## Syntaxe

```
karray maparray kinarray, String  
karray maparray_i kinarray, String
```

## Initialisation

*String* -- une chaîne de caractères nommant un opcode fonctionnel, au taux-i pour *tabmap\_i* ou au taux-k pour *tabmap*.

## Exécution

*karray* -- tableau pour les images de la fonction.

*kinarray* -- tableau pour les arguments de la fonction.

## Exemples

Voici un exemple de l'opcode maparray. Il utilise le fichier *maparray.csd* [examples/maparray.csd].

### Exemple 532. Exxemple de l'opcode maparray.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-n  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 32  
nchnls = 2  
0dbfs = 1  
  
instr 1  
  
;create an array and fill with numbers  
kArrSrc[] fillarray 1.01, 2.02, 3.03, 4.05, 5.08, 6.13, 7.21  
  
;print source array  
printf "%s", 1, "\nSource array:\n"  
kndx = 0  
until kndx == lenarray(kArrSrc) do  
printf "kArrSrc[%d] = %f\n", kndx+1, kndx, kArrSrc[kndx]
```

```

kndx    +=    1
od

;create an empty array for the results
kArrRes[] init 7

;apply the sqrt() function to each element
kArrRes maparray kArrSrc, "sqrt"

;print the result
printf "%s", 1, "\nResult after applying sqrt() to source array\n"
kndx    =    0
until kndx == lenarray(kArrRes) do
printf "kArrRes[%d] = %f\n", kndx+1, kndx, kArrRes[kndx]
kndx    +=    1
od

;apply the log() function to each element
kArrRes maparray kArrSrc, "log"

;print the result
printf "%s", 1, "\nResult after applying log() to source array\n"
kndx    =    0
until kndx == lenarray(kArrRes) do
printf "kArrRes[%d] = %f\n", kndx+1, kndx, kArrRes[kndx]
kndx    +=    1
od

;apply the int() function to each element
kArrRes maparray kArrSrc, "int"

;print the result
printf "%s", 1, "\nResult after applying int() to source array\n"
kndx    =    0
until kndx == lenarray(kArrRes) do
printf "kArrRes[%d] = %f\n", kndx+1, kndx, kArrRes[kndx]
kndx    +=    1
od

;apply the frac() function to each element
kArrRes maparray kArrSrc, "frac"

;print the result
printf "%s", 1, "\nResult after applying frac() to source array\n"
kndx    =    0
until kndx == lenarray(kArrRes) do
printf "kArrRes[%d] = %f\n", kndx+1, kndx, kArrRes[kndx]
kndx += 1
od

;turn instrument instance off
turnoff

endin

</CsInstruments>
<CsScore>
i 1 0 0.1
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

Opcodes vectoriels



## Crédits

Auteur : John ffitch  
Codemist Ltd  
2013

Nouveau dans la version 6.00

# marimba

marimba — Modèle physique de la frappe d'un bloc de bois.

## Description

La sortie audio est un son tel que celui produit lorsque l'on frappe un bloc de bois comme dans un marimba. Il s'agit d'un modèle physique développé d'après Perry Cook mais recodé pour Csound.

## Syntaxe

```
ares marimba kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec \  
    [, idoubles] [, itriples]
```

## Initialisation

*ihrd* -- la dureté de la baguette utilisée pour la frappe. On utilise un intervalle allant de 0 à 1. 0,5 est une valeur adéquate.

*ipos* -- le point d'impact sur le bloc, compris entre 0 et 1.

*imp* -- une table des impulsions de la frappe. Le fichier *marmstk1.wav* [examples/marmstk1.wav] contient une fonction adéquate créée à partir de mesures et l'on peut le charger dans une table *GEN01*. Il est aussi disponible à <ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/>.

*ivfn* -- forme du vibrato, habituellement une table sinus, créée par une fonction

*idec* -- durée avant la fin de la note lorsqu'il y a une atténuation

*idoubles* (facultatif) -- pourcentage de frappes doubles. La valeur par défaut est de 40%.

*itriples* (facultatif) -- pourcentage de frappes triples. La valeur par défaut est de 20%.

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note.

*kvibf* -- Fréquence du vibrato en Hertz. L'intervalle conseillé va de 0 à 12.

*kvamp* -- Amplitude du vibrato.

## Exemples

Voici un exemple de l'opcode marimba. Il utilise les fichiers *marimba.csd* [examples/marimba.csd] et *marmstk1.wav* [examples/marmstk1.wav].

### Exemple 533. Exemple de l'opcode marimba.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o marimba.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 128
nchnls = 2

; Instrument #1.
instr 1
  ifreq = cpspch(p4)
  ihrd = 0.1
  ipos = 0.561
  imp = 1
  kvibf = 6.0
  kvamp = 0.05
  ivibfn = 2
  idec = 0.6

  a1 marimba 20000, ifreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec, 20, 10

  outs a1, a1
endin

</CsInstruments>
<CsScore>

; Table #1, the "marmstkl.wav" audio file.
f 1 0 256 1 "marmstkl.wav" 0 0 0
; Table #2, a sine wave for the vibrato.
f 2 0 128 10 1

; Play Instrument #1 for one second.
i 1 0 1 8.09
i 1 + 0.5 8.00
i 1 + 0.5 7.00
i 1 + 0.25 8.02
i 1 + 0.25 8.01
i 1 + 0.25 7.09
i 1 + 0.25 8.02
i 1 + 0.25 8.01
i 1 + 0.25 7.09
i 1 + 0.3333 8.09
i 1 + 0.3333 8.02
i 1 + 0.3334 8.01
i 1 + 0.25 8.00
i 1 + 0.3333 8.09
i 1 + 0.3333 8.02
i 1 + 0.25 8.01
i 1 + 0.3333 7.00
i 1 + 0.3334 6.00

e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*vibes*

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
Université de Bath, Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.47 de Csound

# massign

massign — Affecte un numéro de canal MIDI à un instrument de Csound.

## Description

Affecte un numéro de canal MIDI à un instrument de Csound.

## Syntaxe

```
massign ichnl, insnum[, ireset]
massign ichnl, "insname"[, ireset]
```

## Initialisation

*ichnl* -- numéro de canal MIDI (1-16).

*insnum* -- numéro de l'instrument d'orchestre de Csound. S'il est inférieur ou égal à zéro, le canal est désactivé (c-à-d. qu'il ne déclenche aucun instrument de csound, bien que l'information soit toujours reçue par des opcodes tels que *midin*).

« *insname* » -- une chaîne de caractères entre guillemets représentant un nom d'instrument.

*ireset* -- s'il est non nul, les contrôleurs sont réinitialisés ; c'est le comportement par défaut.

## Exécution

Affecte un numéro de canal MIDI à un instrument de Csound. Egalement utile pour s'assurer qu'un instrument particulier (si son numéro est compris entre 1 et 16) ne sera pas déclenché par des messages MIDI noteon (si l'on utilise quelque chose comme *midin* pour interpréter l'information MIDI). Dans ce cas, fixer *insnum* à un nombre inférieur ou égal à 0.

Si *ichan* est fixé à 0, la valeur de *insnum* est utilisée pour tous les canaux. On peut envoyer de cette manière tous les canaux MIDI vers un seul instrument de Csound. On peut aussi empêcher le déclenchement des instruments à partir d'événements de note MIDI en provenance de tous les canaux avec la ligne suivante :

```
massign 0, 0
```

Ceci peut être utile si l'on effectue toutes les évaluations MIDI dans Csound avec un instrument actif en permanence (par exemple en utilisant *midin* et *turnon*) pour éviter une doublure de l'instrument quand une note est jouée.

## Exemples

Voici un exemple de l'opcode massign. Il utilise le fichier *massign.csd* [examples/massign.csd].

### Exemple 534. Exemple de l'opcode massign.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
```

```

<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0      ;;realtime audio out and realtime midi in
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o massign.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giengine fluidEngine
; soundfont path to manual/examples
isfnum fluidLoad "19Trumpet.sf2", giengine, 1
fluidProgramSelect giengine, 1, isfnum, 0, 56

massign 0,0 ;disable triggering of all instruments on all channels, but
massign 12,10 ;assign instr. 10 to midi channel 12
massign 3,30 ;assign instr. 30 to midi channel 3

instr 10 ; soundfont only on midi channel 12

mididefault 60, p3
midinoteonkey p4, p5 ; in midi notes
ikey init p4
ivel init p5
fluidNote giengine, 1, ikey, ivel
endin

instr 30 ; FM-oscillator only on midi channel 3

mididefault 60, p3
midinoteoncps p4, p5 ; in Hertz
icps init p4
iamp init p5
iamp = iamp/127
kenv madsr 0.5, 0, 1, 0.5
asig foscil iamp*kenv, icps, 1, 1.414, 2, 1
outs asig, asig
endin

instr 99 ; output sound from fluidengine

imvol init 7
aL, aR fluidOut giengine
outs aL*imvol, aR*imvol
endin
</CsInstruments>
<CsScore>
; sine
f 1 0 16384 10 1

i 10 0 2 60 100 ;one note on the trumpet in midi and...
i 30 2 2 220 80 ;one FM note in Hz
i 99 0 60 ;stay active for 60 sec.
e

</CsScore>
</CsSoundSynthesizer>

```

## Voir aussi

*ctrlinit*

## Crédits

Auteur : Barry L. Vercoe - Mike Berry  
MIT, Cambridge, Mass.

Nouveau dans la version 3.47 de Csound

Le paramètre *ireset* est nouveau dans Csound5

Merci à Rasmus Ekman pour avoir indiqué le bon intervalle pour le numéro de canal MIDI.

# max

max — Produit un signal qui est le maximum de tous les signaux d'entrée.

## Description

L'opcode *max* prend en entrée n'importe quel nombre de signaux de taux-a, de taux-k ou de taux-i (tous au même taux), et retourne un signal du même taux qui est le maximum de toutes les entrées. Pour les signaux de taux-a, les entrées sont comparées échantillon par échantillon (c-à-d que *max* n'examine pas une période *ksmps* entière du signal pour trouver son maximum local comme l'opcode *max\_k* le fait).

## Syntaxe

```
amax max ain1, ain2 [, ain3] [, ain4] [...]
```

```
kmax max kin1, kin2 [, kin3] [, kin4] [...]
```

```
imax max iin1, iin2 [, iin3] [, iin4] [...]
```

## Exécution

*ain1, ain2, ...* -- signaux de taux-a à comparer.

*kin1, kin2, ...* -- signaux de taux-k à comparer.

*iin1, iin2, ...* -- signaux de taux-i à comparer.

## Exemples

Voici un exemple de l'opcode max. Il utilise le fichier *max.csd* [examples/max.csd].

### Exemple 535. Exemple de l'opcode max.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o max.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

k1  oscili 1, 10.0, 1    ;combine 3 sinusses
k2  oscili 1, 1.0, 1    ;at different rates
```



```

k3  oscili 1, 3.0, 1
kmax max k1, k2, k3
kmax = kmax*250 ;scale kmax
printk2 kmax ;check the values

aout vco2 .5, 220, 6 ;sawtooth
asig moogvcf2 aout, 600+kmax, .5 ;change filter around 600 Hz
outs asig, asig

endin

</CsInstruments>
<CsScore>
f1 0 32768 10 1

i1 0 5
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*min, maxabs, minabs, maxaccum, minaccum, maxabsaccum, minabsaccum, max\_k*

## Crédits

Auteur : Anthony Kozar  
Mars 2006

Nouveau dans la version 5.01 de Csound ; taux-i ajouté dans la version 6.04

# maxabs

**maxabs** — Produit un signal qui est le maximum des valeurs absolues de n'importe quel nombre de signaux d'entrée.

## Description

L'opcode *maxabs* prend en entrée n'importe quel nombre de signaux de taux-a ou de taux-k (tous du même taux), et retourne un signal au même taux qui est le maximum de toutes les entrées. Il est identique à l'opcode *max* sauf qu'il prend la valeur absolue de chaque entrée avant de les comparer. Ainsi, la sortie est toujours non-négative. Pour les signaux de taux-a, les entrées sont comparées échantillon par échantillon (c-à-d que *maxabs* n'examine pas une période *ksmps* entière du signal pour trouver son maximum local comme l'opcode *max\_k* le fait).

## Syntaxe

```
amax maxabs ain1, ain2 [, ain3] [, ain4] [...]  
kmax maxabs kin1, kin2 [, kin3] [, kin4] [...]
```

## Exécution

*ain1, ain2, ...* -- signaux de taux-a à comparer.

*kin1, kin2, ...* -- signaux de taux-k à comparer.

## Exemples

Voici un exemple de l'opcode *maxabs*. Il utilise le fichier *maxabs.csd* [examples/maxabs.csd].

### Exemple 536. Exemple de l'opcode *maxabs*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac    ;;;realtime audio out  
;-iadc    ;;;uncomment -iadc if realtime audio input is needed too  
; For Non-realtime ouput leave only the line below:  
; -o maxabs.wav -W ;;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 32  
nchnls = 2  
0dbfs = 1  
  
instr 1  
  
k1    oscili 1, 10.0, 1    ;combine 3 sinusses  
k2    oscili 1, 1.0, 1     ;at different rates  
k3    oscili 1, 3.0, 1
```

```

kmax maxabs k1, k2, k3
kmax = kmax*250 ;scale kmax
printk2 kmax ;check the values

aout vco2 .5, 220, 6 ;sawtooth
asig moogvcf2 aout, 600+kmax, .5 ;change filter above 600 Hz
outs asig, asig

endin

</CsInstruments>
<CsScore>
f1 0 32768 10 1

i1 0 5
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*minabs, max, min, maxaccum, minaccum, maxabsaccum, minabsaccum, max\_k*

## Crédits

Auteur : Anthony Kozar  
Mars 2006

Nouveau dans la version 5.01 de Csound.

# maxabsaccum

maxabsaccum — Accumule le maximum de la valeur absolue de signaux audio.

## Description

*maxabsaccum* compare deux variables de taux-audio et écrit le maximum de la valeur absolue des deux dans la première.

## Syntaxe

```
maxabsaccum aAccumulator, aInput
```

## Exécution

*aAccumulator* -- variable audio dans laquelle la valeur maximale est écrite.

*aInput* -- signal auquel *aAccumulator* est comparé.

L'opcode *maxabsaccum* est conçu pour accumuler la valeur maximale de plusieurs signaux audio qui peuvent provenir de plusieurs instances de note, dans différents canaux, ou qui ne peuvent être comparés en une fois au moyen de l'opcode *maxabs*. *maxabsaccum* est identique à *maxaccum* sauf qu'il prend la valeur absolue de *aInput* avant la comparaison. Sa sémantique est semblable à celle de *vincr* car *aAccumulator* est utilisé à la fois comme variable d'entrée et comme variable de sortie, sauf que *maxabsaccum* garde la valeur absolue maximale au lieu d'additionner les signaux ensemble. *maxabsaccum* exécute l'opération suivante sur chaque paire d'échantillons :

```
if (abs(aInput) > aAccumulator) aAccumulator = abs(aInput)
```

*aAccumulator* sera habituellement une variable audio globale. A la fin de chaque cycle de calcul (période-k), après que sa valeur ait été lue et utilisée, la variable accumulateur doit être remise à zéro (peut-être en utilisant l'opcode *clear*). La remise à zéro est suffisante pour *maxabsaccum*, à la différence de l'opcode *maxaccum*.

## Exemples

Voici un exemple de l'opcode *maxabsaccum*. Il utilise le fichier *maxabsaccum.csd* [examples/maxabsaccum.csd].

### Exemple 537. Exemple de l'opcode maxabsaccum.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o maxabsaccum.wav -W ;; for file output any platform
```

```
</CsOptions>
<CsInstruments>

  sr = 44100
  ksmps = 32
  nchnls = 2
  odbfs = 1

  instr 1 ;saw

  asig vco2 .2, p4
      outs asig, asig
  gasaw = asig
  endin

  instr 2 ;sine

  aout poscil .3, p4, 1
      outs aout, aout
  gasin = aout
  endin

  instr 10

  accum init 0
      maxabsaccum accum, gasaw + gasin ;saw and sine accumulated
  accum dcblock2 accum ;get rid of DC
      outs accum, accum

  clear accum
  endin

</CsInstruments>
<CsScore>
f 1 0 4096 10 1 ;sine wave

i 1 0 7 330
i 2 3 3 440

i 1 10 7 330 ;same notes but without maxabsaccum, for comparison
i 2 13 3 440

i 10 0 6 ;accumulation note stops after 6 seconds

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*minabsaccum, maxaccum, minaccum, max, min, maxabs, minabs, vincr, clear*

## Crédits

Auteur : Anthony Kozar  
Mars 2006

Nouveau dans la version 5.01 de Csound.

# maxaccum

maxaccum — Accumule la valeur maximale de signaux audio.

## Description

*maxaccum* compare deux variables de taux-audio et écrit la valeur maximale des deux dans la première.

## Syntaxe

```
maxaccum aAccumulator, aInput
```

## Exécution

*aAccumulator* -- variable audio dans laquelle la valeur maximale est écrite.

*aInput* -- signal auquel *aAccumulator* est comparé.

L'opcode *maxaccum* est conçu pour accumuler la valeur maximale de plusieurs signaux audio qui peuvent provenir de plusieurs instances de note, dans différents canaux, ou qui ne peuvent être comparés en une fois au moyen de l'opcode *max*. Sa sémantique est semblable à celle de *vincr* car *aAccumulator* est utilisé à la fois comme variable d'entrée et comme variable de sortie, sauf que *maxaccum* garde la valeur maximale au lieu d'additionner les signaux ensemble. *maxaccum* exécute l'opération suivante sur chaque paire d'échantillons :

```
if (aInput > aAccumulator) aAccumulator = aInput
```

*aAccumulator* sera habituellement une variable audio globale. A la fin de chaque cycle de calcul (période-k), après que sa valeur ait été lue et utilisée, la variable accumulateur doit être remise à zéro (peut-être en utilisant l'opcode *clear*). Il faut faire cependant attention si *aInput* est négatif en tout point, auquel cas l'accumulateur doit être initialisé et réinitialisé à une valeur négative suffisamment importante pour être toujours inférieure aux signaux d'entrée auxquels elle est comparé.

## Exemples

Voici un exemple de l'opcode maxaccum. Il utilise le fichier *maxaccum.csd* [exemples/maxaccum.csd].

### Exemple 538. Exemple de l'opcode maxaccum.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime output leave only the line below:
; -o maxaccum.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;saw

asig vco2 .2, p4
outs asig, asig
gasaw = asig
endin

instr 2 ;sine

aout poscil .3, p4, 1
outs aout, aout
gasin = aout
endin

instr 10

accum init 0
maxaccum accum, gasaw + gasin ;saw and sine accumulated
outs accum, accum

clear accum
endin

</CsInstruments>
<CsScore>
f 1 0 4096 10 1

i 1 0 7 330
i 2 3 3 440

i 1 10 7 330 ;same notes but without maxaccum, for comparison
i 2 13 3 440

i 10 0 6 ;accumulation note stops after 6 seconds

e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*minaccum, maxabsaccum, minabsaccum, max, min, maxabs, minabs, vincr, clear*

## Crédits

Auteur : Anthony Kozar  
Mars 2006

Nouveau dans la version 5.01 de Csound.

# maxalloc

maxalloc — Limite le nombre d'allocations pour un instrument.

## Description

Limite le nombre d'allocations pour un instrument.

## Syntaxe

```
maxalloc insnum, icount  
maxalloc Sinsname, icount
```

## Initialisation

*insnum* -- numéro de l'instrument

*Sinsname* -- nom de l'instrument

*icount* -- nombre d'allocations de l'instrument

## Exécution

*maxalloc* limite le nombre d'instances simultanées (notes) d'un instrument. Tout évènement de partition survenant après que le maximum soit atteint est ignoré.

Toutes les instances de *maxalloc* doivent être définies dans la section d'en-tête, pas dans le corps de l'instrument.

## Exemples

Voici un exemple de l'opcode maxalloc. Il utilise le fichier *maxalloc.csd* [examples/maxalloc.csd].

### Exemple 539. Exemple de l'opcode maxalloc.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac      ;;realtime audio out  
;-iadc     ;;uncomment -iadc if realtime audio input is needed too  
; For Non-realtime ouput leave only the line below:  
; -o maxalloc.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 32  
nchnls = 2  
0dbfs = 1
```



```

maxalloc 1, 3 ; Limit to three instances.

instr 1

asig oscil .3, p4, 1
      outs asig, asig

endin
</CsInstruments>
<CsScore>
; sine
f 1 0 32768 10 1

i 1 0 5 220 ;1
i 1 1 4 440 ;2
i 1 2 3 880 ;3, limit is reached
i 1 3 2 1320 ;is not played
i 1 4 1 1760 ;is not played
e
</CsScore>
</CsoundSynthesizer>

```

La sortie contiendra des messages comme ceux-ci :

```
WARNING: cannot allocate last note because it exceeds instr maxalloc
```

## Voir aussi

*cpuprc, prealloc*

## Crédits

Auteur : Gabriel Maldonado  
 Italie  
 Juillet 1999

Nouveau dans la version 3.57 de Csound

# max\_k

max\_k — Maximum (ou minimum) local d'un signal entrant de taux-a.

## Description

*max\_k* retourne le maximum (ou le minimum) local du signal entrant *asig*, mesuré dans l'intervalle de temps entre deux passages à true de *ktrig*.

## Syntaxe

knumkout **max\_k** asig, ktrig, itype

## Initialisation

*itype* - détermine le comportement de *max\_k* (voir ci-dessous)

## Exécution

*asig* -- signal entrée

*ktrig* -- signal de déclenchement

*max\_k* retourne le maximum (ou le minimum) local du signal entrant *asig*, mesuré dans l'intervalle de temps entre deux passages à true de *ktrig*. *itype* détermine le comportement de *max\_k* :

- 1 - maximum absolu (les valeurs négatives sont changées en valeurs positives avant l'évaluation)
- 2 - maximum courant
- 3 - minimum courant
- 4 - calcule la valeur moyenne de *asig* dans l'intervalle de temps depuis le dernier déclenchement.

Cet opcode peut être utile dans plusieurs situations, par exemple pour implémenter un vu-mètre.

## Exemples

Voici un exemple de l'opcode max\_k. Il utilise le fichier *max\_k.csd* [examples/max\_k.csd].

### Exemple 540. Exemple de l'opcode max\_k.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -Ma    ;;realtime audio out and midi in (on all inputs)
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o max_k.wav -W ;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

FLpanel "This Panel contains VU-meter",300,100
gk1,gih1 FLslider "VU-meter", 0,1,0,1, -1, 250,30, 30,30
FLsetColor2 50, 50, 255, gih1
FLpanel_end
FLrun

gal init 0

instr 1

kenv linsegr 0,.5,.7,.5,.5,.2,0
ifreq cpsmidi
a1 poscil 0dbfs*kenv, ifreq, 1
gal = gal+a1

endin

instr 2

outs gal, gal
ktrig metro 25 ;refresh 25 times per second
kval max_k gal, ktrig, 1
FLsetVal ktrig, kval, gih1
gal = 0

endin

</CsInstruments>
<CsScore>
f1 0 1024 10 1

i2 0 3600
f0 3600

e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Written by Gabriel Maldonado.

Nouveau dans Csound 5. (N'était disponible auparavant que dans CsoundAV)

Modifié pour être conforme à la documentation dans la version 5.15.

# maxarray

maxarray — Retourne la valeur maximale dans un tableau.

## Description

L'opcode *maxarray* retourne la valeur maximale dans un tableau de taux-k, et éventuellement son indice.

## Syntaxe

```
kmax [,kindx] maxarray karray
```

## Exécution

*kmax* -- variable pour le résultat.

*kindx* -- position (indice) du résultat dans le tableau.

*karray* -- tableau à lire.

## Exemples

Voici un exemple de l'opcode maxarray. Il utilise le fichier *maxarray.csd* [examples/maxarray.csd].

### Exemple 541. Exemple de l'opcode maxarray.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n
</CsOptions>
<CsInstruments>
kmps = 32
;example by joachim heintz

      seed          0

instr 1
;create an array with 10 elements
kArr[] init      10
;fill in random numbers and print them out
kIndx      =      0
  until kIndx == 10 do
    kNum      random      -100, 100
    kArr[kIndx] =      kNum
    printf      "kArr[%d] = %10f\n", kIndx+1, kIndx, kNum
    kIndx      +=      1
  od
;investigate maximum number and print it out
kMax, kMaxIndx maxarray kArr
  printf      "Maximum of kArr = %f at index %d\n", kIndx+1, kMax, kMaxIndx
  turnoff
endin
```

```
</CsInstruments>
<CsScore>
i1 0 0.1
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*minarray, sumarray, scalearray,*

## Crédits

Auteur : John ffitch  
Octobre 2011

Nouveau dans la version 5.14 de Csound.

Révisé dans la version 6.00 de Csound pour fonctionner avec les tableau multidimensionnels.

# mclock

mclock — Envoie un message MIDI CLOCK.

## Description

Envoie un message MIDI CLOCK.

## Syntaxe

```
mclock ifreq
```

## Initialisation

*ifreq* -- taux de fréquence en Hz du message d'horloge

## Exécution

Envoie un message MIDI CLOCK (0xF8) chaque  $1/\textit{ifreq}$  secondes. Ainsi *ifreq* est le taux de fréquence en Hz du message d'horloge.

## Exemples

Voici un exemple de l'opcode mclock. Il utilise le fichier *mclock.csd* [examples/mclock.csd].

### Exemple 542. Exemple de l'opcode mclock.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-Q0    ;;midi out
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;let csound synchronize a sequencer

mclock 24

endin

</CsInstruments>
<CsScore>

i 1 0 30
e
</CsScore>
```

`</CsoundSynthesizer>`

## Voir aussi

*mrtmsg*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

# mdelay

mdelay — Un opcode de délai MIDI.

## Description

Un opcode de délai MIDI.

## Syntaxe

```
mdelay kstatus, kchan, kd1, kd2, kdelay
```

## Exécution

*kstatus* -- octet d'état du message MIDI message à retarder.

*kchan* -- canal MIDI (1-16)

*kd1* -- premier octet de donnée MIDI

*kd2* -- deuxième octet de donnée MIDI

*kdelay* -- délai en secondes

Chaque fois que *kstatus* est différent zéro, *mdelay* envoie un message MIDI sur le port de sortie MIDI après *kdelay* secondes. Cet opcode est utile pour implémenter des délais MIDI. Il peut y avoir plusieurs instances de *mdelay* dans le même instrument avec des valeurs d'argument différentes, si bien que l'on peut implémenter des echos MIDI complexes et colorés. De plus, on peut changer la durée du retard au taux-k.

## Exemples

Voici un exemple de l'opcode *mdelay*. Il utilise le fichier *mdelay.csd* [examples/mdelay.csd].

### Exemple 543. Exemple de l'opcode *mdelay*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc     -d         -M0   -Q0
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

instr 1  ;Triggered by MIDI notes on channel 1
```



```

kstatus init 0
ifund notnum
ivel veloc

noteondur 1, ifund, ivel, 1

kstatus = kstatus + 1

idel1 = .2
idel2 = .4
idel3 = .6
idel4 = .8

;make four delay lines

mdelay kstatus,1,ifund+2, ivel,idel1
mdelay kstatus,1,ifund+4, ivel,idel2
mdelay kstatus,1,ifund+6, ivel,idel3
mdelay kstatus,1,ifund+8, ivel,idel4

endin

</CsInstruments>
<CsScore>
; Dummy ftable
f 0 60
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Gabriel Maldonado  
 Italie  
 Novembre 1998

Nouveau dans la version 3.492 de Csound

# median

median — Un filtre médian, un filtre RIF passe-bas variant.

## Description

Implémentation d'un filtre méian.

## Syntaxe

```
ares median asig, ksize, imaxsize [, iskip]
```

## Initialisation

*imaxsize* -- la taille maximale de la fenêtre utilisée pour sélectionner les données.

*iskip* -- disposition initiale de l'espace de données interne. Une valeur nulle efface l'espace ; une valeur non nulle provoque le maintien de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal d'entrée à filtrer.

*ksize* -- taille de la fenêtre sur laquelle l'entrée est filtrée. Elle ne doit pas dépasser la taille de fenêtre maximale ; sinon elle est tronquée.

*median* est un simple filtre qui retourne la valeur médiane des *ksize* dernières valeurs. Il a une action passe-bas. L'efficacité varie en fonction inverse de la taille de la fenêtre.

## Exemples

Voici un exemple de l'opcode median. Il utilise le fichier *median.csd* [examples/median.csd].

### Exemple 544. Exemple de l'opcode median.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>

<CsInstruments>
sr = 44100
kr = 441
ksmps = 100
nchnls = 1

instr 1
  a1 oscil 30000, 10, 1
  a2 median a1, 5, 8
  out a2
endin
</CsInstruments>
```

```
<CsScore>
f1 0 4096 10 1
i 1 0 0.1
e

</CsScore>

</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch  
Université de Bath  
Mai 2010

Nouveau dans la version 5.13 de Csound.

# mediank

mediank — Un filtre médian, un filtre RIF passe-bas variant.

## Description

Implémentation d'un filtre méian.

## Syntaxe

```
kres mediank kin, ksize, imaxsize [, iskip]
```

## Initialisation

*imaxsize* -- la taille maximale de la fenêtre utilisée pour sélectionner les données.

*iskip* -- disposition initiale de l'espace de données interne. Une valeur nulle efface l'espace ; une valeur non nulle provoque le maintien de l'information précédente. La valeur par défaut est 0.

## Exécution

*kin* -- valeur de taux-k à filtrer.

*ksize* -- taille de la fenêtre sur laquelle l'entrée est filtrée. Elle ne doit pas dépasser la taille de fenêtre maximale ; sinon elle est tronquée.

*mediank* est un simple filtre qui retourne la valeur médiane des *ksize* dernières valeurs. Il a une action passe-bas. L'efficacité varie en fonction inverse de la taille de la fenêtre.

## Exemples

Voici un exemple de l'opcode mediank. Il utilise le fichier *mediank.csd* [examples/mediank.csd].

### Exemple 545. Exemple de l'opcode mediank.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-n
</CsOptions>
<CsInstruments>
sr = 44100
kr = 147
ksmps = 300
nchnls = 1

instr 1
  k1 oscil 100, 10, 1
  k2 mediank k1, 5, 8
  printk 0, k2
endin
```

```
</CsInstruments>

<CsScore>
f1 0 4096 10 1
i 1 0 1
e

</CsScore>

</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch  
Université de Bath  
Mai 2010

Nouveau dans la version 5.13 de Csound.

# metro

metro — Métronome déclencheur.

## Description

Génère un signal métronomique à utiliser dans toutes les circonstances pour lesquelles un déclencheur isochrone est nécessaire.

## Syntaxe

```
ktrig metro kfreq [, initphase]
```

## Initialisation

*initphase* -- valeur de phase initiale (entre 0 et 1)

## Exécution

*ktrig* -- signal déclencheur en sortie

*kfreq* -- fréquence des impulsions de déclenchement en Hz

*metro* est un opcode simple qui retourne une séquence d'impulsions isochrones (valeurs 1) chaque  $1/kfreq$  secondes. On peut utiliser les signaux déclencheurs en toute occasion, principalement pour temporiser des structures de composition algorithmique en temps réel.



### Note

*metro* produit un signal déclencheur (égal à 1) lorsque sa phase vaut exactement 0 ou 1. Si l'on veut ignorer le déclencheur initial, il faut utiliser une très petite valeur de phase initiale comme 0.00000001.

## Exemples

Voici un exemple de l'opcode *metro*. Il utilise le fichier *metro.csd* [examples/metro.csd]

### Exemple 546. Exemple de l'opcode *metro*.

```
<CsoundSynthesizer>
<CsOptions>
-odac -B441 -b441
</CsOptions>
<CsInstruments>

sr      =      44100
kr      =      100
ksmps   =      441
nchnls  =      2

        instr    1
ktrig metro 0.2
```

```
printk2 ktrig
endin

</CsInstruments>
<CsScore>
i 1 0 20

</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode metro. Il utilise le fichier *metro-2.csd* [examples/metro-2.csd]

### Exemple 547. Un autre exemple de l'opcode metro

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o metro-2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kpch    random    1,20    ;produce values at k-rate
ktrig    metro    10    ;trigger 10 times per second
kval    samphold kpch, ktrig    ;change value whenever ktrig = 1
asig    buzz    1, 220, kval, 1;harmonics
        outs      asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 4096 10 1 ; sine

i 1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

Plus d'information sur cet opcode dans les Floss Manuals : <http://write.flossmanuals.net/csound/c-control-structures/>

## Crédits

Ecrit par Gabriel Maldonado.

Premier exemple écrit par Andrés Cabrera.

Nouveau dans Csound 5. (N'était auparavant disponible que dans CsoundAV).

# metro2

metro2 — Métronome déclencheur avec swing et accents.

## Description

Génère un signal métronomique avec swing contrôlable.

## Syntaxe

```
ktrig metro2 kfreq, kswing [, iamp, initphase]
```

## Initialisation

*iamp* -- amplitude du clic hors pulsation

*initphase* -- valeur de phase initiale (entre 0 et 1)

## Exécution

*ktrig* -- signal déclencheur en sortie

*kfreq* -- fréquence des impulsions de déclenchement en Hz

*kswing* - valeur (comprise entre 0 et 1)

*metro2* est une version modifiée de l'opcode 'classique' *metro* avec swing additionnel. Les clics swingués peuvent être modulés au taux-k. De même, les clics swingués peuvent avoir leur propre valeur d'amplitude fixée par *iamp*.



### Note

*metro2* tout comme *metro* produit un signal déclencheur (égal à 1) lorsque sa phase vaut exactement 0 ou 1. Si l'on veut ignorer le déclencheur initial, il faut utiliser une très petite valeur de phase initiale comme 0.00000001.

## Exemples

Voici un exemple de l'opcode *metro2*. Il utilise le fichier *metro2.csd* [examples/metro2.csd]

### Exemple 548. Exemple de l'opcode *metro2*.

```
<CsoundSynthesizer>
<CsOptions>
-odac ; for RT audio
</CsOptions>
    <CsInstruments>

    sr = 44100
    kmps = 16
    nchnls = 2
    0dbfs = 1.0

    gal init 0 ; delay aux
```



```

instr 1 ; main triggering instrument
kndx init 0
kndx2 init -1
; the clicks are 1/16th notes @ 137 BPM
kT metro2 (4*137/60), p4, -1
if (kT == 0) goto Halt
k1 table kndx, 1, 0, 0, 1
kndx += 1
if (k1 == 0) goto Next
event "i", 2, 0, 0.35, k1
Next:
; positive amplitude values of down-beat clicks
; are used to trigger kick (instr 3)
if (kT < 1) goto Halt
kndx2 += 1
kndx2 wrap kndx2, 0, 2
if kndx2 != 0 goto Halt
event "i", 3, 0, 0.2
Halt:
endin

instr 2 ; simple subtractive bass
kAE linsegr 0, 0.005, 1, p3/2, .7, .04, 0, .04, 0
kFE linsegr 1, 0.005, 2, p3/2, .7, .04, .1, .04, .1
ifr = cpspch(p4)
a1 vco2 1, ifr
a2 vco2 1, ifr * 1.005
a3 vco2 1, ifr * 0.993
aM = (a1+a2+a3)/3
aM moogvcf aM*kAE, 1000 * kFE, 0.5
outs aM, aM
gal += aM * 0.25
endin

instr 3 ; simple techno kick
k1 linseg 200, p3, 10
k2 linseg 0,0.001,1,0.25,0
a1 oscil 0.3*k2, k1
outs a1,a1
endin

instr 99 ; feedback delay for bass
a1 delayr 0.5
ab deltap 0.33
delayw gal + ab*0.3
outs ab,ab
gal = 0
endin

</CsInstruments>
<CsScore>
t 0 137
; ftable1 is the bass sequence to played with various swings
f1 0 16 -2 6.00 0 0 7.00 0 0 6.00 0 6.00 0 0 7.00 0 7.01 6.00 0
; 4 measure pattern of different swinging
i1 0 16 0.5
i1 + . 0.65
i1 + . 0.4
i99 0 46
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*metro*

## Crédits

Auteur : Gleb Rogozinsky;  
Saint-Petersbourg, octobre 2019

Nouveau dans la version 6.14 de Csound.

# mfb

mfb — Banc de filtre étalonnés en mels pour les amplitudes spectrales.

## Description

Applique un banc de filtres étalonnés en mels à un tableau contenant une suite de bins d'amplitude.

## Syntaxe

```
kout[] mfb kin[], klow, khigh, ibands
```

## Initialisation

*ibands* -- nombres de bandes dans le banc de filtres en mels. Détermine la taille du tableau de sortie.

## Exécution

*kout[]* -- tableau de sortie contenant les valeurs des bandes du banc de filtres en mels.

*kin[]* -- tableau contenant les magnitudes des bins en entrée.

*klow* -- fréquence la plus basse (arrondie à la fréquence centrale de bin la plus proche.

*khigh* -- fréquence la plus haute (arrondie à la fréquence centrale de bin la plus proche).

## Exemples

Voici un exemple de l'opcode mfb. Il utilise le fichier *mfb.csd* [examples/mfb.csd].

### Exemple 549. Exemple de l'opcode mfb.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
ksmps = 64
instr 1

  ifftsize init 1024
  ibins init ifftsize/2
  kIn[] init ifftsize
  kcnt init 0
  p3 = filelen("fox.wav")
  asig diskin "fox.wav"
  kIn shiftin asig
  kcnt += ksmps
  if kcnt == ifftsize then
    kFFT[] = rfft(kIn)
    kPows[] = pows(kFFT)
    kMFB[] = log(mfb(kPows, 300, 8000, 32))
```

```
kmfcc[] = dct(kMFB)
kcnt = 0
kfb = 0
while kfb < 32 do
  printf("mfcc[%d] = %.3f \n", kfb+1, kfb, kmfcc[kfb])
  kfb += 1
od
endif

endin
</CsInstruments>
<CsScore>
i1 0 1
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

Vectorial opcodes, array opcodes, *dct*

## Crédits

Auteur : Victor Lazzarini  
NUI Maynooth  
2016

Nouveau dans la version 6.08

# midglobal

**midglobal** — Un opcode que l'on peut utiliser pour implémenter un orchestre midi distant. Cet opcode envoie les évènements midi à toutes les machines impliquées dans le concert à distance.

## Description

Avec les opcodes *midremot* et *midglobal*, il est possible d'exécuter des instruments sur des machines distantes et de les contrôler depuis une machine maître. Les opcodes distants sont implémentés suivant le modèle maître/client. Toutes les machines concernées contiennent le même orchestre mais seule la machine maître possède l'information de la partition midi. Durant l'exécution, la machine maître envoie les évènements midi aux clients. L'opcode *midglobal* envoie les évènements à toutes les machines impliquées dans le concert à distance. Ces machines sont déterminées par les définitions *midremot* placées avant la commande *midglobal*. Pour envoyer des évènements à une seule machine on utilise *midremot*.

## Syntaxe

```
midglobal isource, instrnum [,instrnum...]
```

## Initialisation

*isource* -- une chaîne représentant l'ordinateur hôte (par exemple 192.168.0.100). C'est l'hôte source qui génère les évènements pour le ou les instruments donnés et les envoie à toute les machines impliquées dans le concert à distance.

*instrnum* -- liste des numéros des instruments qui seront joués sur les machines destinataires.

## Exemples

Voir l'entrée *midremot* pour un exemple d'utilisation.

## Voir aussi

*insglobal*, *insremot*, *midremot*, *remoteport*

## Crédits

Auteur : Simon Champijer  
2006

Nouveau dans la version 5.03

# midiarp

midiarp — Génère des arpèges basés sur les notes MIDI tenues.

## Description

*midiarp* construit des arpèges basés sur les notes MIDI tenues. L'opcode produit des notes sous la forme de numéros de note MIDI, et un signal métronomique qui peut être utilisé pour séquencer les notes. On peut choisir la vitesse à laquelle les notes sont générées et aussi choisir parmi un ensemble de modèles d'arpèges.

## Syntaxe

```
kMidiNoteNum, kCountermidiarp kRate[, kMode]
```

## Initialisation

*kRate* -- fixe la vitesse à laquelle les nouvelles notes sont générées et contrôle la fréquence du signal d'impulsions métronomiques.

*kMode* -- Facultatif. Fixe le type d'arpège. 0 pour ascendant et descendant, 1 pour ascendant, 2 pour descendant, 3 pour aléatoire. 0 est la valeur par défaut.

## Exécution

*kMidiNoteNum* -- le numéro de note MIDI courant dans l'arpège.

*kCounter* -- une impulsion métronomique qui peut être utilisée pour déclencher l'exécution des notes de l'arpège. Ce signal produit un 1 suivi de 0 à chaque cycle. La fréquence est fixée par le paramètre *kRate* en entrée.



### Note

Il est important que l'instrument contenant l'opcode *midiarp* ne soit pas activé en continu à chaque nouvelle note MIDI. Pour éviter ceci, il faut utiliser l'opcode *massign* comme dans l'exemple ci-dessous.

## Exemple

Cet exemple montre l'utilisation de *midiarp* pour déclencher des arpèges réalisés sur un second instrument. L'instrument 100 reçoit des notes MIDI en entrée et déclenche l'action de l'instrument 200. *massign* est utilisé pour éviter que l'instrument 100 ne soit activé chaque fois qu'une nouvelle note MIDI est fournie. Cet exemple utilise le fichier *midiarp.csd* [examples/midiarp.csd].

### Exemple 550. Exemple de l'opcode *midiarp*

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac -d      -m0d      -M0  -+rtmidi=virtual ;;;RT audio I/O with MIDI in
```

```

; For Non-realtime ouput leave only the line below:
; -o midiin.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

massign 1, -1; prevent triggering of instrument with MIDI

instr 100
kMode = 3
kTempo = 6
kNote, kCounter midiarp kTempo

kFilterFreq oscil 2000, .05
; if kCounter is 1 trigger instrument 2 to play
if kCounter==1 then
  event "i", 200, 0, 2, kNote, kFilterFreq+2200
endif

endin

instr 200
kEnv expon .4, p3, .001
aOut vco2 kEnv, cpsmidinn(p4)*2 ;convert note number to cps
aFilter moogladder aOut, p5, 0
outs aFilter, aFilter
endin

</CsInstruments>
<CsScore>
i100 0 1000
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Rory Walsh  
2017

# midic14

**midic14** — Permet un signal MIDI sur 14 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Description

Permet un signal MIDI sur 14 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Syntaxe

```
idest midic14 ictlno1, ictlno2, imin, imax [, ifn]  
kdest midic14 ictlno1, ictlno2, kmin, kmax [, ifn]
```

## Initialisation

*idest* -- signal de sortie

*ictlno1* -- numéro de contrôleur pour l'octet de poids fort (0-127)

*ictlno2* -- numéro de contrôleur pour l'octet de poids faible (0-127)

*imin* -- valeur décimale minimale de sortie définie par l'utilisateur

*imax* -- valeur décimale maximale de sortie définie par l'utilisateur

*ifn* (facultatif) -- la table à lire lorsque l'indexation est requise. La table doit être normalisée. La sortie est mise à l'échelle entre les valeurs *imax* et *imin*.

## Exécution

*kdest* -- signal de sortie

*kmin* -- valeur décimale minimale de sortie définie par l'utilisateur

*kmax* -- valeur décimale maximale de sortie définie par l'utilisateur

*midic14* (contrôle MIDI sur 14 bit au taux-i et au taux-k) permet un signal MIDI sur 14 bit en nombres décimaux mis à l'échelle entre des limites minimale et maximale. Les valeurs minimale et maximale peuvent être variées au taux-k. Il peut utiliser en option une indexation de table interpolée. Il nécessite deux contrôleurs MIDI en entrée.



### Note

Veuillez noter que la famille des opcodes *midic* est prévue pour des événements MIDI déclenchés, et ne nécessite pas de numéro de canal car ils vont répondre au même canal que celui qui a déclenché l'instrument (voir *massign*). Cependant ils vont planter s'ils sont appelés depuis un événement de partition.

## Voir aussi

*ctrl7, ctrl14, ctrl21, initc7, initc14, initc21, midic7, midic21*



## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué le bon intervalle pour le numéro de contrôleur.

# midic21

**midic21** — Permet un signal MIDI sur 21 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Description

Permet un signal MIDI sur 21 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Syntaxe

```
idest midic21 ictlno1, ictlno2, ictlno3, imin, imax [, ifn]  
kdest midic21 ictlno1, ictlno2, ictlno3, kmin, kmax [, ifn]
```

## Initialisation

*idest* -- signal de sortie

*ictln1o* -- numéro de contrôleur pour l'octet de poids fort (0-127)

*ictlno2* -- numéro de contrôleur pour l'octet de poids moyen (0-127)

*ictlno3* -- numéro de contrôleur pour l'octet de poids faible (0-127)

*imin* -- valeur décimale minimale de sortie définie par l'utilisateur

*imax* -- valeur décimale maximale de sortie définie par l'utilisateur

*ifn* (facultatif) -- la table à lire lorsque l'indexation est requise. La table doit être normalisée. La sortie est mise à l'échelle entre les valeurs *imax* et *imin*.

## Exécution

*kdest* -- signal de sortie

*kmin* -- valeur décimale minimale de sortie définie par l'utilisateur

*kmax* -- valeur décimale maximale de sortie définie par l'utilisateur

*midic21* (contrôle MIDI sur 21 bit au taux-i et au taux-k) permet un signal MIDI sur 21 bit en nombres décimaux mis à l'échelle entre des limites minimale et maximale. Les valeurs minimale et maximale peuvent être variées au taux-k. Il peut utiliser en option une indexation de table interpolée. Il nécessite trois contrôleurs MIDI en entrée.



### Note

Veuillez noter que la famille des opcodes *midic* est prévue pour des événements MIDI déclenchés, et ne nécessite pas de numéro de canal car ils vont répondre au même canal que celui qui a déclenché l'instrument (voir *massign*). Cependant ils vont planter s'ils sont appelés depuis un événement de partition.

## Voir aussi

*ctrl7, ctrl14, ctrl21, initc7, initc14, initc21, midic7, midic14*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué le bon intervalle pour le numéro de contrôleur.

# midic7

*midic7* — Permet un signal MIDI sur 7 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Description

Permet un signal MIDI sur 7 bit en nombres décimaux selon une échelle entre des limites minimale et maximale.

## Syntaxe

```
idest midic7 ictlno, imin, imax [, ifn]
```

```
kdest midic7 ictlno, kmin, kmax [, ifn]
```

## Initialisation

*idest* -- signal de sortie

*ictlno* -- numéro de contrôleur MIDI (0-127)

*imin* -- valeur décimale minimale de sortie définie par l'utilisateur

*imax* -- valeur décimale maximale de sortie définie par l'utilisateur

*ifn* (facultatif) -- la table à lire lorsque l'indexation est requise. La table doit être normalisée. La sortie est mise à l'échelle entre les valeurs *imin* et *imax*.

## Exécution

*kdest* -- signal de sortie

*kmin* -- valeur décimale minimale de sortie définie par l'utilisateur

*kmax* -- valeur décimale maximale de sortie définie par l'utilisateur

*midic7* (contrôle MIDI sur 7 bit au taux-i et au taux-k) permet un signal MIDI sur 7 bit en nombres décimaux mis à l'échelle entre des limites minimale et maximale. Il permet également en option une indexation de table sans interpolation. Dans *midic7* les valeurs minimale et maximale peuvent varier au taux-k.



### Note

Veuillez noter que la famille des opcodes *midic* est prévue pour des événements MIDI déclenchés, et ne nécessite pas de numéro de canal car ils vont répondre au même canal que celui qui a déclenché l'instrument (voir *massign*). Cependant ils vont planter s'ils sont appelés depuis un événement de partition.

## Exemples

Voici un exemple de l'opcode *midic7*. Il utilise le fichier *midic7.csd* [exemples/midic7.csd]

## Exemple 551. Exemple de l'opcode `midic7`.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0      ;;realtime audio out and realtime midi in
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o midic7.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
; This example expects MIDI controller input on channel 1
; run, play a note and move your midi controller 7 to see result

imax = 1
imin = 0
ichan = 1
ictlno= 7 ; = midi volume

kamp midic7 ictlno, imin, imax
  printk2 kamp
asig oscili kamp, 220, 1
  outs asig, asig

endin
</CsInstruments>
<CsScore>
; no score events allowed
f 0 20 ;20 sec. for real-time MIDI events
f 1 0 4096 10 1 ;sine wave

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*ctrl7, ctrl14, ctrl21, initc7, initc14, initc21, midic14, midic21*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound

Merci à Rasmus Ekman pour avoir indiqué le bon intervalle pour le numéro de contrôleur.

# midichannelaftertouch

midichannelaftertouch — Retourne la valeur d'aftertouch d'un canal MIDI.

## Description

*midichannelaftertouch* est conçu pour simplifier l'écriture d'instruments que l'on peut utiliser de manière interchangeable avec une partition ou depuis l'entrée MIDI, et pour faciliter l'adaptation d'instruments écrits à l'origine pour une partition au fonctionnement à partir d'une entrée MIDI.

En général, il doit être possible d'écrire des définitions d'instrument qui fonctionnent de la même manière avec une partition et avec le MIDI, que ce soit un fichier MIDI ou une entrée MIDI en , sans utiliser d'instructions conditionnelles supplémentaires, et qui tirent pleinement avantage des messages de voix MIDI.

Noter que la liaison entre des instruments de Csound et les numéros de canal MIDI se fait en utilisant l'opcode *massign* lors d'une exécution en . Pour les exécutions de fichier MIDI, les numéros d'instruments sont liés par défaut au canal MIDI + 1, mais ces valeurs par défaut peuvent être modifiées par tout message de changement de programme dans le fichier.

## Syntaxe

```
midichannelaftertouch xchannelaftertouch [, ilow] [, ihigh]
```

## Initialisation

*ilow* (facultatif) -- valeur basse facultative après rééchellonnement, 0 par défaut.

*ihigh* (facultatif) -- valeur haute facultative après rééchellonnement, 127 par défaut.

## Exécution

*xchannelaftertouch* -- retourne l'aftertouch d'un canal MIDI durant l'activation MIDI, et reste inchangé dans les autres cas.

Si l'instrument a été activé par une entrée MIDI, l'opcode remplace la valeur de *xchannelaftertouch* par la valeur correspondante venant de l'entrée MIDI. Si l'instrument n'a *PAS* été activé par une entrée MIDI, la valeur de *xchannelaftertouch* reste inchangée.

Grâce à cela, les p-champs de la partition peuvent recevoir leur valeur de données MIDI en entrée durant l'activation MIDI, et de la partition dans les autres cas.



### Adaptation d'un instrument de Csound activé par partition.

Voir la section *Opcodes pour l'Interopérabilité MIDI/Partition* pour plus de détails sur l'adaptation d'instruments pilotés par partition au MIDI et vice-versa.

## Exemples

Voici un exemple de l'opcode *midichannelaftertouch*. Il utilise le fichier *midichannelaftertouch.csd* [examples/midichannelaftertouch.csd].

## Exemple 552. Exemple de l'opcode `midichannelaftertouch`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      -M1    ;;realtime audio out and midi in
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o midichannelaftertouch.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

midinoteoncps p4, p5    ;puts MIDI key translated to cycles per second into p4, and MIDI velocity into
kafter init 127        ;full open
midichannelaftertouch kafter
printk2 kafter         ;display the key value when it changes and when key is pressed

kvel = p5/127          ;scale midi velocity to 0-1
kenv madsr 0.5, 0.8, 0.8, 0.5 ;amplitude envelope multiplied by
ain pluck kenv*kvel, p4, p4, 2, 1 ;velocity value
asig moogvcf2 ain, 200+(kafter*40), .5 ;scale value of aftertouch and control filter frequency
      outs asig, asig      ;base freq of filter = 200 Hz

endin
</CsInstruments>
<CsScore>
f 0 30 ;runs 30 seconds
f 2 0 4096 10 1

i 1 0 2 440 100 ; play these notes from score as well
i 1 + 2 1440 100
e

</CsScore>
</CsSoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
i1 127.00000
i1 20.00000
i1 44.00000
```

## Voir aussi

*midicontrolchange, mididefault, midinoteoff, midinoteoncps, midinoteonkey, midinoteonoct, midinoteonpch, midipitchbend, midipolyaftertouch, midiprogramchange*

## Crédits

Auteur : Michael Gogins

Nouveau dans la version 4.20

# midichn

midichn — Retourne le numéro de canal MIDI duquel la note a été activée.

## Description

*midichn* retourne le numéro de canal MIDI (1 à 16) duquel la note a été activée. Dans le cas d'une note venant d'une partition, il retourne 0.

## Syntaxe

ichn **midichn**

## Initialisation

*ichn* -- numéro de canal. Si la note courante provient d'une partition, il prend la valeur zéro.

## Exemples

Voici un exemple simple de l'opcode *midichn*. Il utilise le fichier *midichn.csd* [examples/midichn.csd].

### Exemple 553. Exemple simple de l'opcode *midichn*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -M0 -+rtmidi=virtual ;; midi file input
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
;change channel on virtual midi keyboard
i1 midichn
  print i1

endin
</CsInstruments>
<CsScore>
f 0 20 ;runs for 20 seconds

e
</CsScore>
</CsoundSynthesizer>
```

Voici un exemple avancé de l'opcode *midichn*. Il utilise le fichier *midichn\_advanced.csd* [examples/midichn\_advanced.csd].



Ne pas oublier qu'il faut l'option *-F flag* lorsque l'on utilise un fichier MIDI externe comme « midichn\_advanced.mid ».

### Exemple 554. Un exemple avancé de l'opcode midichn.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -F midichn_advanced.mid ;;realtime audio out with MIDI file input
; For Non-realtime ouput leave only the line below:
; -o midichn_advanced.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

massign 1, 1 ; all channels use instr 1
massign 2, 1
massign 3, 1
massign 4, 1
massign 5, 1
massign 6, 1
massign 7, 1
massign 8, 1
massign 9, 1
massign 10, 1
massign 11, 1
massign 12, 1
massign 13, 1
massign 14, 1
massign 15, 1
massign 16, 1

gicnt = 0 ; note counter

instr 1

gicnt = gicnt + 1 ; update note counter
kcnt init gicnt ; copy to local variable
ichn midichn ; get channel number
istime times ; note-on time

if (ichn > 0.5) goto l2 ; MIDI note
printks "note %.0f (time = %.2f) was activated from the score\\n", \
3600, kcmt, istime
goto l1
l2:
printks "note %.0f (time = %.2f) was activated from channel %.0f\\n", \
3600, kcmt, istime, ichn
l1:

icps cpsmidi ; convert midi note to pitch
kenv madsr 0.1, 0, 0.8, 0.9
asig pluck kenv, icps, icps, 1, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
t 0 60 ;beats per minute

f 0 8 ;stay active for 8 seconds
```

```
f 1 0 4096 10 1 ;sine
```

```
e  
</CsScore>  
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
note 1 (time = 0.00) was activated from channel 1  
note 2 (time = 2.00) was activated from channel 4  
note 3 (time = 3.00) was activated from channel 2  
note 4 (time = 5.00) was activated from channel 3
```

## Voir aussi

*pgmassign*

## Crédits

Auteur : Istvan Varga  
Mai 2002

Nouveau dans la version 4.20

# midicontrolchange

midicontrolchange — Retourne la valeur d'un changement de contrôle MIDI.

## Description

*midicontrolchange* est conçu pour simplifier l'écriture d'instruments que l'on peut utiliser de manière interchangeable avec une partition ou depuis l'entrée MIDI, et pour faciliter l'adaptation d'instruments écrits à l'origine pour une partition au fonctionnement à partir d'une entrée MIDI.

En général, il doit être possible d'écrire des définitions d'instrument qui fonctionnent de la même manière avec une partition et avec le MIDI, que ce soit un fichier MIDI ou une entrée MIDI en , sans utiliser d'instructions conditionnelles supplémentaires, et qui tirent pleinement avantage des messages de voix MIDI.

Noter que la liaison entre des instruments de Csound et les numéros de canal MIDI se fait en utilisant l'opcode *massign* lors d'une exécution en . Pour les exécutions de fichier MIDI, les numéros d'instruments sont liés par défaut au canal MIDI + 1, mais ces valeurs par défaut peuvent être modifiées par tout message de changement de programme dans le fichier.

## Syntaxe

```
midicontrolchange xcontroller, xcontrollervalue [, ilow] [, ihigh]
```

## Initialisation

*ilow* (facultatif) -- valeur basse facultative après reéchellonnement, 0 par défaut.

*ihigh* (facultatif) -- valeur haute facultative après reéchellonnement, 127 par défaut.

## Exécution

*xcontroller* -- spécifie un numéro de contrôleur MIDI (0-127).

*xcontrollervalue* -- retourne la valeur du contrôleur MIDI durant l'activation MIDI, et reste inchangé dans les autres cas.

Si l'instrument a été activé par une entrée MIDI, l'opcode remplace les valeurs de *xcontroller* et de *xcontrollervalue* par les valeurs correspondantes venant de l'entrée MIDI. Si l'instrument n'a *PAS* été activé par une entrée MIDI, les valeurs de *xcontroller* et de *xcontrollervalue* restent inchangées.

Grâce à cela, les p-champs de la partition peuvent recevoir leur valeur de données MIDI en entrée durant l'activation MIDI, et de la partition dans les autres cas.



### Adaptation d'un instrument de Csound activé par partition.

Voir la section *Opcodes pour l'Interopérabilité MIDI/Partition* pour plus de détails sur l'adaptation d'instruments pilotés par partition au MIDI et vice-versa.

## Exemples

Voici un exemple de l'opcode *midicontrolchange*. Il utilise le fichier *midicontrolchange.csd* [exemples/midicontrolchange.csd]

**Exemple 555. Exemple de l'opcode midicontrolchange.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0 ;;;realtime audio out and midi in
;-iadc ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o midicontrolchange.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
; use slider of contr. 7 of virtual keyboard
kcont init 1 ; max. volume
midicontrolchange 7, kcont, 0, 1; use controller 7, scaled between 0 and 1
printk2 kcont ; Display the key value when it changes and key is pressed

kenv madsr 0.5, 0.8, 0.8, 0.5 ; envelope multiplied by
asig pluck kenv*kcont, 220, 220, 2, 1 ; value of controller 7
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 0 30
f 2 0 4096 10 1

i 1 10 2 ; play a note from score as well
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*midichannelaftertouch, mididefault, midinoteoff, midinoteoncps, midinoteonkey, midinoteonoct, midino-*  
*teonpch, midipitchbend, midipolyaftertouch, midiprogramchange*

## Crédits

Auteur : Michael Gogins

Nouveau dans la version 4.20

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# midictrl

midictrl — Donne la valeur actuelle (0-127) d'un contrôleur MIDI spécifié.

## Description

Donne la valeur actuelle (0-127) d'un contrôleur MIDI spécifié.

## Syntaxe

```
ival midictrl inum [, imin] [, imax]
```

```
kval midictrl inum [, imin] [, imax]
```

## Initialisation

*inum* -- numéro de contrôleur MIDI (0-127)

*imin*, *imax* -- Ajuste les limites minimale et maximales pour les valeurs obtenues.

## Exécution

Donne la valeur actuelle (0-127) d'un contrôleur MIDI spécifié.

## Avertissement

*midictrl* doit être utilisé seulement pour les notes déclenchées par MIDI, permettant la disponibilité d'un numéro de canal associé. Pour les notes activée depuis la partition, les événements de ligne, ou l'orchestre, il faut utiliser l'opcode *ctrl7* qui prend un numéro de canal explicite.

## Exemples

Voici un exemple de l'opcode midictrl. Il utilise le fichier *midictrl.csd* [examples/midictrl.csd]

### Exemple 556. Exemple de l'opcode midictrl.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0    ;;realtime audio out and realtime midi in
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o midictrl.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
```

```
icps cpsmidi
iamp ampmidi .5
ips midictrl 9, 10, 500 ;controller 9

kenv madsr 0.5, 0, 1, 0.5
asig pluck kenv, icps, ips, 2, 1 ;change tone color
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 2 0 4096 10 1 ;sine wave
; no score events allowed
f0 30 ;runs 30 seconds
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*aftouch, ampmidi, cpsmidi, cpsmidib, notnum, octmidi, octmidib, pchbend, pchmidi, pchmidib, veloc*

## Crédits

Auteur : Barry L. Vercoe - Mike Berry  
MIT - Mills  
Mai 1997

Merci à Rasmus Ekman pour avoir indiqué le bon intervalle pour le numéro de contrôleur.

# mididefault

mididefault — Change des valeurs en fonction de l'activation MIDI.

## Description

*mididefault* est conçu pour simplifier l'écriture d'instruments que l'on peut utiliser de manière interchangeable avec une partition ou depuis l'entrée MIDI, et pour faciliter l'adaptation d'instruments écrits à l'origine pour une partition au fonctionnement à partir d'une entrée MIDI.

En général, il doit être possible d'écrire des définitions d'instrument qui fonctionnent de la même manière avec une partition et avec le MIDI, que ce soit un fichier MIDI ou une entrée MIDI en , sans utiliser d'instructions conditionnelles supplémentaires, et qui tirent pleinement avantage des messages de voix MIDI.

Noter que la liaison entre des instruments de Csound et les numéros de canal MIDI se fait en utilisant l'opcode *massign* lors d'une exécution en . Pour les exécutions de fichier MIDI, les numéros d'instruments sont liés par défaut au canal MIDI + 1, mais ces valeurs par défaut peuvent être modifiées par tout message de changement de programme dans le fichier.

## Syntaxe

```
mididefault xdefault, xvalue
```

## Exécution

*xdefault* -- spécifie une valeur par défaut qui sera utilisée pendant l'activation MIDI.

*xvalue* -- remplacé par *xdefault* durant l'activation MIDI, reste inchangé dans les autres cas.

Si l'instrument a été activé par une entrée MIDI, l'opcode remplace la valeur de *xvalue* par celle de *xdefault*. Si l'instrument n'a *PAS* été activé par une entrée MIDI, la valeur de *xvalue* reste inchangée.

Grâce à cela, les p-champs de la partition peuvent recevoir une valeur par défaut durant l'activation MIDI, et une valeur de la partition dans les autres cas.



### Adaptation d'un instrument de Csound activé par partition.

Voir la section *Opcodes pour l'Interopérabilité MIDI/Partition* pour plus de détails sur l'adaptation d'instruments pilotés par partition au MIDI et vice-versa.

## Exemples

Voici un exemple de l'opcode *mididefault*. Il utilise le fichier *mididefault.csd* [examples/mididefault.csd].

### Exemple 557. Exemple de l'opcode *mididefault*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
```

```

<CsOptions>
; Select audio/midi flags here according to platform
-odac -M1 ;;;realtime audio out and midi in
;-iadc ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o mididefault.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

mididefault 60, p3 ;ensures that a MIDI-activated instrument will have a positive p3 field
midinoteoncps p4, p5 ;puts MIDI key translated to cycles per second into p4, and MIDI velocity into p5

kvel = p5/127 ;scale midi velocity to 0-1
kenv madsr 0.5, 0.8, 0.8, 0.5 ;amplitude envelope multiplied by
asig pluck kenv*kvel, p4, p4, 2, 1 ;velocity
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 0 30 ;runs for 30 seconds
f 2 0 4096 10 1

i 1 0 2 440 100 ; play these notes from score as well
i 1 + 2 1440 100
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*midichannelaftertouch, midicontrolchange, midinoteoff, midinoteoncps, midinoteonkey, midinoteonoct, midinoteonpch, midipitchbend, midipolyaftertouch, midiprogramchange*

## Crédits

Auteur : Michael Gogins

Nouveau dans la version 4.20



# midiiin

midiiin — Retourne un message MIDI générique reçu sur le port MIDI IN.

## Description

Retourne un message MIDI générique reçu sur le port MIDI IN.

## Syntaxe

```
kstatus, kchan, kdata1, kdata2 midiiin
```

## Exécution

*kstatus* -- le type de message MIDI. Peut être :

- 128 (note off)
- 144 (note on)
- 160 (aftertouch polyphonique)
- 176 (changement de contrôle)
- 192 (changement de programme)
- 208 (aftertouch de canal)
- 224 (pitch bend)
- 0 si aucun message MIDI n'est en attente dans le tampon MIDI IN

*kchan* -- canal MIDI (1-16)

*kdata1*, *kdata2* -- données dépendant du message

*midiiin* n'a pas d'arguments en entrée, car il lit implicitement le port MIDI. Il travaille au taux-k. Normalement (quand aucun message n'est en attente), *kstatus* vaut zéro. *kstatus* ne prend la valeur du type de message adéquat que lorsque des données MIDI sont présentes dans le tampon MIDI IN.



### Note

Il faut faire attention lorsque l'on utilise *midiiin* dans des instruments de faible numéro car une note MIDI démarrera des instances supplémentaires des instruments, ce qui provoquera des duplications d'évènement et un comportement étrange. Utiliser *massign* pour diriger les messages de note MIDI vers un instrument différent ou pour désactiver l'activation d'instruments à partir du MIDI.

## Exemples

Voici un exemple de l'opcode *midiiin*. Il utilise le fichier *midiiin.csd* [examples/midiiin.csd].

**Exemple 558. Exemple ce l'opcode midiin.**

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>

<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d          -M0  -+rtmidi=virtual ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o midiin.wav -W ;; for file output any platform
</CsOptions>

<CsInstruments>

sr          = 44100
ksmps      = 10
nchnls     = 1

; Example by schwaahed 2006

    massign      0, 130 ; make sure that all channels
    pgmassign    0, 130 ; and programs are assigned to test instr

instr 130

knotelength    init      0
knoteontime    init      0

kstatus, kchan, kdata1, kdata2          midiin

if (kstatus == 128) then
knoteofftime    times
knotelength     =    knoteofftime - knoteontime
printks "kstatus= %d, kchan = %d, \\tnote#   = %d, velocity = %d \\tNote OFF\\t%f %f\\n", 0, kstatus, kchan, kdata1, kdata2

elseif (kstatus == 144) then
knoteontime      times
printks "kstatus= %d, kchan = %d, \\tnote#   = %d, velocity = %d \\tNote ON\\t%f %f\\n", 0, kstatus, kchan, kdata1, kdata2

elseif (kstatus == 160) then
printks "kstatus= %d, kchan = %d, \\tkdata1 = %d, kdata2 = %d \\tPolyphonic Aftertouch\\n", 0, kstatus, kchan, kdata1, kdata2

elseif (kstatus == 176) then
printks "kstatus= %d, kchan = %d, \\t CC = %d, value = %d \\tControl Change\\n", 0, kstatus, kchan, kdata1, kdata2

elseif (kstatus == 192) then
printks "kstatus= %d, kchan = %d, \\tkdata1 = %d, kdata2 = %d \\tProgram Change\\n", 0, kstatus, kchan, kdata1, kdata2

elseif (kstatus == 208) then
printks "kstatus= %d, kchan = %d, \\tkdata1 = %d, kdata2 = %d \\tChannel Aftertouch\\n", 0, kstatus, kchan, kdata1, kdata2

elseif (kstatus == 224) then
printks "kstatus= %d, kchan = %d, \\t ( data1 , kdata2 ) = ( %d, %d )\\tPitch Bend\\n", 0, kstatus, kchan, kdata1, kdata2

endif

endin

</CsInstruments>

<CsScore>
```

```
i130 0 3600  
e  
</CsScore>  
  
</CsoundSynthesizer>
```

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1998

Nouveau dans le version 3.492 de Csound.

# midifilestatus

midifilestatus — Retourne l'état de la restitution du fichier MIDI en entrée.

## Description

Retourne l'état courant au taux-k de la restitution du fichier MIDI en entrée, 1 si la restitution est en cours, 0 si la fin du fichier a été atteinte.

## Syntaxe

```
ksig midifilestatus
```

## Crédits

Auteur : Victor Lazzarini  
Mars 2006  
Nouveau dans Csound6

# midinoteoff

midinoteoff — Retourne une valeur de note off MIDI.

## Description

*midinoteoff* est conçu pour simplifier l'écriture d'instruments que l'on peut utiliser de manière interchangeable avec une partition ou depuis l'entrée MIDI, et pour faciliter l'adaptation d'instruments écrits à l'origine pour une partition au fonctionnement à partir d'une entrée MIDI.

En général, il doit être possible d'écrire des définitions d'instrument qui fonctionnent de la même manière avec une partition et avec le MIDI, que ce soit un fichier MIDI ou une entrée MIDI en , sans utiliser d'instructions conditionnelles supplémentaires, et qui tirent pleinement avantage des messages de voix MIDI.

Noter que la liaison entre des instruments de Csound et les numéros de canal MIDI se fait en utilisant l'opcode *massign* lors d'une exécution en . Pour les exécutions de fichier MIDI, les numéros d'instruments sont liés par défaut au canal MIDI + 1, mais ces valeurs par défaut peuvent être modifiées par tout message de changement de programme dans le fichier.

## Syntaxe

**midinoteoff** *xkey*, *xvelocity*

## Exécution

*xkey* -- retourne une touche MIDI durant l'activation MIDI, et reste inchangé dans les autres cas.

*xvelocity* -- retourne une vélocité MIDI durant l'activation MIDI, et reste inchangé dans les autres cas.

Si l'instrument a été activé par une entrée MIDI, l'opcode remplace les valeurs de *xkey* et de *xvelocity* par les valeurs correspondantes venant de l'entrée MIDI. Si l'instrument n'a *PAS* été activé par une entrée MIDI, les valeurs de *xkey* et de *xvelocity* restent inchangées.

Grâce à cela, les p-champs de la partition peuvent recevoir leur valeur de données MIDI en entrée durant l'activation MIDI, et de la partition dans les autres cas.



### Adaptation d'un instrument de Csound activé par partition.

Voir la section *Opcodes pour l'Interopérabilité MIDI/Partition* pour plus de détails sur l'adaptation d'instruments pilotés par partition au MIDI et vice-versa.

## Exemples

Voici un exemple de l'opcode *midinoteoff*. Il utilise le fichier *midinoteoff.csd* [examples/midinoteoff.csd].

### Exemple 559. Exemple de l'opcode midinoteoff.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      -M1 -Q1   ;;realtime audio out and midi in and midi out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o midinoteoff.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;uses external midi device

kkey init 0      ;initialize key number
kvel init 0      ;initialize velocity
midinoteoff kkey,kvel ;MIDI noteoff value
printk2 kvel     ;display noteoff value
midion 1, kkey, kvel ;sent note to external device

endin
</CsInstruments>
<CsScore>
f 0 30 ;runs for 30 seconds

i 1 0 2 62 ; play these notes from score as well
i 1 + 2 65
e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

i1      60.00000
i1      76.00000

```

## Voir aussi

*midichannelaftertouch, midicontrolchange, mididefault, midinoteoncps, midinoteonkey, midinoteonoct, midinoteonpch, midipitchbend, midipolyaftertouch, midiprogramchange*

## Crédits

Auteur : Michael Gogins

Nouveau dans la version 4.20

# midinoteoncps

midinoteoncps — Retourne un numéro de note MIDI traduit en fréquence (Hz).

## Description

*midinoteoncps* est conçu pour simplifier l'écriture d'instruments que l'on peut utiliser de manière interchangeable avec une partition ou depuis l'entrée MIDI, et pour faciliter l'adaptation d'instruments écrits à l'origine pour une partition au fonctionnement à partir d'une entrée MIDI.

En général, il doit être possible d'écrire des définitions d'instrument qui fonctionnent de la même manière avec une partition et avec le MIDI, que ce soit un fichier MIDI ou une entrée MIDI en , sans utiliser d'instructions conditionnelles supplémentaires, et qui tirent pleinement avantage des messages de voix MIDI.

Noter que la liaison entre des instruments de Csound et les numéros de canal MIDI se fait en utilisant l'opcode *massign* lors d'une exécution en . Pour les exécutions de fichier MIDI, les numéros d'instruments sont liés par défaut au canal MIDI + 1, mais ces valeurs par défaut peuvent être modifiées par tout message de changement de programme dans le fichier.

## Syntaxe

**midinoteoncps** *xcps*, *xvelocity*

## Exécution

*xcps* -- retourne une touche MIDI traduite en Hz durant l'activation MIDI, et reste inchangé dans les autres cas.

*xvelocity* -- retourne une vélocité MIDI durant l'activation MIDI, et reste inchangé dans les autres cas.

Si l'instrument a été activé par une entrée MIDI, l'opcode remplace les valeurs de *xcps* et de *xvelocity* par les valeurs correspondantes venant de l'entrée MIDI. Si l'instrument n'a *PAS* été activé par une entrée MIDI, les valeurs de *xcps* et de *xvelocity* restent inchangées.

Grâce à cela, les p-champs de la partition peuvent recevoir leur valeur de données MIDI en entrée durant l'activation MIDI, et de la partition dans les autres cas.



### Adaptation d'un instrument de Csound activé par partition.

Voir la section *Opcodes pour l'Interopérabilité MIDI/Partition* pour plus de détails sur l'adaptation d'instruments pilotés par partition au MIDI et vice-versa.

## Exemples

Voici un exemple de l'opcode *midinoteoncps*. Il utilise le fichier *midinoteoncps.csd* [exemples/midinoteoncps.csd].

### Exemple 560. Exemple de l'opcode *midinoteoncps*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      -M1   ;;realtime audio out and midi in
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o midinoteoncps.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

midinoteoncps p4, p5   ;puts MIDI key translated to cycles per second into p4, and MIDI velocity into

print p4              ;display the key value when it changes and when key is pressed
kvel = p5/127          ;scale midi velocity to 0-1
kenv madsr 0.5, 0.8, 0.8, 0.5   ;amplitude envelope multiplied by
asig pluck kenv*kvel, p4, p4, 2, 1 ;velocity value
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 0 30 ;runs for 30 seconds
f 2 0 4096 10 1

i 1 0 2 440 100 ; play these notes from score as well
i 1 + 2 1440 100
e

</CsScore>
</CsoundSynthesizer>

```

La sortie contiendra des lignes comme celles-ci :

```

i1 261.62561
i1 440.00006

```

## Voir aussi

*midichannelaftertouch, midicontrolchange, mididefault, midinoteoff, midinoteonkey, midinoteonoct, midinoteonpch, midipitchbend, midipolyaftertouch, midiprogramchange*

## Crédits

Auteur : Michael Gogins

Nouveau dans la version 4.20



# midinoteonkey

midinoteonkey — Retourne un numéro de note MIDI.

## Description

*midinoteonkey* est conçu pour simplifier l'écriture d'instruments que l'on peut utiliser de manière interchangeable avec une partition ou depuis l'entrée MIDI, et pour faciliter l'adaptation d'instruments écrits à l'origine pour une partition au fonctionnement à partir d'une entrée MIDI.

En général, il doit être possible d'écrire des définitions d'instrument qui fonctionnent de la même manière avec une partition et avec le MIDI, que ce soit un fichier MIDI ou une entrée MIDI en , sans utiliser d'instructions conditionnelles supplémentaires, et qui tirent pleinement avantage des messages de voix MIDI.

Noter que la liaison entre des instruments de Csound et les numéros de canal MIDI se fait en utilisant l'opcode *massign* lors d'une exécution en . Pour les exécutions de fichier MIDI, les numéros d'instruments sont liés par défaut au canal MIDI + 1, mais ces valeurs par défaut peuvent être modifiées par tout message de changement de programme dans le fichier.

## Syntaxe

```
midinoteonkey xkey, xvelocity
```

## Exécution

*xkey* -- retourne une touche MIDI durant l'activation MIDI, et reste inchangé dans les autres cas.

*xvelocity* -- retourne une vélocité MIDI durant l'activation MIDI, et reste inchangé dans les autres cas.

Si l'instrument a été activé par une entrée MIDI, l'opcode remplace les valeurs de *xkey* et de *xvelocity* par les valeurs correspondantes venant de l'entrée MIDI. Si l'instrument n'a *PAS* été activé par une entrée MIDI, les valeurs de *xkey* et de *xvelocity* restent inchangées.

Grâce à cela, les p-champs de la partition peuvent recevoir leur valeur de données MIDI en entrée durant l'activation MIDI, et de la partition dans les autres cas.



### Adaptation d'un instrument de Csound activé par partition.

Voir la section *Opcodes pour l'Interopérabilité MIDI/Partition* pour plus de détails sur l'adaptation d'instruments pilotés par partition au MIDI et vice-versa.

## Exemples

Voici un exemple de l'opcode *midinoteonkey*. Il utilise le fichier *midinoteonkey.csd* [exemples/midinoteonkey.csd].

### Exemple 561. Exaemple de l'opcode midinoteonkey.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      -M1 -Q1   ;;realtime audio out and midi in and midi out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o midinoteonkey.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;uses external midi device

kkey init 0      ;initialize key number
kvel init 0      ;initialize velocity
midinoteonkey kkey, kvel ;MIDI note number value
printk2 kkey      ;display the key value when it changes and when key is pressed
midion 1, kkey, kvel ;sent note to external device

endin
</CsInstruments>
<CsScore>
f 0 30 ;runs for 30 seconds

i 1 0 2 62 ; play these notes from score as well
i 1 + 2 65
e
</CsScore>
</CsoundSynthesizer>
```

La sortie contiendra des lignes comme celles-ci :

```
i1      60.00000
i1      69.00000
```

## Voir aussi

*midichannelaftertouch, midicontrolchange, mididefault, midinoteoff, midinoteoncps, midinoteonoct, midinoteonpch, midipitchbend, midipolyaftertouch, midiprogramchange*

## Crédits

Auteur : Michael Gogins

Nouveau dans la version 4.20

# midinoteonoct

midinoteonoct — Retourne un numéro de note MIDI traduit valeur octave-point-décimal.

## Description

*midinoteonoct* est conçu pour simplifier l'écriture d'instruments que l'on peut utiliser de manière interchangeable avec une partition ou depuis l'entrée MIDI, et pour faciliter l'adaptation d'instruments écrits à l'origine pour une partition au fonctionnement à partir d'une entrée MIDI.

En général, il doit être possible d'écrire des définitions d'instrument qui fonctionnent de la même manière avec une partition et avec le MIDI, que ce soit un fichier MIDI ou une entrée MIDI en , sans utiliser d'instructions conditionnelles supplémentaires, et qui tirent pleinement avantage des messages de voix MIDI.

Noter que la liaison entre des instruments de Csound et les numéros de canal MIDI se fait en utilisant l'opcode *massign* lors d'une exécution en . Pour les exécutions de fichier MIDI, les numéros d'instruments sont liés par défaut au canal MIDI + 1, mais ces valeurs par défaut peuvent être modifiées par tout message de changement de programme dans le fichier.

## Syntaxe

**midinoteonoct** *xoct*, *xvelocity*

## Exécution

*xoct* -- retourne une touche MIDI traduite en octaves linéaires durant l'activation MIDI, et reste inchangé dans les autres cas.

*xvelocity* -- retourne une vélocité MIDI durant l'activation MIDI, et reste inchangé dans les autres cas.

Si l'instrument a été activé par une entrée MIDI, l'opcode remplace les valeurs de *xoct* et de *xvelocity* par les valeurs correspondantes venant de l'entrée MIDI. Si l'instrument n'a *PAS* été activé par une entrée MIDI, les valeurs de *xoct* et de *xvelocity* restent inchangées.

Grâce à cela, les p-champs de la partition peuvent recevoir leur valeur de données MIDI en entrée durant l'activation MIDI, et de la partition dans les autres cas.



### Adaptation d'un instrument de Csound activé par partition.

Voir la section *Opcodes pour l'Interopérabilité MIDI/Partition* pour plus de détails sur l'adaptation d'instruments pilotés par partition au MIDI et vice-versa.

## Exemples

Voici un exemple de l'opcode *midinoteonoct*. Il utilise le fichier *midinoteonoct.csd* [exemples/midinoteonoct.csd].

### Exemple 562. Exemple de l'opcode *midinoteonoct*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      -M1   ;;realtime audio out and midi in
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o midinoteonoct.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

midinoteonoct p4, p5   ;gets a MIDI note number value as octave-point-decimal value into p4, and MIDI v

print p4      ;display the key value when it changes and when key is pressed
kvel = p5/127   ;scale midi velocity to 0-1
ioct = p4
icps = cpsoct(ioct) ;convert octave-point-decimal value into Hz
kenv madsr 0.5, 0.8, 0.8, 0.5 ;amplitude envelope multiplied by
asig pluck kenv*kvel, icps, icps, 2, 1 ;velocity value
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 0 30 ;runs for 30 seconds
f 2 0 4096 10 1

i 1 0 2 8.000 100 ; play these notes from score as well
i 1 + 2 8.917 100
e

</CsScore>
</CsoundSynthesizer>

```

La sortie contiendra des lignes comme celles-ci :

```

i1      8.00000
i1      8.91700
i1      9.33333

```

## Voir aussi

*midichannelaftertouch, midicontrolchange, mididefault, midinoteoff, midinoteoncps, midinoteonkey, midinoteonpch, midipitchbend, midipolyaftertouch, midiprogramchange*

## Crédits

Auteur : Michael Gogins

Nouveau dans la version 4.20

# midinoteonpch

midinoteonpch — Retourne un numéro de note MIDI traduit en classe de hauteur.

## Description

*midinoteonpch* est conçu pour simplifier l'écriture d'instruments que l'on peut utiliser de manière interchangeable avec une partition ou depuis l'entrée MIDI, et pour faciliter l'adaptation d'instruments écrits à l'origine pour une partition au fonctionnement à partir d'une entrée MIDI.

En général, il doit être possible d'écrire des définitions d'instrument qui fonctionnent de la même manière avec une partition et avec le MIDI, que ce soit un fichier MIDI ou une entrée MIDI en , sans utiliser d'instructions conditionnelles supplémentaires, et qui tirent pleinement avantage des messages de voix MIDI.

Noter que la liaison entre des instruments de Csound et les numéros de canal MIDI se fait en utilisant l'opcode *massign* lors d'une exécution en . Pour les exécutions de fichier MIDI, les numéros d'instruments sont liés par défaut au canal MIDI + 1, mais ces valeurs par défaut peuvent être modifiées par tout message de changement de programme dans le fichier.

## Syntaxe

**midinoteonpch** *xpch*, *xvelocity*

## Exécution

*xpch* -- retourne une touche MIDI traduite en classe de hauteur durant l'activation MIDI, et reste inchangé dans les autres cas.

*xvelocity* -- retourne une vélocité MIDI durant l'activation MIDI, et reste inchangé dans les autres cas.

Si l'instrument a été activé par une entrée MIDI, l'opcode remplace les valeurs de *xpch* et de *xvelocity* par les valeurs correspondantes venant de l'entrée MIDI. Si l'instrument n'a *PAS* été activé par une entrée MIDI, les valeurs de *xpch* et de *xvelocity* restent inchangées.

Grâce à cela, les p-champs de la partition peuvent recevoir leur valeur de données MIDI en entrée durant l'activation MIDI, et de la partition dans les autres cas.



### Adaptation d'un instrument de Csound activé par partition.

Voir la section *Opcodes pour l'Interopérabilité MIDI/Partition* pour plus de détails sur l'adaptation d'instruments pilotés par partition au MIDI et vice-versa.

## Exemples

Voici un exemple de l'opcode *midinoteonpch*. Il utilise le fichier *midinoteonpch.csd* [exemples/midinoteonpch.csd].

### Exemple 563. Exemple de l'opcode *midinoteonpch*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      -M1    ;;realtime audio out and midi in
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o midinoteonpch.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

midinoteonpch p4, p5    ;gets a MIDI note number value as octave-point-decimal value into p4, and MIDI

print p4    ;display the pitch value when it changes and when key is pressed
kvel = kvel/127    ;scale midi velocity to 0-1
ipch = p4
icps = cpspch(ipch)    ;convert octave-point-decimal value into Hz
kenv madsr 0.5, 0.8, 0.8, 0.5    ;amplitude envelope multiplied by
asig pluck kenv*kvel, icps, icps, 2, 1    ;velocity value
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 0 30 ;runs for 30 seconds
f 2 0 4096 10 1

i 1 0 2 8.09 100 ; play these notes from score as well
i 1 + 2 9.05 100
e

</CsScore>
</CsoundSynthesizer>
```

La sortie contiendra des lignes comme celles-ci :

```
i1      8.09000
i1      9.05000
```

## Voir aussi

*midichannelaftertouch, midicontrolchange, mididefault, midinoteoff, midinoteoncps, midinoteonkey, midinoteonoct, midipitchbend, midipolyaftertouch, midiprogramchange*

## Crédits

Auteur : Michael Gogins

Nouveau dans la version 4.20

# midion2

midion2 — Envoie des messages note on et note off sur le port MIDI OUT.

## Description

Envoie des messages note on et note off sur le port MIDI OUT lorsqu'il est déclenché par une valeur différente de zéro.

## Syntaxe

```
midion2 kchn, knum, kvel, ktrig
```

## Exécution

*kchn* -- canal MIDI (1-16)

*knun* -- numéro de note MIDI (0-127)

*kvel* -- vélocité de la note (0-127)

*ktrig* -- signal déclencheur en entrée (normalement 0)

Identique à *midion*, cet opcode envoie des messages note on et note off sur le port MIDI OUT, mais seulement lorsque *ktrig* est différent de zéro. Cet opcode peut travailler de concert avec la sortie de l'opcode *trigger*.

## Exemples

Voici un exemple de l'opcode midion2. Il utilise le fichier *midion2.csd* [exemples/midion2.csd].

### Exemple 564. Exemple de l'opcode midion2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-M0 -Q1 ;;midi in and midi out
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kcps line 3, p3, .1
klf lfo 1, kcps, 3 ;use a unipolar square to trigger
ktr trigger klf, 1, 1 ;from 3 times to .1 time per sec.
midion2 1, 60, 100, ktr
```

```
endin
</CsInstruments>
<CsScore>

i 1 0 20
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*moscil, midion, noteon, noteoff, noteondur, noteondur2*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1998

Nouveau dans la version 3.492 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.



# midion

midion — Génère des messages de note MIDI au taux-k.

## Description

Génère des messages de note MIDI au taux-k.

## Syntaxe

```
midion kchn, knum, kvel
```

## Exécution

*kchn* -- numéro de canal MIDI (1-16)

*knun* -- numéro de note (0-127)

*kvel* -- vélocité (0-127)

*midion* (note on au taux-k) joue des notes MIDI avec les valeurs courantes de *kchn*, *knun* et *kvel*. Ces arguments peuvent varier au taux-k. Chaque fois que la valeur MIDI convertie de l'un de ces arguments change, la dernière note MIDI jouée par l'instance courante de *midion* est immédiatement arrêtée et une nouvelle note avec le nouvel argument est activée. Cet opcode, comme *moscil*, peut générer des textures mélodiques très complexes s'il est contrôlé par des signaux complexes de taux-k.

Il peut y avoir n'importe quel nombre d'opcodes *midion* dans le même instrument de Csound, ce qui permet une polyphonie de style contrapointique avec un seul instrument.

## Exemples

Voici un exemple de l'opcode *midion*. Il utilise le fichier *midion\_simple.csd* [examples/midion\_simple.csd].

### Exemple 565. Exemple simple de l'opcode *midion*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

Cet exemple génère un accord mineur sur chaque note reçue sur l'entrée MIDI. Il génère des notes MIDI sur la sortie MIDI de Csound, si bien qu'il faut y connecter quelque chose.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadac     -d          -M0  -Q1  ;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
```

```

nchnls = 2

; Example by Giorgio Zucco 2007

instr 1 ;Triggered by MIDI notes on channel 1

    ifund notnum
    ivel  veloc

    knote1 init ifund
    knote2 init ifund + 3
    knote3 init ifund + 5

    ;minor chord on MIDI out channel 1
    ;Needs something plugged to csound's MIDI output
    midion 1, knote1,ivel
    midion 1, knote2,ivel
    midion 1, knote3,ivel

endin

</CsInstruments>
<CsScore>
; Dummy ftable
f0 60
</CsScore>
</CsoundSynthesizer>

```

Voici un autre exemple de l'opcode midion. Il utilise le fichier *midion\_scale.csd* [examples/midion\_scale.csd].

### Exemple 566. Exemple de l'opcode midion pour générer aléatoirement des notes sur une échelle.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

Cet exemple génère aléatoirement des notes prises dans une échelle donnée pour chaque note reçue sur l'entrée MIDI. Il génère des notes MIDI sur la sortie MIDI de Csound, si bien qu'il faut y connecter quelque chose.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d           -M0  -Q1  ;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

instr 1 ; Triggered by MIDI notes on channel 1

    ivel  veloc

    krate = 8

```

```

    iscale = 100 ;f

    ; Random sequence from table f100
    krnd randh int(14),krate,-1
    knote table abs(krnd),iscale
    ; Generates random notes from the scale on ftable 100
    ; on channel 1 of csound's MIDI output
    midion 1,knote,ivel

endin

</CsInstruments>
<CsScore>
f100 0 32 -2 40 50 60 70 80 44 54 65 74 84 39 49 69 69

; Dummy ftable
f0 60
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*moscil, midion2, noteon, noteoff, noteondur, noteondur2*

## Crédits

Auteur : Gabriel Maldonado  
 Italie  
 Mai 1997

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# midout

midout — Envoie un message générique MIDI sur le port MIDI OUT.

## Description

Envoie un message générique MIDI sur le port MIDI OUT.

## Syntaxe

```
midout kstatus, kchan, kdata1, kdata2
```

## Exécution

*kstatus* -- le type du message MIDI. Peut être :

- 128 (note off)
- 144 (note on)
- 160 (aftertouch polyphonique)
- 176 (changement de contrôle)
- 192 (changement de programme)
- 208 (aftertouch de canal)
- 224 (pitch bend)
- 0 si aucun message MIDI ne doit être envoyé au port MIDI OUT

*kchan* -- canal MIDI (1-16)

*kdata1*, *kdata2* -- données dépendant du message

*midout* n'a pas d'arguments de sortie, car il envoie implicitement un message sur le port MIDI OUT. Il travaille au taux-k. Il n'envoie un message MIDI que lorsque *kstatus* est différent de zéro.



### Avertissement

*Avertissement* : Normalement *kstatus* doit valoir 0. Il ne faut lui donner le numéro d'un type de message MIDI que si l'on veut envoyer ce message MIDI.

## Exemples

Voici un exemple de l'opcode *midout*. Il utilise le fichier *midout.csd* [examples/midout.csd].

### Exemple 567. Exemple de l'opcode *midout*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac -Ma -Q1 ;;realtime audio out and midi out and midi in (all midi inputs)
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

midiout 192, 1, 21, 0 ;program change to instr. 21
inum notnum
ivel veloc
midion 1, inum, ivel

endin
</CsInstruments>
<CsScore>

i 1 0 3 80 100 ;play note for 3 seconds

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1998

Nouveau dans le version 3.492 de Csound.

# midiout\_i

`midiout_i` — Envoie un message générique MIDI sur le port MIDI OUT.

## Description

Envoie un message générique MIDI sur le port MIDI OUT.

## Syntaxe

```
midiout_i istatus, ichan, idata1, idata2
```

## Initialisation

*istatus* -- le type du message MIDI. Peut être :

- 128 (note off)
- 144 (note on)
- 160 (aftertouch polyphonique)
- 176 (changement de contrôle)
- 192 (changement de programme)
- 208 (aftertouch de canal)
- 224 (pitch bend)
- 0 si aucun message MIDI ne doit être envoyé au port MIDI OUT

*ichan* -- canal MIDI (1-16)

*idata1*, *idata2* -- données dépendant du message

`midiout_i` n'a pas d'arguments de sortie, car il envoie implicitement un message sur le port MIDI OUT. Il travaille au taux-i. Il n'envoie un message MIDI que lorsque *istatus* est différent de zéro.

## Exemples

Voici un exemple de l'opcode `midiout_i`. Il utilise le fichier *midiout\_i.csd* [examples/midiout\_i.csd].

### Exemple 568. Exemple de l'opcode `midiout_i`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac -Ma -Q1 ;;;realtime audio out and midi out and midi in (all midi inputs)
```

```

</CsOptions>
<CsInstruments>

  sr = 44100
  ksmps = 32
  nchnls = 2
  0dbfs = 1

  instr 1

  midiout_i 192, 1, 21, 0 ;program change to instr. 21
  inum notnum
  ivel veloc
  midion 1, inum, ivel

  endin
</CsInstruments>
<CsScore>

  i 1 0 3 80 100 ;play note for 3 seconds

  e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Gabriel Maldonado  
 Italie  
 1998

Nouveau dans le version 6.10 de Csound.

# midipitchbend

midipitchbend — Retourne une valeur de pitchbend MIDI.

## Description

*midipitchbend* est conçu pour simplifier l'écriture d'instruments que l'on peut utiliser de manière interchangeable avec une partition ou depuis l'entrée MIDI, et pour faciliter l'adaptation d'instruments écrits à l'origine pour une partition au fonctionnement à partir d'une entrée MIDI.

En général, il doit être possible d'écrire des définitions d'instrument qui fonctionnent de la même manière avec une partition et avec le MIDI, que ce soit un fichier MIDI ou une entrée MIDI en , sans utiliser d'instructions conditionnelles supplémentaires, et qui tirent pleinement avantage des messages de voix MIDI.

Noter que la liaison entre des instruments de Csound et les numéros de canal MIDI se fait en utilisant l'opcode *massign* lors d'une exécution en . Pour les exécutions de fichier MIDI, les numéros d'instruments sont liés par défaut au canal MIDI + 1, mais ces valeurs par défaut peuvent être modifiées par tout message de changement de programme dans le fichier.

## Syntaxe

```
midipitchbend xpitchbend [, ilow] [, ihigh]
```

## Initialisation

*ilow* (facultatif) -- valeur basse facultative après reéchellonnement, 0 par défaut.

*ihigh* (facultatif) -- valeur haute facultative après reéchellonnement, 127 par défaut.

## Exécution

*xpitchbend* -- retourne le pitchbend MIDI durant l'activation MIDI, et reste inchangé dans les autres cas.

Si l'instrument a été activé par une entrée MIDI, l'opcode remplace la valeur de *xpitchbend* par la valeur correspondante venant de l'entrée MIDI. Si l'instrument n'a *PAS* été activé par une entrée MIDI, la valeur de *xpitchbend* reste inchangée.

Grâce à cela, les p-champs de la partition peuvent recevoir leur valeur de données MIDI en entrée durant l'activation MIDI, et de la partition dans les autres cas.



### Adaptation d'un instrument de Csound activé par partition.

Voir la section *Opcodes pour l'Interopérabilité MIDI/Partition* pour plus de détails sur l'adaptation d'instruments pilotés par partition au MIDI et vice-versa.

## Exemples

Voici un exemple de l'opcode *midipitchbend*. Il utilise le fichier *midipitchbend.csd* [exemples/midipitchbend.csd].



## Exemple 569. Exemple de l'opcode `midipitchbend`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -M1 ;;;realtime audio out and midi in
;-iadc ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime output leave only the line below:
; -o midipitchbend.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

midinoteonoct p4, p5 ;gets a MIDI note number value as octave-point-decimal value into p4, and MIDI

kvel = p5/127 ;scale midi velocity to 0-1
kpb init 0
midipitchbend kpb
printk2 kpb ;display the pitch-bend value when it changes
koct = p4+kpb ;add pitchbend values to octave-point-decimal value
kcps = cpsoct(koct) ;convert octave-point-decimal value into Hz
kenv madsr 0.5, 0.8, 0.8, 0.5 ;amplitude envelope multiplied by
asig vco2 kenv*kvel, kcps ;velocity value
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 0 30 ;runs for 30 seconds
f 2 0 4096 10 1

i 1 0 2 8.000 100 ; play these notes from score as well
i 1 + 2 8.917 100
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
i1 0.12695
i1 0.00000
i1 -0.01562
```

## Voir aussi

*midichannelaftertouch, midicontrolchange, mididefault, midinoteoff, midinoteoncps, midinoteonkey, midinoteonoct, midinoteonpch, midipolyaftertouch, midiprogramchange*

## Crédits

Auteur : Michael Gogins

Nouveau dans la version 4.20

# midipolyaftertouch

midipolyaftertouch — Retourne une valeur d'aftertouch polyphonique MIDI.

## Description

*midipolyaftertouch* est conçu pour simplifier l'écriture d'instruments que l'on peut utiliser de manière interchangeable avec une partition ou depuis l'entrée MIDI, et pour faciliter l'adaptation d'instruments écrits à l'origine pour une partition au fonctionnement à partir d'une entrée MIDI.

En général, il doit être possible d'écrire des définitions d'instrument qui fonctionnent de la même manière avec une partition et avec le MIDI, que ce soit un fichier MIDI ou une entrée MIDI en , sans utiliser d'instructions conditionnelles supplémentaires, et qui tirent pleinement avantage des messages de voix MIDI.

Noter que la liaison entre des instruments de Csound et les numéros de canal MIDI se fait en utilisant l'opcode *massign* lors d'une exécution en . Pour les exécutions de fichier MIDI, les numéros d'instruments sont liés par défaut au canal MIDI + 1, mais ces valeurs par défaut peuvent être modifiées par tout message de changement de programme dans le fichier.

## Syntaxe

```
midipolyaftertouch xpolyaftertouch, xcontrollervalue [, ilow] [, ihigh]
```

## Initialisation

*ilow* (facultatif) -- valeur basse facultative après rééchellonnement, 0 par défaut.

*ihigh* (facultatif) -- valeur haute facultative après rééchellonnement, 127 par défaut.

## Exécution

*xpolyaftertouch* -- retourne l'aftertouch polyphonique MIDI durant l'activation MIDI, et reste inchangé dans les autres cas.

*xcontrollervalue* -- retourne la valeur du contrôleur MIDI durant l'activation MIDI, et reste inchangé dans les autres cas.

Si l'instrument a été activé par une entrée MIDI, l'opcode remplace les valeurs de *xpolyaftertouch* et de *xcontrollervalue* par les valeurs correspondantes venant de l'entrée MIDI. Si l'instrument n'a *PAS* été activé par une entrée MIDI, les valeurs de *xpolyaftertouch* et de *xcontrollervalue* restent inchangées.

Grâce à cela, les p-champs de la partition peuvent recevoir leur valeur de données MIDI en entrée durant l'activation MIDI, et de la partition dans les autres cas.



### Adaptation d'un instrument de Csound activé par partition.

Voir la section *Opcodes pour l'Interopérabilité MIDI/Partition* pour plus de détails sur l'adaptation d'instruments pilotés par partition au MIDI et vice-versa.



## Crédits

Auteur : Michael Gogins

Nouveau dans la version 4.20

# midiprogramchange

midiprogramchange — Retourne une valeur de changement de programme MIDI.

## Description

*midiprogramchange* est conçu pour simplifier l'écriture d'instruments que l'on peut utiliser de manière interchangeable avec une partition ou depuis l'entrée MIDI, et pour faciliter l'adaptation d'instruments écrits à l'origine pour une partition au fonctionnement à partir d'une entrée MIDI.

En général, il doit être possible d'écrire des définitions d'instrument qui fonctionnent de la même manière avec une partition et avec le MIDI, que ce soit un fichier MIDI ou une entrée MIDI en , sans utiliser d'instructions conditionnelles supplémentaires, et qui tirent pleinement avantage des messages de voix MIDI.

Noter que la liaison entre des instruments de Csound et les numéros de canal MIDI se fait en utilisant l'opcode *massign* lors d'une exécution en . Pour les exécutions de fichier MIDI, les numéros d'instruments sont liés par défaut au canal MIDI + 1, mais ces valeurs par défaut peuvent être modifiées par tout message de changement de programme dans le fichier.

## Syntaxe

`midiprogramchange xprogram`

## Exécution

*xprogram* -- retourne une valeur de changement de programme MIDI durant l'activation MIDI, et reste inchangé dans les autres cas.

Si l'instrument a été activé par une entrée MIDI, l'opcode remplace la valeur de *xprogram* par la valeur correspondante venant de l'entrée MIDI. Si l'instrument n'a *PAS* été activé par une entrée MIDI, la valeur de *xprogram* reste inchangée.

Grâce à cela, les p-champs de la partition peuvent recevoir leur valeur de données MIDI en entrée durant l'activation MIDI, et de la partition dans les autres cas.



### Adaptation d'un instrument de Csound activé par partition.

Voir la section *Opcodes pour l'Interopérabilité MIDI/Partition* pour plus de détails sur l'adaptation d'instruments pilotés par partition au MIDI et vice-versa.

## Exemples

Voici un exemple de l'opcode *midiprogramchange*. Il utilise le fichier *midiprogramchange.csd* [exemples/midiprogramchange.csd].

### Exemple 571. Exemple de l'opcode *midiprogramchange*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

## Voir aussi

## Crédits

Nouveau dans la version 4.20

# miditempo

miditempo — Retourne le tempo courant au taux-k du fichier MIDI (s'il y en a un) ou de la partition.

## Description

Retourne le tempo courant au taux-k du fichier MIDI (s'il y en a un) ou de la partition (si l'option -t est utilisée).

## Syntaxe

ksig miditempo

## Exemples

Voici un exemple de l'opcode miditempo. Il utilise le fichier *miditempo.csd* [examples/miditempo.csd].

### Exemple 572. Exemple de l'opcode miditempo.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -F Anna.mid ;;realtime audio out and midi file input
;-iadc ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o miditempo.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

massign 0, 1 ; make sure that all channels
pgmassign 0, 1 ; and programs are assigned to test instr

instr 1

ksig miditempo
prints "miditempo = %d\\n", ksig

icps cpsmidi ; convert midi note to pitch
kenv madsr 0.1, 0, 0.8, 0.3
asig pluck kenv*.15, icps, icps, 1, 1 ;low volume
outs asig, asig

endin
</CsInstruments>
<CsScore>

f 0 200 ;stay active for 120 seconds
f 1 0 4096 10 1 ;sine
```



```
e  
</CsScore>  
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celle-ci :

```
miditempo = 96
```

## Crédits

Auteur : Istvan Varga  
Mars 2005  
Nouveau dans Csound5

# midremot

**midremot** — Un opcode que l'on peut utiliser pour implémenter un orchestre midi distant. Cet opcode envoie des évènements midi d'une machine source vers une machine de destination.

## Description

Avec les opcodes *midremot* et *midglobal* il est possible d'exécuter des instruments sur des machines distantes et de les contrôler depuis une machine maître. Les opcodes distants sont implémentés suivant le modèle maître/client. Toutes les machines concernées contiennent le même orchestre mais seule la machine maître possède l'information de la partition midi. Durant l'exécution, la machine maître envoie les évènements midi aux clients. L'opcode *midremot* envoie des évènements d'une machine source à une machine de destination. Pour envoyer des évènements à plusieurs destinations (diffusion), on utilise l'opcode *midglobal*. Ces deux opcodes peuvent être utilisés en combinaison.

## Syntaxe

```
midremot idestination, isource, instrnum [,instrnum...]
```

## Initialisation

*idestination* -- une chaîne représentant l'ordinateur de destination (par exemple 192.168.0.100). C'est l'hôte de destination qui reçoit les évènements de l'instrument donné.

*isource* -- une chaîne représentant l'ordinateur hôte (par exemple 192.168.0.100). C'est l'hôte source qui génère les évènements pour l'instruments donné et les envoie à l'adresse donnée par *idestination*.

*instrnum* -- liste des numéros des instruments qui seront joués sur la machines destinataire.

## Exemples

Voici un exemple de l'opcode *midremot*. Il utilise le fichier *insremot.csd* [examples/midremot.csd].

### Exemple 573. Exemple de l'opcode *midremot*.

L'exemple montre une fugue de Bach jouée sur 4 ordinateurs distants. La machine maître est nommée "192.168.1.100", le client1 "192.168.1.101" et ainsi de suite. Démarrer le client sur chaque machine (ils attendront de recevoir les évènements de la machine maître), puis démarrer le maître. Sur un système linux, on démarre un client avec la commande (csound -dm0 -odac -+rtaudio=alsa midremot.csd -+rtmidi=None), tandis que la commande sur la machine maître ressemble à ceci (csound -dm0 -odac -+rtaudio=alsa midremot.csd -F midremot.mid).

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
```

```
<CsOptions>
```

```
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o midremot.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>
sr = 44100
kr = 441
ksmps = 100
nchnls = 2

massign 1,1
massign 2,2
massign 3,3
massign 4,4
massign 5,5

ga1 init 0
ga2 init 0

gi1 sload "19Trumpet.sf2"

gi2 sload "01hpschd.sf2"

gi3 sload "07AcousticGuitar.sf2"

gi4 sload "22Bassoon.sf2"

gitab ftgen 1,0,1024,10,1

midremot "192.168.1.100", "192.168.1.101", 1
midremot "192.168.1.100", "192.168.1.102", 2
midremot "192.168.1.100", "192.168.1.103", 3

midglobal "192.168.1.100", 5

instr 1
  sfpassign 0, gi1
  ifreq cpsmidi
  iamp ampmidi 10
  inum notnum
  ivel veloc
  kamp linsegr 1,1,1,.1,0
  kfreq init 1
  a1,a2 sfplay ivel,inum,kamp*iamp,kfreq,0,0
  outs a1,a2
  vincer ga1, a1*.5
  vincer ga2, a2*.5
endin

instr 2
  sfpassign 0, gi2
  ifreq cpsmidi
  iamp ampmidi 15
  inum notnum
  ivel veloc
  kamp linsegr 1,1,1,.1,0
  kfreq init 1
  a1,a2 sfplay ivel,inum,kamp*iamp,kfreq,0,0
  outs a1,a2
  vincer ga1, a1*.4
  vincer ga2, a2*.4
endin

instr 3
  sfpassign 0, gi3
  ifreq cpsmidi
  iamp ampmidi 10
  inum notnum
  ivel veloc
  kamp linsegr 1,1,1,.1,0

```

```
kfreq init 1
a1,a2 sfplay ivel,inum,kamp*iamp,kfreq,0,0
outs a1,a2
vincr ga1, a1*.5
vincr ga2, a2*.5
endin

instr 4
sfpassign 0, gi4
ifreq cpsmidi
iamp ampmidi 15
inum notnum
ivel veloc
kamp linsegr 1,1,1,.1,0
kfreq init 1
a1,a2 sfplay ivel,inum,kamp*iamp,kfreq,0,0
outs a1,a2
vincr ga1, a1*.5
vincr ga2, a2*.5
endin

instr 5
    kamp midic7 1,0,1
    denorm ga1
    denorm ga2
aL, aR reverbsc ga1, ga2, .9, 16000, sr, 0.5
    outs aL, aR
    ga1 = 0
    ga2 = 0
endin

</CsInstruments>
<CsScore>
; Score
f0 160
</CsScore>

</CsoundSynthesizer>
```

## Voir aussi

*insglobal, insremot, midglobal, remoteport*

## Crédits

Auteur : Simon Schampijer  
2006

Nouveau dans la version 5.03

# min

min — Produit un signal qui est le minimum de tous les signaux d'entrée.

## Description

L'opcode *min* prend en entrée n'importe quel nombre de signaux de taux-a, de taux-k ou de taux-i (tous au même taux), et retourne un signal du même taux qui est le minimum de toutes les entrées. Pour les signaux de taux-a, les entrées sont comparées échantillon par échantillon (c-à-d que *min* n'examine pas une période *ksmps* entière du signal pour trouver son minimum local comme l'opcode *max\_k* le fait).

## Syntaxe

```
amin min ain1, ain2 [, ain3] [, ain4] [...]
```

```
kmin min kin1, kin2 [, kin3] [, kin4] [...]
```

```
imin min iin1, iin2 [, iin3] [, iin4] [...]
```

## Exécution

*ain1, ain2, ...* -- signaux de taux-a à comparer.

*kin1, kin2, ...* -- signaux de taux-k à comparer.

*iin1, iin2, ...* -- signaux de taux-i à comparer.

## Exemples

Voici un exemple de l'opcode min. Il utilise le fichier *min.csd* [examples/min.csd].

### Exemple 574. Exemple de l'opcode min.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o min.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

k1  oscili 1, 10.0, 1    ;combine 3 sinusses
k2  oscili 1, 1.0, 1    ;at different rates
```

```

k3  oscili 1, 3.0, 1
kmin min k1, k2, k3
kmin = kmin*250 ;scale kmin
printk2 kmin ;check the values

aout vco2 .5, 220, 6 ;sawtooth
asig moogvcf2 aout, 600+kmin, .5 ;change filter around 600 Hz
outs asig, asig

endin

</CsInstruments>
<CsScore>

f1 0 32768 10 1

i1 0 5
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*max, maxabs, minabs, maxaccum, minaccum, maxabsaccum, minabsaccum, max\_k*

## Crédits

Auteur : Anthony Kozar  
Mars 2006

Nouveau dans la version 5.01 de Csound ; taux-i ajouté dans la version 6.04

# minabs

**minabs** — Produit un signal qui est le minimum des valeurs absolues de n'importe quel nombre de signaux d'entrée.

## Description

L'opcode *minabs* prend en entrée n'importe quel nombre de signaux de taux-a ou de taux-k (tous du même taux), et retourne un signal au même taux qui est le minimum de toutes les entrées. Il est identique à l'opcode *min* sauf qu'il prend la valeur absolue de chaque entrée avant de les comparer. Ainsi, la sortie est toujours non-négative. Pour les signaux de taux-a, les entrées sont comparées échantillon par échantillon (c-à-d que *minabs* n'examine pas une période *ksmps* entière du signal pour trouver son minimum local comme l'opcode *max\_k* le fait).

## Syntaxe

```
amin minabs ain1, ain2 [, ain3] [, ain4] [...]  
kmin minabs kin1, kin2 [, kin3] [, kin4] [...]
```

## Exécution

*ain1, ain2, ...* -- signaux de taux-a à comparer.

*kin1, kin2, ...* -- signaux de taux-k à comparer.

## Exemples

Voici un exemple de l'opcode *minabs*. Il utilise le fichier *minabs.csd* [examples/minabs.csd].

### Exemple 575. Exemple de l'opcode *minabs*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac    ;;realtime audio out  
;-iadc    ;;uncomment -iadc if realtime audio input is needed too  
; For Non-realtime ouput leave only the line below:  
; -o minabs.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 32  
nchnls = 2  
0dbfs = 1  
  
instr 1  
  
k1    oscili 1, 10.0, 1    ;combine 3 sinusses  
k2    oscili 1, 1.0, 1    ;at different rates  
k3    oscili 1, 3.0, 1
```

```

kmin minabs k1, k2, k3
kmin = kmin*250 ;scale kmin
printk2 kmin ;check the values

aout vco2 .5, 220, 6 ;sawtooth
asig moogvcf2 aout, 600+kmin, .5 ;change filter above 600 Hz
outs asig, asig

endin

</CsInstruments>
<CsScore>
f1 0 32768 10 1

i1 0 5
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*maxabs, max, min, maxaccum, minaccum, maxabsaccum, minabsaccum, max\_k*

## Crédits

Auteur : Anthony Kozar  
Mars 2006

Nouveau dans la version 5.01 de Csound.



# minabsaccum

minabsaccum — Accumule le minimum de la valeur absolue de signaux audio.

## Description

*minabsaccum* compare deux variables de taux-audio et écrit le minimum de la valeur absolue des deux dans la première.

## Syntaxe

```
minabsaccum aAccumulator, aInput
```

## Exécution

*aAccumulator* -- variable audio dans laquelle la valeur minimale est écrite.

*aInput* -- signal auquel *aAccumulator* est comparé.

L'opcode *minabsaccum* est conçu pour accumuler la valeur minimale de plusieurs signaux audio qui peuvent provenir de plusieurs instances de note, dans différents canaux, ou qui ne peuvent être comparés en une fois au moyen de l'opcode *minabs*. *minabsaccum* est identique à *minaccum* sauf qu'il prend la valeur absolue de *aInput* avant la comparaison. Sa sémantique est semblable à celle de *vincr* car *aAccumulator* est utilisé à la fois comme variable d'entrée et comme variable de sortie, sauf que *minabsaccum* garde la valeur absolue minimale au lieu d'additionner les signaux ensemble. *minabsaccum* exécute l'opération suivante sur chaque paire d'échantillons :

```
if (abs(aInput) < aAccumulator) aAccumulator = abs(aInput)
```

*aAccumulator* sera habituellement une variable audio globale. A la fin de chaque cycle de calcul (période-k), après que sa valeur ait été lue et utilisée, la variable accumulateur doit habituellement être ré-initialisée à une valeur positive suffisamment grande pour être toujours supérieure aux signaux d'entrée auxquels elle est comparée.

## Exemples

Voici un exemple de l'opcode *minabsaccum*. Il utilise le fichier *minabsaccum.csd* [exemples/minabsaccum.csd].

### Exemple 576. Exemple de l'opcode minabsaccum.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime output leave only the line below:
; -o minabsaccum.wav -W ;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

  sr = 44100
  ksmps = 32
  nchnls = 2
  0dbfs = 1

  instr 1 ;saw

  asig vco2 .2, p4
      outs asig, asig
  gasaw = asig
  endin

  instr 2 ;sine

  aout poscil .3, p4, 1
      outs aout, aout
  gasin = aout
  endin

  instr 10

  accum init 0
      minabsaccum accum, gasaw + gasin ;saw and sine accumulated
  accum dcblock2 accum ;get rid of DC
      outs accum, accum

  clear accum
  endin

</CsInstruments>
<CsScore>
f 1 0 4096 10 1

i 1 0 7 330
i 2 3 3 440

i 1 10 7 330 ;same notes but without minabsaccum, for comparison
i 2 13 3 440

i 10 0 6 ;accumulation note stops after 6 seconds

e
</CsScore>
</CsSoundSynthesizer>

```

## Voir aussi

*maxabsaccum, maxaccum, minaccum, max, min, maxabs, minabs, vincr*

## Crédits

Auteur : Anthony Kozar  
Mars 2006

Nouveau dans la version 5.01 de Csound.

# minaccum

minaccum — Accumule la valeur minimale de signaux audio.

## Description

*minaccum* compare deux variables de taux-audio et écrit la valeur minimale des deux dans la première.

## Syntaxe

```
minaccum aAccumulator, aInput
```

## Exécution

*aAccumulator* -- variable audio dans laquelle la valeur minimale est écrite.

*aInput* -- signal auquel *aAccumulator* est comparé.

L'opcode *minaccum* est conçu pour accumuler la valeur minimale de plusieurs signaux audio qui peuvent provenir de plusieurs instances de note, dans différents canaux, ou qui ne peuvent être comparés en une fois au moyen de l'opcode *min*. Sa sémantique est semblable à celle de *vincr* car *aAccumulator* est utilisé à la fois comme variable d'entrée et comme variable de sortie, sauf que *minaccum* garde la valeur minimale au lieu d'additionner les signaux ensemble. *minaccum* exécute l'opération suivante sur chaque paire d'échantillons :

```
if (aInput < aAccumulator) aAccumulator = aInput
```

*aAccumulator* sera habituellement une variable audio globale. A la fin de chaque cycle de calcul (période-k), après que sa valeur ait été lue et utilisée, la variable accumulateur doit habituellement être ré-initialisée à une valeur positive suffisamment grande pour être toujours supérieure aux signaux d'entrée auxquels elle est comparée.

## Exemples

Voici un exemple de l'opcode minaccum. Il utilise le fichier *minaccum.csd* [examples/minaccum.csd].

### Exemple 577. Exemple de l'opcode minaccum.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o minaccum.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
```

```
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;saw

asig vco2 .2, p4
outs asig, asig
gasaw = asig
endin

instr 2 ;sine

aout poscil .3, p4, 1
outs aout, aout
gasin = aout
endin

instr 10

accum init 0
minaccum accum, gasaw + gasin ;saw and sine accumulated
outs accum, accum

clear accum
endin

</CsInstruments>
<CsScore>
f 1 0 4096 10 1

i 1 0 7 330
i 2 3 3 440

i 1 10 7 330 ;same notes but without minaccum, for comparison
i 2 13 3 440

i 10 0 6 ;accumulation note stops after 6 seconds

e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*maxaccum, maxabsaccum, minabsaccum, max, min, maxabs, minabs, vincr*

## Crédits

Auteur : Anthony Kozar  
Mars 2006

Nouveau dans la version 5.01 de Csound.

# mincer

mincer — Traitement vocodeur à verrouillage de phase.

## Description

*mincer* implémente le traitement vocodeur à verrouillage de phase en utilisant des tables de fonction contenant des sources sonores échantillonnées avec *GEN01*. *mincer* accepte les tables à allocation différée.

Cet opcode permet des mises à l'échelle du temps et de la fréquence indépendantes. Le temps est contrôlé par un indice temporel (en secondes) de la position dans la table de fonction et peut évoluer en avant ou en arrière à n'importe quelle vitesse ainsi que s'arrêter à une position donnée ("gelé"). La qualité de l'effet est généralement améliorée avec le verrouillage de phase activé.

*mincer* met aussi à l'échelle la hauteur, indépendamment de la fréquence, en utilisant un facteur de transposition (de taux-k).

## Syntaxe

```
asig mincer atimpt, kamp, kpitch, ktab, klock[,ifftsize,idecim]
```

## Initialisation

*ifftsize* -- taille de la TFR size (puissance de deux), 2048 par défaut.

*idecim* -- décimation, 4 par défaut (ce qui signifie que la taille du saut est égale à la taille de la TFR divisée par 4)

## Exécution

*atimpt* -- position temporelle de l'échantillon audio courant en secondes. La lecture de table cycle à la fin de la table de fonction.

*kamp* -- mise à l'échelle de l'amplitude.

*kpitch* -- mise à l'échelle de la hauteur de grain (1=hauteur normale, < 1 plus grave, > 1 plus aigu ; négatif, inversée)

*klock* -- 0 ou 1, pour désactiver ou activer le verrouillage de phase.

*ktab* -- table de fonction du signal source. Les tables à allocation différée sont acceptées (voir *GEN01*), mais l'opcode attend une source mono. On peut changer de table au taux-k.

## Exemples

Voici un exemple de l'opcode mincer. Il utilise le fichier *mincer.csd* [examples/mincer.csd]

**Exemple 578. Exemple de l'opcode mincer.**

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o mincer.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

idur = p3
ilock = p4
ipitch = 1
itimescale = 0.5
iamp = 0.8

atime line 0,idur,idur*itimescale
asig mincer atime, iamp, ipitch, 1, ilock
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 0 1 "fox.wav" 0 0 0

i 1 0 5 0 ;not locked
i 1 6 5 1 ;locked

e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Victor Lazzarini  
Février 2010

Nouveau greffon dans la version 5.13

Février 2005.

# minarray

minarray — Retourne la valeur minimale dans un tableau.

## Description

L'opcode *minarray* retourne la valeur minimale dans un tableau de taux-k, et éventuellement son indice.

## Syntaxe

```
kmin [,kindx] minarray karray
```

## Exécution

*kmin* -- variable pour le résultat.

*kindx* -- position (indice) du résultat dans le tableau.

*tab* -- tableau à lire.

## Exemples

Voici un exemple de l'opcode minarray. Il utilise le fichier *minarray.csd* [examples/minarray.csd].

### Exemple 579. Exemple de l'opcode minarray.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n
</CsOptions>
<CsInstruments>
ksmps = 32
;example by joachim heintz

      seed          0

instr 1
;create an array with 10 elements
kArr[] init      10
;fill in random numbers and print them out
kIndx      =      0
  until kIndx == 10 do
    kNum      random      -100, 100
    kArr[kIndx] =      kNum
    printf      "kArr[%d] = %10f\n", kIndx+1, kIndx, kNum
    kIndx      +=      1
  od
;investigate minimum number and print it out
kMin, kMinIndx minarray kArr
      printf      "Minimum of kArr = %f at index %d\n", kIndx+1, kMin, kMinIndx
      turnoff
endin
```

```
</CsInstruments>
<CsScore>
i1 0 0.1
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*minarray*, *sumarray*, *scalearray*,

## Crédits

Auteur : John fitch  
Octobre 2011

Nouveau dans la version 5.14 de Csound.

Révisé dans la version 6.00 de Csound pour fonctionner avec les tableau multidimensionnels.



# mirror

mirror — Réfléchit le signal lorsqu'il dépasse les limites inférieure ou supérieure.

## Description

Réfléchit le signal lorsqu'il dépasse les limites inférieure ou supérieure.

## Syntaxe

```
ares mirror asig, klow, khigh  
ires mirror isig, ilow, ihigh  
kres mirror ksig, klow, khigh
```

## Initialisation

*isig* -- signal d'entrée

*ilow* -- limite inférieure

*ihigh* -- limite supérieure

## Exécution

*xsig* -- signal d'entrée

*klow* -- limite inférieure

*khigh* -- limite supérieure

*mirror* « réfléchit » le signal qui dépasse les limites inférieure ou supérieure.

Cet opcode est utile dans plusieurs situations, telles que l'indexation de table ou pour l'écrtage et le modelage de signaux de taux-a, de taux-i ou de taux-k.

## Exemples

Voici un exemple de l'opcode mirror. Il utilise le fichier *mirror.csd* [examples/mirror.csd].

### Exemple 580. Exemple de l'opcode mirror.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac      ;;;realtime audio out  
;-iadc      ;;;uncomment -iadc if realtime audio input is needed too  
; For Non-realtime ouput leave only the line below:  
; -o mirror.wav -W ;;; for file output any platform
```

```
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
Odbfs = 1
nchnls = 2

instr 1 ; Limit / Mirror / Wrap

igain = p4 ;gain
ilevl1 = p5 ; + level
ilevl2 = p6 ; - level
imode = p7 ;1 = limit, 2 = mirror, 3 = wrap

ain soundin "fox.wav"
ain = ain*igain

if imode = 1 goto limit
if imode = 2 goto mirror

asig wrap ain, ilevl2, ilevl1
goto outsignal

limit:
asig limit ain, ilevl2, ilevl1
goto outsignal

mirror:
asig mirror ain, ilevl2, ilevl1
outsignal:

outs asig*.5, asig*.5 ;mind your speakers

endin

</CsInstruments>
<CsScore>

;          Gain +Levl -Levl Mode
i1 0 3 4.00 .25 -1.00 1 ;limit
i1 4 3 4.00 .25 -1.00 2 ;mirror
i1 8 3 4.00 .25 -1.00 3 ;wrap
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*limit, wrap*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.49 de Csound

# MixerSetLevel

MixerSetLevel — Fixe le niveau d'un départ vers un bus.

## Syntaxe

```
MixerSetLevel isend, ibuss, kgain
```

## Description

Opcodes du greffon mixer.

Fixe le niveau avec lequel les signaux d'un départ sont ajoutés au bus. L'envoi du signal vers le bus est effectué par l'opcode *MixerSend*.

## Initialisation

*isend* -- Le numéro du départ, par exemple, le numéro de l'instrument envoyant le signal (mais on peut utiliser n'importe quel nombre entier.

*ibuss* -- Le numéro du bus, par exemple, le numéro de l'instrument recevant le signal (mais on peut utiliser n'importe quel nombre entier.

En fixant le gain pour un bus, le bus est également créé.

## Exécution

*kgain* -- Le niveau (n'importe quel nombre réel) avec lequel le signal du départ est mélangé sur le bus. La valeur par défaut est 0.

L'utilisation du mélangeur nécessite que les instruments fixant les gains aient des numéros inférieurs à ceux des instruments envoyant des signaux, et que les instruments envoyant des signaux aient des numéros inférieurs à ceux des instruments recevant ces signaux. Cependant, un instrument peut avoir n'importe quel nombre de départs et de retours. Après la réception du dernier signal, il faut invoquer *MixerClear* pour réinitialiser les bus à 0 avant le k-cycle suivant.

## Exemples

Dans l'orchestre, définir un instrument pour contrôler les niveaux du mélangeur :

```
instr 1
  MixerSetLevel    p4, p5, p6
endin
```

Dans la partition, utiliser cet instrument pour fixer les niveaux du mélangeur :

```
; SoundFonts
; to Chorus
i 1 0 0 100 200 0.9
; to Reverb
i 1 0 0 100 210 0.7
; to Output
```

```
i 1 0 0 100 220 0.3

; Kelley Harpsichord
; to Chorus
i 1 0 0 3 200 0.30
; to Reverb
i 1 0 0 3 210 0.9
; to Output
i 1 0 0 3 220 0.1

; Chorus to Reverb
i 1 0 0 200 210 0.5
; Chorus to Output
i 1 0 0 200 220 0.5
; Reverb to Output
i 1 0 0 210 220 0.2
```

Voici un exemple complet de l'opcode MixerSetLevel. Il utilise le fichier *Mixer.csd* [examples/Mixer.csd]

### Exemple 581. Exemple complet de l'opcode MixerSetLevel.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      ; -iadc      ;;RT audio out
; For Non-realtime ouput leave only the line below:
; -o Mixer.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

katt expon 0.01, p3, 1 ;create an attack
aout poscil .7, 440,1
  MixerSetLevel 1, 3, katt ;impose attack on the gain level
  MixerSend aout, 1, 3, 0 ;send to channel 0
endin

instr 2

aout vco2 .5, 110 ;saw wave
  MixerSetLevel 2, 3, .25 ;set level to .25 of vco2
  MixerSend aout, 2, 3, 1 ;send to channel 1
endin

instr 3 ;mix instr.1 and 2 with reverb

kgain1 MixerGetLevel 1,3 ;get level form buss 3
kgain2 MixerGetLevel 2,3 ;get level form buss 3
a1 MixerReceive 3,0 ;receive channel 0
a2 MixerReceive 3,1 ;receive channel 1
aout = a1*kgain1+a2*kgain2 ;mix them
aoutL, aoutR reverb aout, aout, 0.85, 12000 ;add a nice reverb
outs aoutL, aoutR
MixerClear
endin

</CsInstruments>
<CsScore>
```

```
f1 0 4096 10 1

i1 0 2
i2 0 2
i3 0 8 ;reverb stays on for 8 sec.

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Michael Gogins (gogins at pipeline dot com).

# MixerSetLevel\_i

MixerSetLevel\_i — Fixe le niveau d'un départ vers un bus.

## Syntaxe

```
MixerSetLevel_i isend, ibuss, igain
```

## Description

Opcodes du greffon mixer.

Fixe le niveau avec lequel les signaux d'un départ sont ajoutés au bus. Parce que tous ses paramètres sont de taux-i, on peut utiliser cet opcode dans l'en-tête de l'orchestre. L'envoi du signal vers le bus est effectué par l'opcode *MixerSend*.

## Initialisation

*isend* -- Le numéro du départ, par exemple, le numéro de l'instrument envoyant le signal (mais on peut utiliser n'importe quel nombre entier.

*ibuss* -- Le numéro du bus, par exemple, le numéro de l'instrument recevant le signal (mais on peut utiliser n'importe quel nombre entier.

*igain* -- Le niveau (n'importe quel nombre réel) avec lequel le signal du départ est mélangé sur le bus. La valeur par défaut est 0.

En fixant le gain pour un bus, le bus est également créé.

## Exécution

L'utilisation du mélangeur nécessite que les instruments fixant les gains aient des numéros inférieurs à ceux des instruments envoyant des signaux, et que les instruments envoyant des signaux aient des numéros inférieurs à ceux des instruments recevant ces signaux. Cependant, un instrument peut avoir n'importe quel nombre de départs et de retours. Après la réception du dernier signal, il faut invoquer *MixerClear* pour réinitialiser les bus à 0 avant le k-cycle suivant.

## Exemples

Dans l'en-tête de l'orchestre, fixer le gain pour le départ du bus 3 vers le bus 4 :

```
MixerSetLevel_i 3, 4, 0.76
```

## Crédits

Michael Gogins (gogins at pipeline dot com).

# MixerGetLevel

MixerGetLevel — Retourne le niveau d'un départ vers un bus.

## Syntaxe

```
kgain MixerGetLevel isend, ibuss
```

## Description

Opcode du greffon mixer.

Retourne le niveau avec lequel les signaux d'un départ sont ajoutés au bus. L'envoi du signal vers le bus est effectué par l'opcode *MixerSend*.

## Initialisation

*isend* -- Le numéro du départ, par exemple, le numéro de l'instrument envoyant le signal.

*ibuss* -- Le numéro du bus, par exemple, le numéro de l'instrument recevant le signal.

## Exécution

*kgain* -- Le niveau (n'importe quel nombre réel) avec lequel le signal du départ est mélangé sur le bus.

Cet opcode retourne le niveau fixé par *MixerSetLevel* pour un départ et un bus.

L'utilisation du mélangeur nécessite que les instruments fixant les gains aient des numéros inférieurs à ceux des instruments envoyant des signaux, et que les instruments envoyant des signaux aient des numéros inférieurs à ceux des instruments recevant ces signaux. Cependant, un instrument peut avoir n'importe quel nombre de départs et de retours. Après la réception du dernier signal, il faut invoquer *MixerClear* pour réinitialiser les bus à 0 avant le k-cycle suivant.

## Exemples

Voici un exemple de l'opcode MixerGetLevel. Il utilise le fichier *Mixer.csd* [examples/Mixer.csd]

### Exemple 582. Exemple de l'opcode MixerGetLevel.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      ; -iadc    ;;RT audio out
; For Non-realtime ouput leave only the line below:
; -o Mixer.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
```

```

Odbfs = 1

instr 1

katt expon 0.01, p3, 1 ;create an attack
aout poscil .7, 440,1
  MixerSetLevel 1, 3, katt ;impose attack on the gain level
  MixerSend aout, 1, 3, 0 ;send to channel 0
endin

instr 2

aout vco2 .5, 110 ;saw wave
  MixerSetLevel 2, 3, .25 ;set level to .25 of vco2
  MixerSend aout, 2, 3, 1 ;send to channel 1
endin

instr 3 ;mix instr.1 and 2 with reverb

kgain1 MixerGetLevel 1,3 ;get level form buss 3
kgain2 MixerGetLevel 2,3 ;get level form buss 3
a1 MixerReceive 3,0 ;receive channel 0
a2 MixerReceive 3,1 ;receive channel 1
aout = a1*kgain1+a2*kgain2 ;mix them
aoutL, aoutR reverbsc aout, aout, 0.85, 12000 ;add a nice reverb
  outs aoutL, aoutR
  MixerClear
endin

</CsInstruments>
<CsScore>
f1 0 4096 10 1

i1 0 2
i2 0 2
i3 0 8 ;reverb stays on for 8 sec.

e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Michael Gogins (gogins at pipeline dot com).



# MixerSend

MixerSend — Mélange un signal de taux-a dans un canal d'un bus.

## Syntaxe

```
MixerSend asignal, isend, ibuss, ichannel
```

## Description

Mélange un signal de taux-a dans un canal d'un bus.

## Initialisation

*isend* -- Le numéro du départ, par exemple, le numéro de l'instrument envoyant le signal. Le gain du départ est contrôlé par l'opcode *MixerSetLevel*. Les départs sont numérotés pour pouvoir fixer différents niveaux pour différents départs indépendamment du niveau courant des signaux.

*ibuss* -- Le numéro du bus, par exemple, le numéro de l'instrument recevant le signal.

*ichannel* -- Le numéro du canal. Chaque bus a `nchnls` canaux.

## Exécution

*asignal* -- Le signal qui est mélangé dans le canal indiqué du bus.

L'utilisation du mélangeur nécessite que les instruments fixant les gains aient des numéros inférieurs à ceux des instruments envoyant des signaux, et que les instruments envoyant des signaux aient des numéros inférieurs à ceux des instruments recevant ces signaux. Cependant, un instrument peut avoir n'importe quel nombre de départs et de retours. Après la réception du dernier signal, il faut invoquer *MixerClear* pour réinitialiser les bus à 0 avant le k-cycle suivant.

## Exemples

```
instr 100 ; Fluidsynth output
; INITIALIZATION
; Normalize so iamplitude for p5 of 80 == ampdb(80).
iamplitude      =      ampdb(p5) * 2.0
; AUDIO
aleft, aright    fluidAllOut    giFluidsynth
asig1            =      aleft * iamplitude
asig2            =      aright * iamplitude
; To the chorus.
MixerSend  asig1, 100, 200, 0
MixerSend  asig2, 100, 200, 1
; To the reverb.
MixerSend  asig1, 100, 210, 0
MixerSend  asig2, 100, 210, 1
; To the output.
MixerSend  asig1, 100, 220, 0
MixerSend  asig2, 100, 220, 1
endin
```

Voici un exemple complet de l'opcode MixerSend. Il utilise le fichier *Mixer.csd* [examples/Mixer.csd]

**Exemple 583. Exemple complet de l'opcode MixerSend.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac        ; -iadc      ;;RT audio out
; For Non-realtime ouput leave only the line below:
; -o Mixer.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

katt expon 0.01, p3, 1 ;create an attack
aout poscil .7, 440,1
  MixerSetLevel 1, 3, katt ;impose attack on the gain level
  MixerSend aout, 1, 3, 0 ;send to channel 0
endin

instr 2

aout vco2 .5, 110 ;saw wave
  MixerSetLevel 2, 3, .25 ;set level to .25 of vco2
  MixerSend aout, 2, 3, 1 ;send to channel 1
endin

instr 3 ;mix instr.1 and 2 with reverb

kgain1 MixerGetLevel 1,3 ;get level form buss 3
kgain2 MixerGetLevel 2,3 ;get level form buss 3
a1 MixerReceive 3,0 ;receive channel 0
a2 MixerReceive 3,1 ;receive channel 1
aout = a1*kgain1+a2*kgain2 ;mix them
aoutL, aoutR reverbbsc aout, aout, 0.85, 12000 ;add a nice reverb
outs aoutL, aoutR
  MixerClear
endin

</CsInstruments>
<CsScore>
f1 0 4096 10 1

i1 0 2
i2 0 2
i3 0 8 ;reverb stays on for 8 sec.

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Michael Gogins (gogins at pipeline dot com).

# MixerReceive

MixerReceive — Reçoit un signal de taux-a depuis un canal d'un bus.

## Syntaxe

```
asignal MixerReceive ibuss, ichannel
```

## Description

Reçoit un signal de taux-a qui a été mélangé sur un canal d'un bus.

## Initialisation

*ibuss* -- Le numéro du bus, par exemple, le numéro de l'instrument recevant le signal.

*ichannel* -- Le numéro du canal. Chaque bus a *nchnls* canaux.

## Exécution

*asignal* -- Le signal qui a été mélangé sur le canal indiqué du bus.

L'utilisation du mélangeur nécessite que les instruments fixant les gains aient des numéros inférieurs à ceux des instruments envoyant des signaux, et que les instruments envoyant des signaux aient des numéros inférieurs à ceux des instruments recevant ces signaux. Cependant, un instrument peut avoir n'importe quel nombre de départs et de retours. Après la réception du dernier signal, il faut invoquer *MixerClear* pour réinitialiser les bus à 0 avant le k-cycle suivant.

## Exemples

```
instr 220 ; Master output
; It applies a bass enhancement, compression and fadeout
; to the whole piece, outputs signals, and clears the mixer.
a1 MixerReceive 220, 0
a2 MixerReceive 220, 1
; Bass enhancement
a11 butterlp a1, 100
a12 butterlp a2, 100
a1 = a11*1.5 + a1
a2 = a12*1.5 + a2

; Global amplitude shape
kenv linseg 0., p5 / 2.0, p4, p3 - p5, p4, p5 / 2.0, 0.
a1=a1*kenv
a2=a2*kenv

; Compression
a1 dam a1, 5000, 0.5, 1, 0.2, 0.1
a2 dam a2, 5000, 0.5, 1, 0.2, 0.1

; Remove DC bias
a1blocked dcblock a1
a2blocked dcblock a2

; Output signals
outs a1blocked, a2blocked
```

```
MixerClear
endin
```

Voici un exemple complet de l'opcode MixerReceive. Il utilise le fichier *Mixer.csd* [examples/Mixer.csd]

### Exemple 584. Exemple complet de l'opcode MixerReceive.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac      ; -iadc      ;;RT audio out
; For Non-realtime ouput leave only the line below:
; -o Mixer.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

katt expon 0.01, p3, 1 ;create an attack
aout poscil .7, 440,1
  MixerSetLevel 1, 3, katt ;impose attack on the gain level
  MixerSend aout, 1, 3, 0 ;send to channel 0
endin

instr 2

aout vco2 .5, 110 ;saw wave
  MixerSetLevel 2, 3, .25 ;set level to .25 of vco2
  MixerSend aout, 2, 3, 1 ;send to channel 1
endin

instr 3 ;mix instr.1 and 2 with reverb

kgain1 MixerGetLevel 1,3 ;get level form buss 3
kgain2 MixerGetLevel 2,3 ;get level form buss 3
a1 MixerReceive 3,0 ;receive channel 0
a2 MixerReceive 3,1 ;receive channel 1
aout = a1*kgain1+a2*kgain2 ;mix them
aoutL, aoutR reverbbsc aout, aout, 0.85, 12000 ;add a nice reverb
outs aoutL, aoutR
  MixerClear
endin

</CsInstruments>
<CsScore>
f1 0 4096 10 1

i1 0 2
i2 0 2
i3 0 8 ;reverb stays on for 8 sec.

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Michael Gogins (gogins at pipeline dot com).

# MixerClear

MixerClear — Réinitialise tous les canaux d'un bus à 0.

## Syntaxe

**MixerClear**

## Description

Réinitialise tous les canaux d'un bus à 0.

## Exécution

L'utilisation du mélangeur nécessite que les instruments fixant les gains aient des numéros inférieurs à ceux des instruments envoyant des signaux, et que les instruments envoyant des signaux aient des numéros inférieurs à ceux des instruments recevant ces signaux. Cependant, un instrument peut avoir n'importe quel nombre de départs et de retours. Après la réception du dernier signal, il faut invoquer *MixerClear* pour réinitialiser les bus à 0 avant le k-cycle suivant.

## Exemples

```
instr 220 ; Master output
    ; It applies a bass enhancement, compression and fadeout
    ; to the whole piece, outputs signals, and clears the mixer.
a1 MixerReceive 220, 0
a2 MixerReceive 220, 1
    ; Bass enhancement
a11 butterlp a1, 100
a12 butterlp a2, 100
a1 = a11*1.5 + a1
a2 = a12*1.5 + a2

    ; Global amplitude shape
kenv linseg 0., p5 / 2.0, p4, p3 - p5, p4, p5 / 2.0, 0.
a1=a1*kenv
a2=a2*kenv

    ; Compression
a1 dam a1, 5000, 0.5, 1, 0.2, 0.1
a2 dam a2, 5000, 0.5, 1, 0.2, 0.1

    ; Remove DC bias
alblocked dcblock a1
a2blocked dcblock a2

    ; Output signals
outs alblocked, a2blocked
MixerClear
endin
```

Voici un exemple complet de l'opcode Mixerclear. Il utilise le fichier *Mixer.csd* [examples/Mixer.csd]

**Exemple 585. Exemple complet de l'opcode Mixerclear.**

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac        ; -iadc      ;;RT audio out
; For Non-realtime ouput leave only the line below:
; -o Mixer.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

katt expon 0.01, p3, 1 ;create an attack
aout poscil .7, 440,1
  MixerSetLevel 1, 3, katt ;impose attack on the gain level
  MixerSend aout, 1, 3, 0 ;send to channel 0
endin

instr 2

aout vco2 .5, 110 ;saw wave
  MixerSetLevel 2, 3, .25 ;set level to .25 of vco2
  MixerSend aout, 2, 3, 1 ;send to channel 1
endin

instr 3 ;mix instr.1 and 2 with reverb

kgain1 MixerGetLevel 1,3 ;get level form buss 3
kgain2 MixerGetLevel 2,3 ;get level form buss 3
a1 MixerReceive 3,0 ;receive channel 0
a2 MixerReceive 3,1 ;receive channel 1
aout = a1*kgain1+a2*kgain2 ;mix them
aoutL, aoutR reverbSC aout, aout, 0.85, 12000 ;add a nice reverb
  outs aoutL, aoutR
  MixerClear
endin

</CsInstruments>
<CsScore>
f1 0 4096 10 1

i1 0 2
i2 0 2
i3 0 8 ;reverb stays on for 8 sec.

e
</CsScore>
</CsSoundSynthesizer>

```

## Crédits

Auteur : Michael Gogins (gogins at pipeline dot com).

# mode

mode — Un filtre simulant un système masse-ressort-amortisseur.

## Description

Filtre le signal entrant avec la fréquence de résonance et le facteur de qualité donnés. On peut aussi le voir comme un générateur de signal pour un grand facteur de qualité, avec une impulsion pour l'excitation. On peut combiner plusieurs modes pour construire des instruments complexes tels que des cloches ou des tables d'harmonie de guitare.

## Syntaxe

```
aout mode ain, xfreq, xQ [, iskip]
```

## Initialisation

*iskip* (facultatif, 0 par défaut) -- s'il est non nul, l'initialisation du filtre est ignorée.

## Exécution

*aout* -- signal filtré

*ain* -- signal à filtrer

*xfreq* -- fréquence de résonance du filtre



### Avertissement

Comme ce filtre devient instable si  $sr/xfreq < \pi$ , la valeur de *xfreq* est limitée en interne à  $sr/\pi - sr/100 >$  (par exemple *xfreq* > 13595 Hz à 44.1 kHz). Le terme  $sr/100$  est dû au fait que le filtre, bien que mathématiquement stable, a une très forte amplification lorsque l'on s'approche de la région instable.

*xQ* -- facteur de qualité du filtre

La durée de résonance est approximativement proportionnelle à  $xQ/xfreq$ .

Voir *Rapports de Fréquence Modale* pour des rapports de fréquence d'instruments réels que l'on peut utiliser pour déterminer les valeurs de *xfreq*.

## Exemples

Voici un exemple de l'opcode mode. Il utilise le fichier *mode.csd* [examples/mode.csd].

### Exemple 586. Exemple de l'opcode mode.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o moogvcf.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

instr 1; 2 modes excitator

idur init p3
ifreq11 init p4
ifreq12 init p5
iQ11      init p6
iQ12      init p7
iamp      init ampdb(p8)
ifreq21 init p9
ifreq22 init p10
iQ21      init p11
iQ22      init p12

; to simulate the shock between the excitator and the resonator
ashock mpulse 3,0

aexc1 mode ashock,ifreq11,iQ11
aexc1 = aexc1*iamp
aexc2 mode ashock,ifreq12,iQ12
aexc2 = aexc2*iamp

aexc = (aexc1+aexc2)/2

;"Contact" condition : when aexc reaches 0, the excitator looses
;contact with the resonator, and stops "pushing it"
aexc limit aexc,0,3*iamp

; 2modes resonator

ares1 mode aexc,ifreq21,iQ21
ares2 mode aexc,ifreq22,iQ22

ares = (ares1+ares2)/2

display aexc+ares,p3
outs aexc+ares,aexc+ares

endin

</CsInstruments>
<CsScore>

;wooden excitator against glass resonator
i1 0 8 1000 3000 12 8 70 440 888 500 420

;felt against glass
i1 4 8 80 188 8 3 70 440 888 500 420

```



```
;wood against wood
i1 8 8 1000 3000 12 8 70 440 630 60 53

;felt against wood
i1 12 8 80 180 8 3 70 440 630 60 53

i1 16 8 1000 3000 12 8 70 440 888 2000 1630
i1 23 8 80 180 8 3 70 440 888 2000 1630

;With a metallic excitator

i1 33 8 1000 1800 1000 720 70 440 882 500 500
i1 37 8 1000 1800 1000 850 70 440 630 60 53

i1 42 8 1000 1800 2000 1720 70 440 442 500 500

</CsScore>
</CsoundSynthesizer>
```

## Crédits

UDO original et documentation/exemple par François Blanc

Traduction de l'opcode en code-C par Steven Yi

Nouveau dans la version 5.04

Paramètres de taux audio introduits dans la version 6.02

Novembre 2013.

# modmatrix

modmatrix — Opcode matrice de modulation avec optimisation pour les matrices creuses.

## Description

On peut utiliser cet opcode pour faire moduler un grand nombre de paramètres variables de taux-k par un grand nombre de variables modulantes de taux-k, avec une pondération arbitraire de chaque connexion paramètre-modulateur. Des ftables de Csound sont utilisées pour contenir les variables en entrée (les paramètres), les variables modulantes et les coefficients de pondération. Les variables de sorties sont écrites dans une autre ftable de Csound.

## Syntaxe

```
modmatrix iresfn, isrcmodfn, isrcparmfn, imodscale, inum_mod, \\  
inum_parm, kupdate
```

## Initialisation

*iresfn* -- numéro de la ftable pour les variables de sortie.

*isrcmodfn* -- numéro de la ftable pour les variables sources de modulation.

*isrcparmfn* -- numéro de la ftable pour les paramètres variables en entrée.

*imodscale* -- matrice des coefficients de pondération/routage. C'est aussi une ftable de Csound, utilisée comme une matrice de *inum\_mod* lignes et *inum\_parm* colonnes.

*inum\_mod* -- nombre de variables de modulation.

*inum\_parm* -- nombre de paramètres variables (en entrée et en sortie).

Les arguments *inum\_mod* et *inum\_parm* ne doivent pas nécessairement être des puissances de deux.

## Exécution

*kupdate* -- indicateur pour la mise à jour des coefficients de pondération. Quand l'indicateur a une valeur non nulle, les coefficients de pondération sont lus directement de la table *imodscale*. Quand l'indicateur vaut zéro, les coefficients de pondérations sont parcourus et une matrice de pondération optimisée est stockée en interne dans l'opcode.

Chaque modulateur dans *isrcmodfn*, est pondéré par un coefficient (dans *imodscale*) déterminant son degré d'influence sur chaque paramètre. Puis tous les modulateurs pour un paramètre sont additionnés et la valeur de modulation résultante est ajoutée à la valeur du paramètre d'entrée lu dans *isrcparmfn*. Enfin, les valeurs du paramètre de sortie sont écrites dans la table *iresfn*.

Les tables suivantes donnent un aperçu du processus exécuté par l'opcode *modmatrix*, dans un exemple simplifié utilisant 3 paramètres et 2 modulateurs. Appelons les paramètres "cps1", "cps2" et "cutoff", et les modulateurs "lfo1" et "lfo2".

Les variables d'entrée peuvent avoir ces valeurs à un certain moment :

**Tableau 12.**

	<b>cps1</b>	<b>cps2</b>	<b>cutoff</b>
<i>isrcparmfn</i>	400	800	3

... tandis que les variables de modulation ont ces valeurs :

**Tableau 13.**

	<b>lfo1</b>	<b>lfo2</b>
<i>isrcmodfn</i>	0.5	-0.2

Les coefficients de pondération/routage sont :

**Tableau 14.**

<i>imodscale</i>	<b>cps1</b>	<b>cps2</b>	<b>cutoff</b>
<i>lfo1</i>	40	0	-2
<i>lfo2</i>	-50	100	3

... et les valeurs de sortie résultantes sont :

**Tableau 15.**

	<b>cps1</b>	<b>cps2</b>	<b>cutoff</b>
<i>iresfn</i>	430	780	1.4
<i>lfo2</i>	-50	100	3

La valeur de sortie pour "cps1" est calculée comme  $400 + (0.5 * 40) + (-0.2 * -50)$ , de même pour "cps2"  $800 + (0.5 * 0) + (-0.2 * 100)$ , et pour "cutoff" :  $3 + (0.5 * -2) + (-0.2 * 3)$

La ftable *imodscale* peut être spécifiée dans la partition comme ceci :

```
f1 0 8 -2 200 0 2 50 300 -1.5
```

Ou mieux en utilisant *ftgen* dans l'orchestre :

```
gimodscale ftgen 0, 0, 8, -2, 200, 0, 2, 50, 300, -1.5
```

Evidemment, les paramètres variables et les modulateurs n'ont pas nécessairement des valeurs statiques, de même que la table des coefficients de pondération/routage peut être continuellement renouvelée au moyen d'opcodes comme *tablew*.

## Exemples

Voici un exemple de l'opcode *modmatrix*. Il utilise le fichier *modmatrix.csd* [examples/modmatrix.csd].

### Exemple 587. Exemple de l'opcode *modmatrix*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio flags here according to platform
; Audio out   Audio in
;-odac        -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o modmatrix.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

  sr = 44100
  kr = 441
  ksmips = 100
  nchnls = 2
  0dbfs = 1

; basic waveforms
giSine ftgen 0, 0, 65537, 10, 1 ; sine wave
giSaw  ftgen 0, 0, 4097, 7, 1, 4096, -1 ; saw (linear)
giSoftSaw ftgen 0, 0, 65537, 30, giSaw, 1, 10 ; soft saw (only 10 first harmonics)

; modmatrix tables
giMaxNumParam = 128
giMaxNumMod = 32
giParam_In ftgen 0, 0, giMaxNumParam, 2, 0 ; input parameters table
; output parameters table (parameter values with added modulators)
giParam_Out ftgen 0, 0, giMaxNumParam, 2, 0
giModulators ftgen 0, 0, giMaxNumMod, 2, 0 ; modulators table
; modulation scaling and routing (mod matrix) table, start with empty table
giModScale ftgen 0, 0, giMaxNumParam*giMaxNumMod, -2, 0

;*****
; generate the modulator signals
;*****
instr 1

; LFO1, 1.5 Hz, normalized range (0.0 to 1.0)
kLFO1 oscil 0.5, 1.5, giSine ; generate LFO signal
kLFO1 = kLFO1+0.5 ; offset

; LFO2, 0.4 Hz, normalized range (0.0 to 1.0)
kLFO2 oscil 0.5, 0.4, giSine ; generate LFO signal
kLFO2 = kLFO2+0.5 ; offset

; write modulators to table
tablew kLFO1, 0, giModulators
tablew kLFO2, 1, giModulators

endin

;*****
; set parameter values
;*****
instr 2

; Here we can set the parameter values
icps1 = p4
icps2 = p5
icutoff = p6

; write parameters to table
tableiw icps1, 0, giParam_In
tableiw icps2, 1, giParam_In
tableiw icutoff, 2, giParam_In

```

```

    endin

;*****
; mod matrix edit
;*****
    instr 3

; Here we can write to the modmatrix table by using tablew or tableiw

iLfo1ToCps1 = p4
iLfo1ToCps2 = p5
iLfo1ToCutoff = p6
iLfo2ToCps1 = p7
iLfo2ToCps2 = p8
iLfo2ToCutoff = p9

    tableiw iLfo1ToCps1, 0, giModScale
    tableiw iLfo1ToCps2, 1, giModScale
    tableiw iLfo1ToCutoff, 2, giModScale
    tableiw iLfo2ToCps1, 3, giModScale
    tableiw iLfo2ToCps2, 4, giModScale
    tableiw iLfo2ToCutoff, 5, giModScale

; and set the update flag for modulator matrix
; ***(must update to enable changes)
ktrig init 1
    chnset ktrig, "modulatorUpdateFlag"
ktrig = 0

    endin

;*****
; mod matrix
;*****
    instr 4

; get the update flag
kupdate chnget "modulatorUpdateFlag"

; run the mod matrix
inum_mod = 2
inum_parm = 3
    modmatrix giParam_Out, giModulators, giParam_In, \
        giModScale, inum_mod, inum_parm, kupdate

; and reset the update flag
    chnset 0, "modulatorUpdateFlag" ; reset the update flag

    endin

;*****
; audio generator to test values
;*****
    instr 5

; basic parameters
iamp = ampdbfs(-5)

; read modulated parameters from table
kcps1 table 0, giParam_Out
kcps2 table 1, giParam_Out
kcutoff table 2, giParam_Out

; set filter parameters
kCF_freq1 = kcps1*kcutoff
kCF_freq2 = kcps2*kcutoff

```

```

kReso = 0.7
kDist = 0.3

; oscillators and filters
a1 oscili iamp, kcps1, giSoftSaw
a1 lpfl8 a1, kCF_freq1, kReso, kDist

a2 oscili iamp, kcps2, giSoftSaw
a2 lpfl8 a2, kCF_freq2, kReso, kDist

outs a1, a2

endin

</CsInstruments>
<CsScore>

;*****
; set initial parameters
; cps1 cps2 cutoff
i2 0 1 400 800 3

;*****
; set modmatrix values
; lfo1ToCps1 lfo1ToCps2 lfo1ToCut lfo2ToCps1 lfo2ToCps2 lfo2ToCut
i3 0 1 40 0 -2 -50 100 3

;*****
; start "always on" instruments
#define SCORELEN # 20 # ; set length of score

i1 0 $SCORELEN ; start modulators
i4 0 $SCORELEN ; start mod matrix
i5 0 $SCORELEN ; start audio oscillator

e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*Opcodes d'Algèbre Linéaire, Opcodes Vectoriels, tablew.*

## Crédits

Auteurs : Oeyvind Brandtsegg et Thom Johansen

Nouveau dans la version 5.12

# monitor

monitor — Retourne la trame audio de spout.

## Description

Retourne la trame audio de spout (s'il est actif), sinon retourne zéro.

## Syntaxe

```
aout1 [,aout2 ... aoutX] monitor  
aarra monitor
```

## Exécution

Dans sa forme tableau, il lit tous les canaux vers un tableau unidimensionnel.

Cet opcode peut être utilisé pour surveiller le signal de sortie de Csound. Il ne faut pas l'utiliser pour un traitement en aval du signal.

Voir l'article sur l'opcode *fout* pour un exemple de l'utilisation de *monitor*.

## Exemples

Voici un exemple de l'opcode monitor. Il utilise le fichier *monitor.csd* [examples/monitor.csd].

### Exemple 588. Exemple de l'opcode monitor.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac      ;;realtime audio out  
;-iadc      ;;uncomment -iadc if realtime audio input is needed too  
; For Non-realtime ouput leave only the line below:  
; -o monitor.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 32  
nchnls = 2  
0dbfs = 1  
  
giSine ftgen 0, 0, 2^10, 10, 1  
  
instr 1  
  
asig poscil3 .5, 880, giSine  
;write a raw file: 32 bits with header  
fout "fout_880.wav", 15, asig  
outs asig, asig
```

```

    endin

    instr 2

    klfo lfo 1, 2, 0
    asig poscil3 .5*klfo, 220, giSine
    ;write an aiff file: 32 bits with header
        fout "fout_aif.aiff", 25, asig
    ;        fout "fout_all3.wav", 14, asig
    outs asig, asig

    endin

    instr 99 ;read the stereo csound output buffer

    allL, allR monitor
    ;write the output of csound to an audio file
    ;to a wav file: 16 bits with header
        fout "fout_all.wav", 14, allL, allR

    endin
</CsInstruments>
<CsScore>

i 1 0 2
i 2 0 3
i 99 0 3
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*fout*, les *opcodes Mixer* et le *Système de Patch Zak*.

## Crédits

Istvan Varga 2006

John ffitich 2016, pour la forme tableau, nouveau dans la version 6.08



# moog

moog — Emulation d'un synthétiseur mini-Moog.

## Description

Emulation d'un synthétiseur mini-Moog.

## Syntaxe

```
ares moog kamp, kfreq, kfiltq, kfiltrate, kvibf, kvamp, iafn, iwfn, ivfn
```

## Initialisation

*iafn*, *iwfn*, *ivfn* -- les trois numéros des tables contenant la forme d'onde de l'attaque (non bouclée), la forme d'onde de la boucle principale, et la forme d'onde du vibrato. Les fichiers *mandpluk.aiff* [exemples/mandpluk.aiff] et *impuls20.aiff* [exemples/impuls20.aiff] conviennent bien pour les deux premières et une sinusoïde fera l'affaire pour la troisième.



### Note

Les fichiers « mandpluk.aiff » et « impuls20.aiff » sont aussi disponibles à <ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/>.

## Exécution

*kamp* -- amplitude de la note.

*kfreq* -- fréquence de la note.

*kfiltq* -- Q du filtre, compris entre 0,8 et 0,9

*kfiltrate* -- taux de contrôle pour le filtre, compris entre 0 et 0,0002

*kvibf* -- fréquence du vibrato en Hertz. L'intervalle conseillé va de 0 à 12

*kvamp* -- amplitude du vibrato

## Exemples

Voici un exemple de l'opcode moog. Il utilise les fichiers *moog.csd* [exemples/moog.csd], *mandpluk.aiff* [exemples/mandpluk.aiff] et *impuls20.aiff* [exemples/impuls20.aiff].

### Exemple 589. Exemple de l'opcode moog.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform
```

```

-odac    ;;;realtime audio out
;-iadc    ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime output leave only the line below:
; -o moog.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kfreq = cpspch(p4)
kfiltq = p5
kfiltrate = 0.0002
kvibf = 5
kvamp = .01
;low volume is needed
asig moog .15, kfreq, kfiltq, kfiltrate, kvibf, kvamp, 1, 2, 3
outs asig, asig

endin
</CsInstruments>
<CsScore>

f 1 0 8192 1 "mandpluk.aiff" 0 0 0
f 2 0 256 1 "impuls20.aiff" 0 0 0
f 3 0 256 10 1 ; sine

i 1 0 3 6.00 .1
i 1 + 3 6.05 .89
i 1 + 3 6.09 .50
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 3.47 de Csound

# moogladder

moogladder — Filtre passe-bas en échelle de Moog.

## Description

*moogladder* est une nouvelle implémentation numérique du filtre en échelle de Moog, basée sur le travail d'Antti Huovilainen décrit dans le papier "Non-Linear Digital Implementation of the Moog Ladder Filter" (Proceedings of DaFX04, Université de Naples). Cette implémentation est probablement une représentation numérique plus précise du filtre analogique original.

## Syntaxe

```
asig moogladder ain, kcf, kres[, istor]
asig moogladder ain, acf, kres[, istor]
asig moogladder ain, kcf, ares[, istor]
asig moogladder ain, acf, ares[, istor]
```

## Initialisation

*istor* -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal d'entrée.

*kcf/acf* -- fréquence de coupure du filtre.

*kres/ares* -- résonance, généralement  $< 1$ , mais pas limitée à cette valeur. Les valeurs de résonance supérieures à 1 peuvent produire des bruits de repliement ; les synthétiseurs analogiques permettent généralement d'avoir des résonances supérieures à 1.

## Exemples

Voici un exemple de l'opcode moogladder. Il utilise le fichier *moogladder.csd* [examples/moogladder.csd].

### Exemple 590. Exemple de l'opcode moogladder.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
```

```
; -o moogladder.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kfe expseg 500, p3*0.9, 1800, p3*0.1, 3000
asig buzz 1, 100, 20, 1
kres line .1, p3, .99 ;increase resonance
afil moogladder asig, kfe, kres
outs afil, afil

endin
</CsInstruments>
<CsScore>
f 1 0 4096 10 1

i 1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
Janvier 2005

Nouveau greffon dans la version 5

Janvier 2005.

Paramètres de taux audio introduits dans la version 6.02

Octobre 2013.

# moogladder2

moogladder2 — Filtre passe-bas en échelle de Moog.

## Description

*moogladder2* est une nouvelle implémentation numérique du filtre en échelle de Moog, basée sur le travail d'Antti Huovilainen décrit dans le papier "Non-Linear Digital Implementation of the Moog Ladder Filter" (Proceedings of DaFX04, Université de Naples). Cette implémentation utilise des valeurs approchées de la fonction tanh ce qui la rend plus rapide mais moins précise que *moogladder*.

## Syntaxe

```
asig moogladder2 ain, kcf, kres[, istor]
asig moogladder2 ain, acf, kres[, istor]
asig moogladder2 ain, kcf, ares[, istor]
asig moogladder2 ain, acf, ares[, istor]
```

## Initialisation

*istor* -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal d'entrée.

*kcf/acf* -- fréquence de coupure du filtre.

*kres/ares* -- résonance, généralement  $< 1$ , mais pas limitée à cette valeur. Les valeurs de résonance supérieures à 1 peuvent produire des bruits de repliement ; les synthétiseurs analogiques permettent généralement d'avoir des résonances supérieures à 1.

## Exemples

Voici un exemple de l'opcode *moogladder2*. Il utilise le fichier *moogladder2.csd* [exemples/moogladder2.csd].

### Exemple 591. Exemple de l'opcode *moogladder2*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;;realtime audio out
```

```

;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o moogladder2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kfe  expseg 500, p3*0.9, 1800, p3*0.1, 3000
asig buzz 1, 100, 20, 1
kres line .1, p3, .99 ;increase resonance
afil moogladder2 asig, kfe, kres
      outs afil, afil

endin
</CsInstruments>
<CsScore>
f 1 0 4096 10 1

i 1 0 10
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Victor Lazzarini

Auteur : John ffitich

Avril 2016

Nouveau dans la version 6.07

# moogvcf

**moogvcf** — Une simulation numérique de la configuration du filtre en échelle à diode de Moog.

## Description

Une simulation numérique de la configuration du filtre en échelle à diode de Moog.

## Syntaxe

```
ares moogvcf asig, xfco, xres [,iscale, iskip]
```

## Initialisation

*iscale* (facultatif, 1 par défaut) -- facteur de pondération interne. A utiliser si *asig* n'est pas dans l'intervalle +/-1. L'entrée est d'abord divisée par *iscale*, puis la sortie est multipliée par *iscale*. La valeur par défaut est 1. (Nouveau dans la version 3.50 de Csound).

*iskip* (facultatif, 0 par défaut) -- s'il est non nul, l'initialisation du filtre est ignorée. (Nouveau dans les versions 4.23f13 et 5.0 de Csound).

## Exécution

*asig* -- signal d'entrée

*xfco* -- fréquence de coupure du filtre en Hz. A partir de la version 3.50, peut-être de taux-i, de taux-k ou de taux-a.

*xres* -- quantité de résonance. Il y a des auto-oscillations lorsque *xres* est proche de 1. A partir de la version 3.50, peut-être de taux-i, de taux-k ou de taux-a.

*moogvcf* est une simulation numérique de la configuration du filtre en échelle à diode de Moog. Cette émulation est librement basée sur le papier « Analyzing the Moog VCF with Considerations for Digital Implementation » par Stilson et Smith (CCRMA). Cette version fut codée dans Csound à l'origine par Josep Comajuncosas. Quelques modifications et conversions en C ont été apportées par Hans Mikelson.



### Avertissement

Avant la version 6.02, ce filtre nécessitait un signal d'entrée normalisé à un. On peut l'obtenir facilement au moyen de *0dbfs*, comme ceci :

```
ares moogvcf asig, kfco, kres, 0dbfs
```

On peut aussi utiliser *moogvcf2* qui utilise comme mise à l'échelle par défaut *0dbfs*.

## Exemples

Voici un exemple de l'opcode *moogvcf*. Il utilise le fichier *moogvcf.csd* [examples/moogvcf.csd].

### Exemple 592. Exemple de l'opcode *moogvcf*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime output leave only the line below:
; -o moogvcf.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
; iscale does not need to be set here because already 0dbfs = 1
aout vco .3, 220, 1 ; Use a nice sawtooth waveform.
kfco line 200, p3, 2000 ; filter-cutoff frequency from .2 to 2 KHz
krez init p4
asig moogvcf aout, kfco, krez
    outs asig, asig

endin
</CsInstruments>
<CsScore>
;a sine wave
f 1 0 16384 10 1

i 1 0 3 .1
i 1 + 3 .7
i 1 + 3 .95
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*moogvcf2, biquad, rezzv*

## Crédits

Auteur : Hans Mikelson  
Octobre 1998

Nouveau dans la version 3.49 de Csound.



# moogvcf2

moogvcf2 — Une simulation numérique de la configuration du filtre en échelle à diode de Moog.

## Description

Une simulation numérique de la configuration du filtre en échelle à diode de Moog.

## Syntaxe

```
ares moogvcf2 asig, xfco, xres [,iscale, iskip]
```

## Initialisation

*iscale* (facultatif, 0dBfs par défaut) -- facteur de pondération interne, car les opérations du code nécessitent que le signal soit dans l'intervalle +/-1. L'entrée est d'abord divisée par *iscale*, puis la sortie est multipliée par *iscale*.

*iskip* (facultatif, 0 par défaut) -- s'il est non nul, l'initialisation du filtre est ignorée.

## Exécution

*asig* -- signal d'entrée

*xfco* -- fréquence de coupure du filtre en Hz. Peut-être de taux-i, de taux-k ou de taux-a.

*xres* -- quantité de résonance. Il y a des auto-oscillations lorsque *xres* est proche de 1. Peut-être de taux-i, de taux-k ou de taux-a.

*moogvcf2* est une simulation numérique de la configuration du filtre en échelle à diode de Moog. Cette émulation est librement basée sur le papier « Analyzing the Moog VCF with Considerations for Digital Implementation » par Stilson et Smith (CCRMA). Cette version fut codée dans Csound à l'origine par Josep Comajuncosas. Quelques modifications et conversions en C ont été apportées par Hans Mikelson et ensuite ajustées.

*moogvcf2* est identique à *moogvcf*, sauf que le paramètre *iscale* vaut par défaut *0dbfs* au lieu de 0, ce qui garantit que l'amplitude sera normalement correcte.

## Exemples

Voici un exemple de l'opcode *moogvcf2*. Il utilise le fichier *moogvcf2.csd* [examples/moogvcf2.csd].

### Exemple 593. Exemple de l'opcode *moogvcf2*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;;realtime audio out
```

```

;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o moogvcf2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

aout diskin2 "beats.wav", 1, 0, 1
kfco line 100, p3, 10000 ;filter-cutoff
krez init p4
asig moogvcf2 aout, kfco, krez
    outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 4 .1
i 1 + 4 .6
i 1 + 4 .9
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*moogvcf, biquad, rezy*

## Crédits

Auteur : Hans Mikelson et John ffitch  
 Octobre 1998 / Juillet 2006

Nouveau dans la version 5.03 de Csound.

# moscil

moscil — Envoie un flot de notes MIDI.

## Description

Envoie un flot de notes MIDI.

## Syntaxe

```
moscil kchn, knum, kvel, kdur, kpause
```

## Exécution

*kchn* -- numéro de canal MIDI (1-16)

*knum* -- numéro de note (0-127)

*kvel* -- vitesse (0-127)

*kdur* -- durée de note en secondes

*kpause* -- durée de la pause après chaque noteoff et avant la note suivante en secondes

*moscil* et *midion* sont les opcodes MIDI OUT les plus puissants. *moscil* (MIDI oscil) joue un flot de notes de durée *kdur*. Le canal, la hauteur, la vitesse, la durée et le temps de pause sont contrôlables au taux-*k*, ce qui permet de générer par algorithme des lignes mélodiques très complexes. Lorsque l'instrument courant est désactivé, les notes jouées par l'instance courante de *moscil* sont tronquées d'office.

Il peut y avoir n'importe quel nombre d'opcodes *moscil* dans le même instrument de Csound, ce qui permet une polyphonie de style contrapointique avec un seul instrument.

## Exemples

Voici un exemple de l'opcode *moscil*. Il utilise le fichier *moscil.csd* [examples/moscil.csd].

### Exemple 594. Exemple de l'opcode *moscil*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

Cet exemple génère un flot de notes pour chaque note reçue sur l'entrée MIDI. Il génère des notes MIDI sur la sortie MIDI de Csound, si bien qu'il faut y connecter quelque chose.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc    -d          -M0  -Q1;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
```

```

kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

instr 1 ;Triggered by MIDI notes on channel 1

inote notnum
ivel veloc

kpitch = 40
kfreq = 2

kdur = .04
kpause = .1

k1 lfo kpitch, kfreq, 5

;plays a stream of notes of kdur duration on MIDI channel 1
moscil 1, inote + k1, ivel, kdur, kpause

endin

</CsInstruments>
<CsScore>
; Dummy ftable
f0 60
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*midion, midion2, noteon, noteoff, noteondur, noteondur2*

## Crédits

Auteur : Gabriel Maldonado  
 Italie  
 Mai 1997

Nouveau dans la version 3.47 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# mp3in

mp3in — Lit des données audio mono ou stéréo depuis un fichier MP3 externe.

## Description

Lit des données audio mono ou stéréo depuis un fichier MP3 externe.

## Syntaxe

```
ar1, ar2 mp3in ifilcod[, iskptim, iformat, iskipinit, ibufsize]
ar1 mp3in ifilcod[, iskptim, iformat, iskipinit, ibufsize]
```

## Initialisation

*ifilcod* -- entier ou chaîne de caractères donnant le nom du fichier son source. Un entier indique le fichier soundin.filcod ; une chaîne de caractères (entre guillemets, espaces autorisés) donne le nom de fichier lui-même, éventuellement un nom de chemin complet. Si ce n'est pas un nom de chemin complet, le fichier nommé est d'abord cherché dans le répertoire courant, puis dans celui qui est donné par la variable d'environnement *SSDIR* (si elle est définie) puis par *SFDIR*.

*iskptim* (facultatif) -- portion du son en entrée à ignorer, exprimée en secondes. La valeur par défaut est 0.

*iformat* (facultatif) -- spécifie le format des données du fichier audio : n'est pas encore implémenté et vaut stéréo par défaut.

*iskipinit* (facultatif) -- supprime toute initialisation s'il est non nul (vaut 0 par défaut).

*ibuffersize* (facultatif) -- fixe la taille du tampon de lecture interne. Si la valeur est omise, nulle ou négative la taille par défaut est de 4096 octets.

## Exécution

Lit des données audio depuis un fichier MP3 externe.

## Exemples

Voici un exemple de l'opcode mp3in. Il utilise le fichier *mp3in.csd* [examples/mp3in.csd].

### Exemple 595. Exemple de l'opcode mp3in.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o mp3in.wav -W ;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

  sr = 44100
  ksmpps = 32
  nchnls = 2
  odbfs = 1

  instr 1

  iskptim = .3
  ibufsize = 64
  ar1, ar2 mp3in "beats.mp3", iskptim, 0, 0, ibufsize
    outs ar1, ar2

  endin
</CsInstruments>
<CsScore>

  i 1 0 2
  e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*diskin, ins, in, inh, inh, ino, inq, mp3len, soundin*

## Crédits

Auteur : John ffitch  
 Codemist Ltd  
 2009

Nouveau dans la version 5.11

Mono ajouté dans la version 6.05

# mp3len

mp3len — Retourne la longueur d'un fichier son MP3.

## Description

Retourne la longueur d'un fichier son MP3.

## Syntaxe

```
ir mp3len ifilcod
```

## Initialisation

*ifilcod* -- fichier son à interroger

## Exécution

*mp3len* retourne la longueur du fichier son *ifilcod* en secondes.

## Exemples

Voici un exemple de l'opcode mp3len. Il utilise le fichier *mp3len.csd* [examples/mp3len.csd].

### Exemple 596. Exemple de l'opcode mp3len.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o mp3len.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ilen  mp3len p4      ;calculate length of mp3 file
print ilen

asigL, asigR mp3in p4
outs  asigL, asigR

endin
</CsInstruments>
<CsScore>
```

```
i 1 0 30 "XORNOT_jul-14-05.mp3" ; long signal  
e  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*mp3in*

## Crédits

Auteur : John ffitch  
Février 2011

Nouveau dans la version 5.14 de Csound



# mp3scal

**mp3scal** — Traitement vocoder à verrouillage de phase avec détection/traitement d'attaque, 'pondération du tempo'.

## Description

*mp3scal* implémente un traitement vocoder à verrouillage de phase à partir de fichiers mp3 sur disque, avec rééchantillonnage si nécessaire.

Cet opcode permet une pondération indépendante du temps et de la fréquence. Le temps progresse en interne. La qualité de l'effet est généralement améliorée lorsque le verrouillage de phase est actif.

*mp3scal* pondère également la hauteur, indépendamment de la fréquence, avec un facteur de transposition de taux-k.

## Syntaxe

```
asig, asig2, ktime mp3scal Sfile, ktimescal, kpitch, kamp [,iskip, ifftsize, idecim, ilock]
```

## Initialisation

*Sfile* -- fichier son source, mp3 stéréo.

*ifftsize* -- taille de TFR (puissance de deux), 2048 par défaut.

*idecim* -- décimation, 4 par défaut (ce qui signifie hopsize = fftsize/4).

*iskip* -- temps de décalage en secondes, 1 par défaut.

*ilock* -- 0 ou 1, pour désactiver ou activer le verrouillage de phase, 1 par défaut.

## Exécution

*ktimescal* -- rapport de pondération temporelle, < 1 étirement, > 1 contraction. Nombres non-négatifs seulement.

*kamp* -- pondération de l'amplitude.

*kpitch* -- pondération de la hauteur des grains (1 = hauteur normale, < 1 inférieure, > 1 supérieure ; négative, lecture inversée).

*ktime* -- marque temporelle.

*asig, asig2* -- signal de sortie stéréo.

## Exemples

Voici un exemple de l'opcode *mp3scal*. Il utilise le fichier *mp3scal.csd* [exemples/mp3scal.csd].

### Exemple 597. Exemple de l'opcode mp3scal.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
nchnls=2
ksmps=64
sr=44100
instr 1
SFile = p4
p3 = mp3len(SFile)/p5
a1,a2,k2 mp3scal SFile,p5,1,1
outs a1,a2

endin

</CsInstruments>
<CsScore>
i1.1 0 1 "beats.mp3" .75
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
Avril 2016

Nouveau dans la version 6.07

Avril 2016.

# mpulse

mpulse — Génère un train d'impulsions.

## Description

Génère un train d'impulsions d'amplitude *kamp* séparées par *kintvl* secondes (ou échantillons si *kintvl* est négatif). La première impulsion est générée après un délai de *ioffset* secondes.

## Syntaxe

```
ares mpulse kamp, kintvl [, ioffset]
```

## Initialisation

*ioffset* (facultatif, par défaut 0) -- le délai avant la première impulsion. S'il est négatif, la valeur est interprétée comme le nombre d'échantillons, sinon il représente des secondes. La valeur par défaut est zéro.

## Exécution

*kamp* -- amplitude des impulsions générées

*kintvl* -- intervalle de temps en secondes (ou en nombre d'échantillons si *kintvl* est négatif) jusqu'à la prochaine impulsion.

Après le délai initial, une impulsion d'amplitude *kamp* est générée comme échantillon unique. Immédiatement après la génération de l'impulsion, la date de la suivante est déterminée par la valeur de *kintvl* à ce moment précis. Cela signifie que tous les changements de *kintvl* entre les impulsions sont ignorés. Si *kintvl* est nul, il y a un temps d'attente infini jusqu'à la prochaine impulsion. Si *kintvl* est négatif, l'intervalle est compté en nombre d'échantillons plutôt qu'en secondes.

## Exemples

Voici un exemple de l'opcode mpulse. Il utilise le fichier *mpulse.csd* [examples/mpulse.csd].

### Exemple 598. Exemple de l'opcode mpulse.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d          ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o mpulse.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
```

```

kr = 4410
ksmps = 10
nchnls = 1

gkfreq init 0.1

instr 1
  kamp = 10000

  a1 mpulse kamp, gkfreq
  out a1
endin

instr 2
; Assign the value of p4 to gkfreq
gkfreq init p4
endin
</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 11
i 2 2 1 0.05
i 2 4 1 0.01
i 2 6 1 0.005
; only last notes are audible
i 2 8 1 0.003
i 2 10 1 0.002

e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

On peut trouver un autre exemple montrant comment utiliser *mpulse* ici : *mode*.

## Crédits

Ecrit par John ffitch.

Nouveau dans la version 4.08

# mrtmsg

mrtmsg — Send system real-time messages to the MIDI OUT port.

## Description

Envoie des messages système MIDI sur le port MIDI OUT.

## Syntaxe

```
mrtmsg imgtype
```

## Initialisation

*imgtype* -- type du message b:

- 1 envoie un message START (0xFA) ;
- 2 envoie un message CONTINUE (0xFB) ;
- 0 envoie un message STOP (0xFC) ;
- -1 envoie un message SYSTEM RESET (0xFF) ;
- -2 envoie un message ACTIVE SENSING (0xFE)

## Exécution

Envoie un message unique, durant la phase d'initialisation de l'instrument courant. Le paramètre *imgtype* indique le type du message.

## Voir aussi

*mclock*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

# mtof

mtof — Conversion MIDI-fréquence.

## Description

Opcode du greffon emugens.

Convertit un numéro de note MIDI en fréquence en tenant compte de la valeur globale du la3 (A4).

## Syntaxe

```
ifreq mtof imidi  
kfreq mtof kmidi  
ifreqs[] mtof imidis[]  
kfreqs[] mtof kmidis[]
```

## Exécution

*kmidi* / *imidi* -- Numéro de note MIDI (également comme tableau).

*kfreq* / *ifreq* -- Fréquence correspondant à la valeur de note MIDI. Un tableau est retourné si l'entrée est un tableau.

## Exemples

Voici un exemple de l'opcode mtof. Il utilise le fichier *mtof-ftom.csd* [examples/mtof-ftom.csd].

### Exemple 599. Exemple de l'opcode mtof.

```
<CsoundSynthesizer>  
<CsOptions>  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 128  
nchnls = 2  
0dbfs = 1.0  
A4 = 440  
  
instr 1  
  kfreq = mtof(69)  
  printks2 "midi 69 -> %f\n", kfreq  
  
  kmidi = ftom(442)  
  printks2 "freq 442 -> %f\n", kmidi  
  
  kmidi = ftom(442,1)  
  printks2 "freq 442 -> %f rounded\n", kmidi  
  
  kfreq = mtof(kmidi)  
  printks "midi %f -> %f\n", 1, kmidi, kfreq
```

```
imidi = ftom:i(440)
print imidi

ifreq = mtof:i(60)
print ifreq

turnoff
endin

instr 2
imidis[] fillarray 60, 62, 64, 69
ifreqs[] mtof imidis0
printarray ifreqs0, "", "ifreqs0"

kfreqs[] fillarray 220, 440, 880
kmidis[] ftom kfreqs
puts "kfreqs", 1
printarray kmidis, 1, "%.2f", "kmidis"
turnoff
endin

</CsInstruments>
<CsScore>
i 1 0 1
i 2 0 1
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*cpsmidinn, ftom, mton, nton*

## Crédits

Par : Eduardo Moguillansky 2017

Nouveau greffon dans la version 6.11

Version tableau ajoutée dans la 6.13

# mton

mton — Conversion d'un numéro de note MIDI en nom de note.

## Description

Opcode du greffon emugens.

Convertit un numéro de note MIDI en nom de note, avec une précision d'un cent.

## Syntaxe

*Snote* **mton** *kmidi*

*Snote* **mton** *imidi*

## Exécution

*kmidi* / *imidi* -- Numéro de note MIDI.

*Snote* -- Nom de la note.

Exemple de noms de note :

midi	numéro de note
-----	
60	4C
60.4	4C+40
60.5	4C+
60.9	4Db-10
61	4C#
61.5	4D-

## Exemples

Voici un exemple de l'opcode mton. Il utilise le fichier *mton-ntom.csd* [examples/mton-ntom.csd].

### Exemple 600. Exemple de l'opcode mton.

```
<CsoundSynthesizer>
<CsOptions>
--nosound
</CsOptions>
<CsInstruments>

instr 1
  S4 mton ntom("7D+63")
  puts S4, 1

  S1 mton 60
  printf_i "midi 60 = %s \n", 1, S1

  S2 mton fton(442)
```



```

printf_i "442 Hz = %s \n", 1, S2

S3 = mton(48.25)
printf_i "midi 48.25 = %s \n", 1, S3

k1 = ntom("4C")
printf_i "4C = midi %f \n", 1, k1

i2 ntom "4E"
printf_i "4E = %f \n", 1, i2

S5 = mton(ntom("4G+"))
printf_i "roundtrip 4G+: %s \n", 1, S5

turnoff
endin

instr 2
; test i-time and k-time execution
k1 = ntom("4Eb-31")
printf "4Eb-31 = %f \n", 1, k1

i0 ntom "4C+"
printf_i "4C+ = %f \n", 1, i0

i1 = ntom:i("4A")
printf_i "4A = %f \n", 1, i1
turnoff
endin

</CsInstruments>
<CsScore>

i 1 0 1
i 2 0 1

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*cpsmidinn, mtof, from, ntom*

## Crédits

Par : Eduardo Moguillansky 2017

Nouveau greffon dans la version 6.11

# multitap

multitap — Ligne à retard avec plusieurs points de lecture.

## Description

Ligne à retard avec plusieurs points de lecture.

## Syntaxe

```
ares multitap asig [, itime1, igain1] [, itime2, igain2] [...]
```

## Initialisation

Les arguments *itime* et *igain* fixent la position et le gain de chaque point de lecture.

La ligne à retard est remplie par *asig*.

## Exemples

Voici un exemple de l'opcode multitap. Il utilise le fichier *multitap.csd* [examples/multitap.csd]

### Exemple 601. Exemple de l'opcode multitap.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o multitap.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gal init 0

instr 1

asig disk2 "beats.wav", 1,0
outs asig, asig

gal = gal+asig
endin

instr 2

asig multitap gal, 1.2, .5, 1.4, .2
outs asig, asig

gal = 0
endin
```

```
</CsInstruments>
<CsScore>

i 1 .5 .2 ; short sound
i 2 0 3 ; echoes
e
</CsScore>
</CsoundSynthesizer>
```

Cela produit deux délais, l'un de longueur 1.2 et de gain 0.5, et l'autre de longueur 1.4 et de gain 0.2

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1996

# mute

mute — Rend muettes/sonores de nouvelles instances d'un instrument donné.

## Description

Rend muettes/sonores de nouvelles instances d'un instrument donné.

## Syntaxe

```
mute insnum [, iswitch]
```

```
mute "insname" [, iswitch]
```

## Initialisation

*insnum* -- numéro d'instrument. Equivalent à *pI* dans une *instruction i* de partition.

« *insname* » -- Une chaîne de caractères (entre guillemets) représentant un instrument nommé.

*iswitch* (facultatif, 0 par défaut) -- représente un commutateur pour rendre muet/sonore un instrument. Une valeur de 0 rendra muettes de nouvelles instances de l'instrument, tandis que les autres valeurs les rendront sonores. La valeur par défaut est 0.

## Exécution

Toutes les nouvelles instances de l'instrument seront muettes (*iswitch* = 0) ou sonores (*iswitch* différent de 0). Il n'y a aucun problème à rendre muets des instruments muets ou à rendre sonores des instruments sonores. Le mécanisme est le même que celui qui est utilisé par l'*instruction q*. de partition. Par exemple, il est possible de rendre muet depuis la partition et de rendre ensuite sonore depuis un instrument.

L'état Muet/Sonore est indiqué par un message (en fonction du niveau des messages).

## Exemples

Voici en exemple de l'opcode mute. Il utilise le fichier *mute.csd* [examples/mute.csd].

### Exemple 602. Exemple de l'opcode mute.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o mute.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
```

```
ksmps = 32
nchnls = 2
0dbfs = 1

; Mute Instrument #2.
mute 2
; Mute Instrument three.
mute "three"

instr 1

a1 oscils 0.2, 440, 0
outs a1, a1
endin

instr 2 ; gets muted

a1 oscils 0.2, 880, 0
outs a1, a1
endin

instr three ; gets muted

a1 oscils 0.2, 1000, 0
outs a1, a1
endin

</CsInstruments>
<CsScore>

i 1 0 1
i 2 0 1
i "three" 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Nouveau dans la version 4.22

# mvchpf

mvchpf — Emulation d'un filtre passe-haut de Moog contrôlé en tension.

## Description

*mvchpf* est une implémentation numérique du filtre passe-haut de Moog du 4ème ordre (24 dB/oct) écrite à l'origine par Fons Andriaensen. Selon l'auteur, *mvchpf* "... est basé sur le filtre passe-haut contrôlé en tension de Robert Moog, avec une certaine attention portée sur les effets non-linéaires".

## Syntaxe

```
asig mvchpf ain, xcf[, istor]
```

## Initialisation

*istor* -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*ain* -- signal d'entrée.

*xcf* -- fréquence de coupure du filtre. L'intervalle utile est d'environ six octaves autour du do médian (pch 8.00).

## Exemples

Voici un exemple de l'opcode mvchpf. Il utilise le fichier *mvchpf.csd* [examples/mvchpf.csd].

### Exemple 603. Exemple de l'opcode mvchpf.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>
0dbfs = 1

instr 1
  kenv linen p4,0.1,p3,0.1
  ain rand kenv
  kfr expon 220, p3, 1760
  asig mvchpf ain,kfr
    out asig
endin

</CsInstruments>
```

```
<CsScore>  
i1 0 5 0.9  
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : Fons Andriaensen,  
Porté dans Csound par Victor Lazzarini  
Janvier 2016

Nouveau dans la version 6.07

Janvier 2016.

# mvclpf1

mvclpf1 — Emulation d'un filtre passe-bas de Moog contrôlé en tension.

## Description

*mvclpf1* est une implémentation numérique du filtre en échelle de Moog du 4ème ordre (24 dB/oct) écrite à l'origine par Fons Andriaensen. Selon l'auteur, *mvclpf1* "est d'une conception simpliste et ne prétend aucunement s'approcher de l'objet réel". Il utilise une approximation grossière de la résistance non-linéaire seulement dans la première section du filtre. [...] C'est un filtre passe-bas à 24 dB/oct, économique (en termes d'utilisation CPU), à vocation généraliste, qui peut être utile.

## Syntaxe

```
asig mvclpf1 ain, xcf, xres[, istor]
```

## Initialisation

*istor* -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*ain* -- signal d'entrée.

*xcf* -- fréquence de coupure du filtre. L'intervalle utile est d'environ six octaves autour du do médian (pch 8.00).

*xres* -- résonance, limitée à l'intervalle [0, 1].

## Exemples

Voici un exemple de l'opcode mvclpf1. Il utilise le fichier *mvclpf1.csd* [examples/mvclpf1.csd].

### Exemple 604. Exemple de l'opcode mvclpf1.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>
0dbfs = 1

instr 1
  kenv linen p4, 0.1, p3, 0.1
  ain rand kenv
  kfr expon 220, p3, 1760
```



```
asig mvclpfl ain,kfr,0.9
    out asig
endin

</CsInstruments>
<CsScore>
i1 0 5 0.9
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Fons Andriaensen,  
Porté dans Csound par Victor Lazzarini  
Janvier 2016

Nouveau dans la version 6.07

Janvier 2016.

# mvclpf2

mvclpf2 — Emulation d'un filtre passe-bas de Moog contrôlé en tension.

## Description

*mvclpf2* est une implémentation numérique du filtre en échelle de Moog du 4ème ordre (24 dB/oct) écrite à l'origine par Fons Andriaensen. Selon l'auteur, *mvclpf2* "utilise cinq éléments non-linéaires en entrée et dans les quatre sections du filtre. Il utilise la dérivée de la non-linéarité (pour laquelle  $1 / (1 + x * x)$  est une approximation raisonnable). Le principal avantage de ceci est qu'une seule évaluation de la fonction non-linéaire est requise par section".

## Syntaxe

```
asig mvclpf2 ain, xcf, xres[, istor]
```

## Initialization

*istor* -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*ain* -- signal d'entrée.

*xcf* -- fréquence de coupure du filtre. L'intervalle utile est d'environ six octaves autour du do médian (pch 8.00).

*xres* -- résonance, limitée à l'intervalle [0, 1].

## Exemples

Voici un exemple de l'opcode mvclpf2. Il utilise le fichier *mvclpf2.csd* [examples/mvclpf2.csd].

### Exemple 605. Exemple de l'opcode mvclpf2.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>
0dbfs = 1

instr 1
  kenv linen p4,0.1,p3,0.1
  ain rand kenv
  kfr expon 220, p3, 1760
```

```
asig mvclpfl ain,kfr,0.9
    out asig
endin

</CsInstruments>
<CsScore>
i1 0 5 0.9
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Fons Andriaensen,  
Porté dans Csound par Victor Lazzarini  
Janvier 2016

Nouveau dans la version 6.07

Janvier 2016.

# mvclpf3

mvclpf3 — Emulation d'un filtre passe-bas de Moog contrôlé en tension.

## Description

*mvclpf3* est une implémentation numérique du filtre en échelle de Moog du 4ème ordre (24 dB/oct) écrite à l'origine par Fons Andriaensen. Selon l'auteur, *mvclpf3* "est basé sur *mvclpf2* avec deux différences. Il utilise la technique décrite par Stilson et Smith pour étendre l'intervalle de la constante Q et la fréquence d'échantillonnage interne est doublée, ce qui donne une meilleure approximation du comportement non-linéaire dans les hautes fréquences. Cette version a un Q élevé sur tout l'intervalle de fréquences et oscillera jusqu'à 10 kHz, tandis que les deux versions précédentes ont un Q décroissant dans les hautes fréquences. *mvclpf3* présente un accord raisonnable et peut être 'joué' en VCO jusqu'à 5 kHz".

## Syntaxe

```
asig mvclpf3 ain, xcf, xres[, istor]
```

## Initialisation

*istor* -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*ain* -- signal d'entrée.

*xcf* -- fréquence de coupure du filtre. L'intervalle utile est d'environ six octaves autour du do médian (pch 8.00).

*xres* -- résonance, limitée à l'intervalle [0, 1].

## Exemples

Voici un exemple de l'opcode mvclpf3. Il utilise le fichier *mvclpf3.csd* [examples/mvclpf3.csd].

### Exemple 606. Exemple de l'opcode mvclpf3.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>
0dbfs = 1

instr 1
  kenv linen p4,0.1,p3,0.1
```

```
ain rand kenv  
kfr expon 220, p3, 1760  
asig mvclpf3 ain,kfr,0.9  
out asig  
endin  
  
</CsInstruments>  
<CsScore>  
i1 0 5 0.9  
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : Fons Andriaensen,  
Porté dans Csound par Victor Lazzarini  
Janvier 2016

Nouveau dans la version 6.07

Janvier 2016.

# mvclpf4

mvclpf4 — Emulation d'un filtre passe-bas de Moog contrôlé en tension.

## Description

*mvclpf4* est une implémentation numérique du filtre en échelle de Moog du 4ème ordre (24 dB/oct) écrite à l'origine par Fons Andriaensen. C'est une version de l'opcode *mvclpf3* avec quatre sorties pour les réponses en fréquence à 6dB, 12 dB, 18 dB et 24 dB/octave.

## Syntaxe

```
asig1, asig2, asig3, asig4 mvclpf4 ain, xcf, xres[, istor]
```

## Initialisation

*istor* -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig1* -- sortie de la réponse passe-bas à 6 dB/oct.

*asig2* -- sortie de la réponse passe-bas à 12 dB/oct.

*asig3* -- sortie de la réponse passe-bas à 18 dB/oct.

*asig4* -- sortie de la réponse passe-bas à 24 dB/oct.

*ain* -- signal d'entrée.

*xcf* -- fréquence de coupure du filtre. L'intervalle utile est d'environ six octaves autour du do médian (pch 8.00).

*xres* -- résonance, limitée à l'intervalle [0, 1].

## Exemples

Voici un exemple de l'opcode mvclpf4. Il utilise le fichier *mvclpf4.csd* [examples/mvclpf4.csd].

### Exemple 607. Exemple de l'opcode mvclpf4.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
-odac  
</CsOptions>
```

```

<CsInstruments>
0dbfs = 1

instr 1
  asig[] init 4
  kenv linen p4,0.1,p3,0.1
  ain rand kenv
  kfr expon 220, p3, 1760
  asig[0],asig[1],asig[2],asig[3] mvclpf4 ain,kfr,0.9
  out asig[p5]
endin

</CsInstruments>
<CsScore>
i1 0 5 0.9 0
i1 + 5 0.9 1
i1 + 5 0.9 2
i1 + 5 0.9 3
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Fons Andriaensen,  
 Porté dans Csound par Victor Lazzarini  
 Janvier 2016

Nouveau dans la version 6.07

Janvier 2016.

# mxadsr

`mxadsr` — Calcule l'enveloppe ADSR classique en utilisant le mécanisme de *expsegr*.

## Description

Calcule l'enveloppe ADSR classique en utilisant le mécanisme de *expsegr*.

## Syntaxe

```
ares mxadsr iatt, idec, islev, irel [, idel] [, ireltim]
```

```
kres mxadsr iatt, idec, islev, irel [, idel] [, ireltim]
```

## Initialisation

*iatt* -- durée de l'attaque (attack)

*idec* -- durée de la première chute (decay)

*islev* -- niveau d'entretien (sustain)

*irel* -- durée de la chute (release)

*idel* (facultatif, 0 par défaut) -- délai de niveau zéro avant le démarrage de l'enveloppe

*ireltim* (facultatif, -1 par défaut) -- Contrôle la durée du relâchement après la réception d'un évènement MIDI note-off. S'il est inférieur à zéro, la durée de relâchement la plus longue de l'instrument courant est utilisée. S'il est nul ou positif, la valeur donnée sera utilisée comme durée de relâchement. Sa valeur par défaut est -1. (Nouveau dans Csound 3.59 - pas encore entièrement testé).

## Exécution

L'enveloppe évolue dans l'intervalle de 0 à 1 et peut être changée d'échelle par la suite. Voici une description de l'enveloppe :

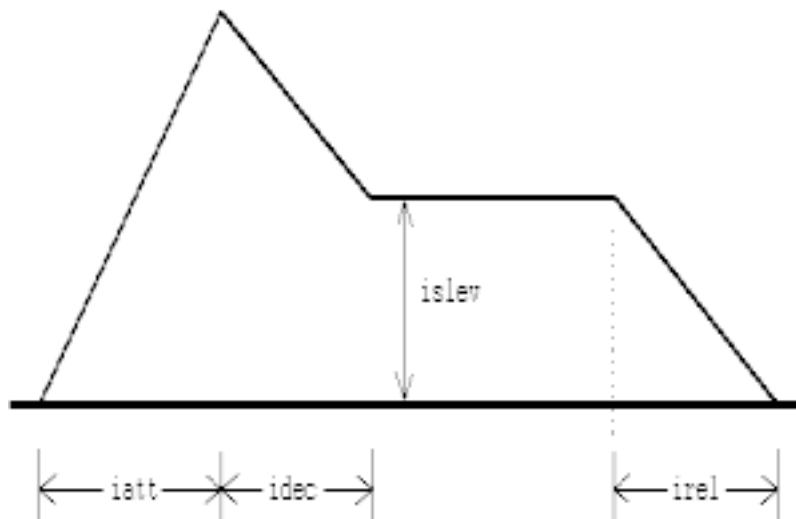




Image d'une enveloppe ADSR.

La longueur de la période d'entretien est calculée à partir de la longueur de la note. C'est pourquoi *adsr* n'est pas adapté au traitement des événements MIDI. L'opcode *madsr* utilise le mécanisme de *linsegr*, et peut donc être utilisé dans les applications MIDI. L'opcode *mxadsr* est identique à *madsr* sauf qu'il utilise des segments exponentiels plutôt que linéaires.

On peut utiliser d'autres enveloppes préfabriquées pour lancer un segment de relâchement à la réception d'un message note off, comme *linsegr* et *expsegr*, ou bien l'on peut construire des enveloppes plus complexes au moyen de *xtratim* et de *release*. Noter qu'il n'est pas nécessaire d'utiliser *xtratim* avec *mxadsr*, car la durée est allongée automatiquement.

*mxadsr* est nouveau dans la version 3.51 de Csound.

## Exemples

Voici un exemple de l'opcode *mxadsr*. Il utilise le fichier *mxadsr.csd* [examples/mxadsr.csd].

### Exemple 608. Exemple de l'opcode *mxadsr*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0   ;;realtime audio out and realtime midi in
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o mxadsr.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

icps cpsmidi
iamp ampmidi .5

kenv mxadsr 0.5, 0, 1, 0.5
asig pluck kenv, icps, icps, 2, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 2 0 4096 10 1

f0 30 ;runs 30 seconds
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*linsegr*, *expsegr*, *envlpxr*, *mxadsr*, *madsr*, *adsr*, *expon*, *expsegr*, *expsega* *line*, *linsegr*, *xtratim*

## Crédits

Auteur : John ffitch

Novembre 2002. Merci à Rasmus Ekman pour avoir documenté le paramètre *ireltim*.

Novembre 2003. Merci à Kanata Motohashi pour avoir fixé le lien vers l'opcode *linsegr*.

# nchnls

nchnls — Fixe le nombre de canaux de la sortie audio.

## Description

Ces instructions sont des *affectations* de valeurs globales réalisées au début d'un orchestre, avant que tout bloc d'instrument ne soit défini. Leur fonction est de fixer certaines *variables* dont le nom est un mot réservé et qui sont nécessaires à l'exécution. Une fois fixés, ces mots réservés peuvent être utilisés dans des expressions n'importe où dans l'orchestre.

## Syntaxe

```
nchnls = iarg
```

## Initialisation

*nchnls* = (facultatif) -- fixe le nombre de canaux de la sortie audio à *iarg*. (1 = mono, 2 = stéréo, 4 = quadriphonique.) La valeur par défaut est 1 (mono).

De plus, toute *variable globale* peut être initialisée par une *instruction de la période d'initialisation* n'importe où avant la première *instruction instr*. Toutes les affectations ci-dessus sont exécutées dans l'instrument 0 (passe-i seulement) au début de l'exécution réelle.

## Exemples

Voici un exemple de l'opcode nchnls. Il utilise le fichier *nchnls.csd* [examples/nchnls.csd].

### Exemple 609. Exemple de l'opcode nchnls.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -idac   ;;realtime audio I/O
; For Non-realtime ouput leave only the line below:
; nchnls.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2      ;two channels out
0dbfs = 1

instr 1

ainr1, ainr ins    ;grab your mic and sing
adel linseg 0, p3*.5, 0.02, p3*.5, 0 ;max delay time = 20ms
aoutl flanger ainr1, adel, .7
aoutr flanger ainr1, adel*2, .8
fout "in_s.wav", 14, aoutl, aoutr ;write to stereo file,
```

```
    outs aoutl, aoutr    ;16 bits with header

    endin
</CsInstruments>
<CsScore>

i 1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*kr, ksmpr, sr*

# nchnls\_hw

nchnls\_hw — Retourne le nombre de canaux audio du matériel utilisé.

## Description

Retourne le nombre maximum de canaux audio du matériel utilisé. Ca ne correspond pas forcément au nombre de canaux utilisés par Csound (fixé par *nchnls* et *nchnls\_i*).

## Syntaxe

```
idacc, iadcc nchnls_hw
```

## Initialisation

Lors de l'initialisation, *idacc* contiendra le nombre de canaux du périphérique de sortie et *iadcc* le nombre de canaux d'entrée. Ca correspond aux périphériques matériels sélectionnés/utilisés.

## Crédits

Auteur : Victor Lazzarini  
2016

Dans la version 6.07

# nchnls\_i

nchnls\_i — Fixe le nombre de canaux de l'entrée audio.

## Description

Ces instructions sont des *affectations* de valeurs globales réalisées au début d'un orchestre, avant que tout bloc d'instrument ne soit défini. Leur fonction est de fixer certaines *variables* dont le nom est un mot réservé et qui sont nécessaires à l'exécution. Une fois fixés, ces mots réservés peuvent être utilisés dans des expressions n'importe où dans l'orchestre.

## Syntaxe

```
nchnls_i = iarg
```

## Initialisation

*nchnls\_i* = (facultatif) -- fixe le nombre de canaux de l'entrée audio à *iarg*. (1 = mono, 2 = stéréo, 4 = quadriphonique.) La valeur par défaut est celle de *nchnls*.

De plus, toute *variable globale* peut être initialisée par une *instruction de la période d'initialisation* n'importe où avant la première *instruction instr*. Toutes les affectations ci-dessus sont exécutées dans l'instrument 0 (passe-i seulement) au début de l'exécution réelle.

## Exemples

Voici un exemple de l'opcode nchnls\_i. Il utilise le fichier *nchnls\_i.csd* [examples/nchnls\_i.csd].

### Exemple 610. Exemple de l'opcode nchnls\_i.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -idac ;;realtime audio I/O
; For Non-realtime ouput leave only the line below:
; nchnls_i.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 32
nchnls  = 2 ;2 channels out
0dbfs   = 1
nchnls_i = 4 ;4 channels in

instr 1 ;4 channels in, two channels out

ain1, ain2, ain3, ain4 inq ;grab your mics and sing

adel1    linseg 0, p3*.5, 0.02, p3*.5, 0    ;max delay time = 20ms
adel2    linseg 0.02, p3*.5, 0, p3*.5, 0.02 ;max delay time = 20ms
```

```
aoutl flanger ain1, adel, .7
aoutr flanger ain2, adel*2, .8
aoutla flanger ain3, adel2, .9
aoutra flanger ain4, adel2*2, .5
;write to quad file, 16 bits with header
      fout "in_4.wav", 14, aoutl, aoutr, aoutla, aoutra
      outs (aoutl+aoutla)*.5, (aoutr+aoutra)*.5 ;stereo out

endin
</CsInstruments>
<CsScore>

i 1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*kr, ksmps, nchnls, sr*

# nestedap

nestedap — Trois différents filtres passe-tout imbriqués.

## Description

Trois différents filtres passe-tout imbriqués, utiles pour implémenter des réverbérations.

## Syntaxe

```
ares nestedap asig, imode, imaxdel, idel1, igain1 [, idel2] [, igain2] \  
      [, idel3] [, igain3] [, istor]
```

## Initialisation

*imode* -- mode opératoire du filtre :

- 1 = simple filtre passe-tout
- 2 = filtre passe-tout imbriqué
- 3 = double filtre passe-tout imbriqué

*idel1*, *idel2*, *idel3* -- retards des étages du filtre. Les retards sont en secondes et doivent être supérieurs à zéro. *idel1* doit être supérieur à la somme de *idel2* et de *idel3*.

*igain1*, *igain2*, *igain3* -- gain des étages du filtre.

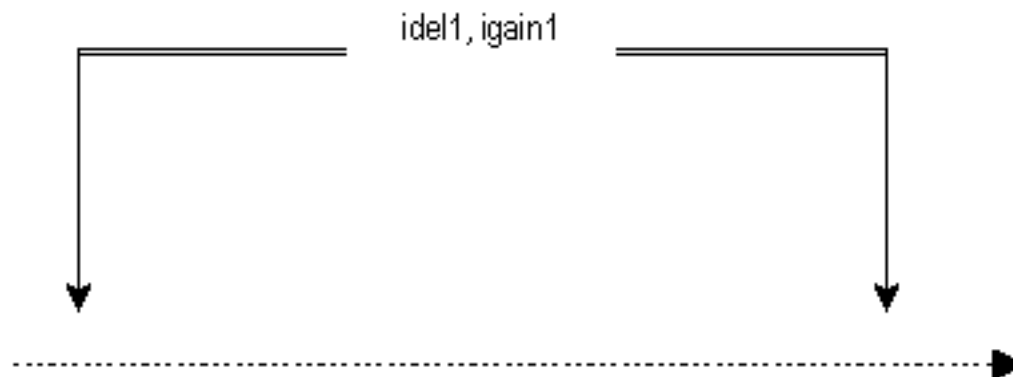
*imaxdel* -- deviendra nécessaire lorsque les retards de taux-k auront été implémentés. N'est pas utilisé actuellement.

*istor* -- L'initialisation est ignorée s'il est différent de zéro (0 par défaut).

## Exécution

*asig* -- signal d'entrée

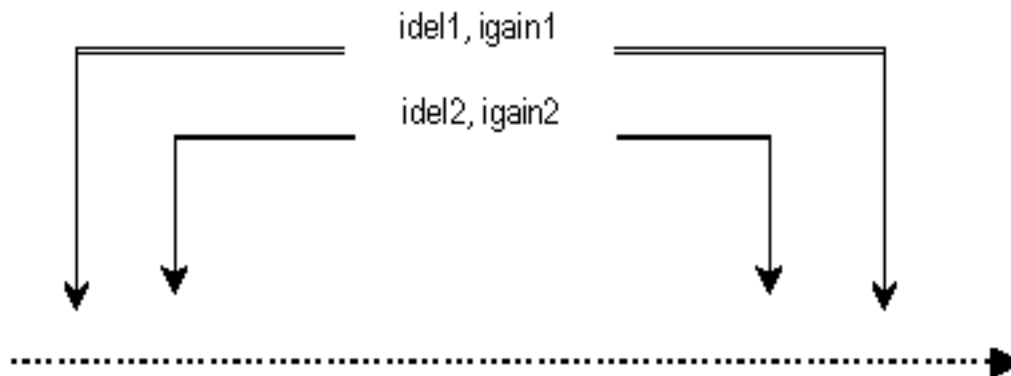
Si *imode* = 1, le filtre prend la forme :



Représentation du filtre d'imode 1.

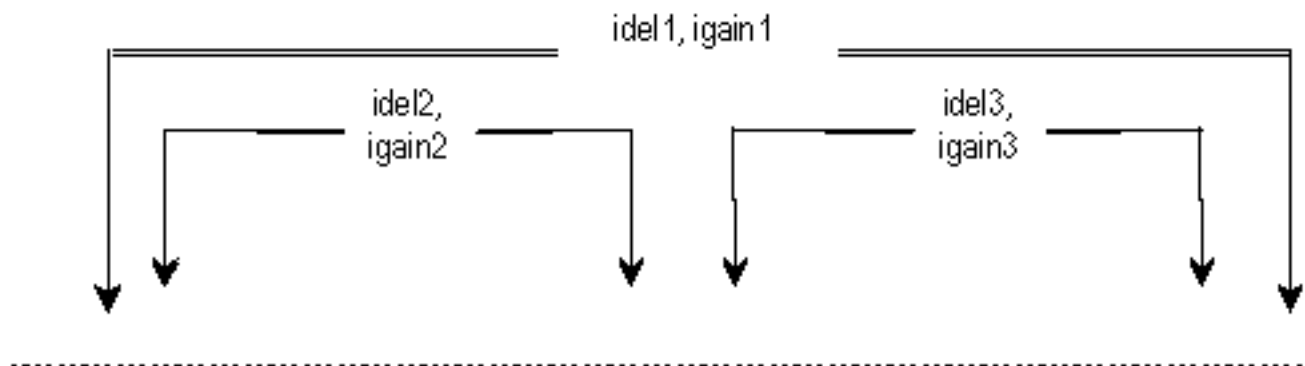


Si *imode* = 2, le filtre prend la forme :



Représentation du filtre d'imode 2.

Si *imode* = 3, le filtre prend la forme :



Représentation du filtre d'imode 3.

## Exemples

Voici un exemple de l'opcode *nestedap*. Il utilise les fichiers *nestedap.csd* [examples/nestedap.csd] et *beats.wav* [examples/beats.wav].

### Exemple 611. Exemple de l'opcode *nestedap*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc    -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o nestedap.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
```

```

nchnls = 2

instr 5
  insnd      =      p4
  gasig      diskin2  insnd, 1
endin

instr 10
  imax       =      1
  idel1      =      p4/1000
  igain1     =      p5
  idel2      =      p6/1000
  igain2     =      p7
  idel3      =      p8/1000
  igain3     =      p9
  idel4      =      p10/1000
  igain4     =      p11
  idel5      =      p12/1000
  igain5     =      p13
  idel6      =      p14/1000
  igain6     =      p15

  afdbk      init 0

  aout1      nestedap gasig+afdbk*.4, 3, imax, idel1, igain1, idel2, igain2, idel3, igain3

  aout2      nestedap aout1, 2, imax, idel4, igain4, idel5, igain5

  aout       nestedap aout2, 1, imax, idel6, igain6

  afdbk      butterlp aout, 1000

              outs gasig+(aout+aout1)/2, gasig-(aout+aout1)/2

gasig      =      0
endin

</CsInstruments>
<CsScore>

f1 0 8192 10 1

; Diskin
;  Sta  Dur  Soundin
i5 0 3 "beats.wav"

; Reverb
;  St  Dur  Del1 Gn1  Del2 Gn2  Del3 Gn3  Del4 Gn4  Del5 Gn5  Del6 Gn6
i10 0 4 97 .11 23 .07 43 .09 72 .2 53 .2 119 .3
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Hans Mikelson  
Février 1999

Nouveau dans la version 3.53 de Csound

L'exemple a été mis à jour en mai 2002, grâce à Hans Mikelson

# nlfilt

nlfilt — Un filtre avec un effet non-linéaire.

## Description

Implémente le filtre :

$$Y\{n\} = a Y\{n-1\} + b Y\{n-2\} + d Y^2\{n-L\} + X\{n\} - C$$

décrit dans Dobson et Fitch (ICMC'96)

## Syntaxe

ares **nlfilt** ain, ka, kb, kd, kC, kL

## Exécution

1. Effet non-linéaire. L'ensemble de définition des paramètres est :

a = b = 0  
d = 0.8, 0.9, 0.7  
C = 0.4, 0.5, 0.6  
L = 20

Cela affecte surtout le registre grave mais il y a des effets audibles sur tout le registre. Peut être utile pour colorer des sons de percussion et pour renforcer arbitrairement des notes.

2. Passe-bas non-linéaire. L'ensemble de définition des paramètres est :

a = 0.4  
b = 0.2  
d = 0.7  
C = 0.11  
L = 20, ... 200

Cette variante présente des problèmes d'instabilité mais l'effet est plus prononcé dans le registre grave, sinon elle ressemble beaucoup à un filtre en peigne. De courtes valeurs de *L* peuvent renforcer l'attaque du son.

3. Passe-haut non-linéaire. L'ensemble de définition des paramètres est :

a = 0.35  
b = -0.3  
d = 0.95  
C = 0.2, ... 0.4  
L = 200

4. Passe-haut non-linéaire. L'ensemble de définition des paramètres est :

a = 0.7  
b = -0.2, ... 0.5  
d = 0.9  
C = 0.12, ... 0.24  
L = 500, 10

La version passe-haut est moins sujette aux oscillations. Elle ajoute de la brillance dans le registre medium-aigu. Avec un long délai  $L$  cela ressemble un peu à de la réverbération, tandis qu'avec de petites valeurs apparaissent des régions comme des formants. Il y a des changements de couleur arbitraires et des résonances lorsque la hauteur change. Fonctionne bien avec des notes seules.



## Avertissement

Les ensembles des valeurs "utiles" des paramètres n'ont pas encore été explorés.

## Exemples

Voici un exemple de l'opcode `nlfilt`. Il utilise le fichier `nlfilt.csd` [examples/nlfilt.csd].

### Exemple 612. Exemple de l'opcode `nlfilt`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o nlfilt.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;unfiltered noise

asig rand .7
outs asig, asig

endin

instr 2 ;filtered noise

ka = p4
kb = p5
kd = p6
kC = p7
kL = p8
asig rand .3
afilt nlfilt asig, ka, kb, kd, kC, kL
asig clip afilt, 2, .9
outs asig, asig
```

```

endin
</CsInstruments>
<CsScore>

i 1 0 2      ; unfiltersd

;
i 2 2 2 0 0 0.8 0.5 20 ; non-linear effect
i 2 + 2 .4 0.2 0.7 0.11 200 ; low=paas with non-linear
i 2 + 2 0.35 -0.3 0.95 0.1 200 ; high-pass with non-linear
i 2 + 2 0.7 -0.2 0.9 0.2 20 ; high-pass with non-linear

e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch  
 Université de Bath/Codemist Ltd.  
 Bath, UK  
 1997

Nouveau dans la version 3.44

# nlfilt2

nlfilt2 — Un filtre avec effet non-linéaire et protection contre l'explosion.

## Description

Implémente le filtre :

$$Y\{n\} = \tanh(a Y\{n-1\} + b Y\{n-2\} + d Y^2\{n-L\} + X\{n\} - C)$$

décrit dans Dobson et Fitch (ICMC'96) et selon les modifications de Risto Holopainen.

## Syntaxe

ares **nlfilt2** ain, ka, kb, kd, kC, kL

## Exécution

1. Effet non-linéaire. L'ensemble de définition des paramètres est :

a = b = 0  
d = 0.8, 0.9, 0.7  
C = 0.4, 0.5, 0.6  
L = 20

Cela affecte surtout le registre grave mais il y a des effets audibles sur tout le registre. Peut être utile pour colorer des sons de percussion et pour renforcer arbitrairement des notes.

2. Passe-bas non-linéaire. L'ensemble de définition des paramètres est :

a = 0.4  
b = 0.2  
d = 0.7  
C = 0.11  
L = 20, ... 200

Cette variante présente des problèmes d'instabilité mais l'effet est plus prononcé dans le registre grave, sinon elle ressemble beaucoup à un filtre en peigne. De courtes valeurs de *L* peuvent renforcer l'attaque du son.

3. Passe-haut non-linéaire. L'ensemble de définition des paramètres est :

a = 0.35  
b = -0.3  
d = 0.95  
C = 0.2, ... 0.4  
L = 200

4. Passe-haut non-linéaire. L'ensemble de définition des paramètres est :

a = 0.7  
b = -0.2, ... 0.5  
d = 0.9  
C = 0.12, ... 0.24  
L = 500, 10

La version passe-haut est moins sujette aux oscillations. Elle ajoute de la brillance dans le registre medium-aigu. Avec un long délai  $L$  cela ressemble un peu à de la réverbération, tandis qu'avec de petites valeurs apparaissent des régions comme des formants. Il y a des changements de couleur arbitraires et des résonances lorsque la hauteur change. Fonctionne bien avec des notes seules.



## Avertissement

Les ensembles des valeurs "utiles" des paramètres n'ont pas encore été explorés.

## Exemples

Voici un exemple de l'opcode `nlfilt2`. Il utilise le fichier `nlfilt2.csd` [examples/nlfilt2.csd].

### Exemple 613. Exemple de l'opcode `nlfilt2`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o nlfilt.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;unfiltered noise

asig rand .7
outs asig, asig

endin

instr 2 ;filtered noise

ka = p4
kb = p5
kd = p6
kC = p7
kL = p8
asig rand .3
afilt nlfilt2 asig, ka, kb, kd, kC, kL
outs afilt, afilt

endin
</CsInstruments>
```

```
<CsScore>

i 1 0 2      ; unfiltered

;          a      b      d      C      L
i 2 2 2 0      0      0.8  0.5  20 ; non-linear effect
i 2 + 2 .4      0.2  0.7  0.11 200 ; low-pass with non-linear
i 2 + 2 0.35 -0.3 0.95 0.1  200 ; high-pass with non-linear
i 2 + 2 0.7 -0.2 0.9  0.2  20  ; high-pass with non-linear

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
2012

Nouveau dans la version 5.19



# noise

noise — Un générateur de bruit blanc avec un filtre passe-bas à RII.

## Description

Un générateur de bruit blanc avec un filtre passe-bas à RII.

## Syntaxe

```
ares noise xamp, kbeta
```

## Exécution

*xamp* -- amplitude de la sortie finale

*kbeta* -- beta du filtre passe-bas. Doit être compris entre -1 et 1, à l'exclusion des extrémités (intervalle ouvert).

L'équation du filtre est :

$$y_n = \sqrt{(1 - \beta^2)} * x_n + \beta y_{(n-1)}$$

où  $x_n$  est le bruit blanc original et  $y_n$  est le bruit filtré. Plus  $\beta$  est élevé, plus basse est la fréquence de coupure du filtre. La fréquence de coupure vaut approximativement  $sr * ((1 - kbeta) / 2)$ .

## Exemples

Voici un exemple de l'opcode noise. Il utilise le fichier *noise.csd* [examples/noise.csd].

### Exemple 614. Exemple de l'opcode noise.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o noise.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
```

```

nchnls = 2
0dbfs = 1

instr 1

kbeta line -0.9999, p3, 0.9999 ;change beta value between -1 to 1
asig noise .3, kbeta
asig clip asig, 2, .9 ;clip signal
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 10

e
</CsScore>
</CsoundSynthesizer>

```

Voici un exemple de l'opcode noise dans lequel on contrôle le paramètre *kbeta* au moyen d'une interface graphique. Il utilise le fichier *noise-2.csd* [examples/noise-2.csd].

### Exemple 615. Exemple de l'opcode noise contrôlé par une interface graphique.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      ; -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o noise.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

FLpanel "noise", 200, 50, -1, -1
gkbeta, gislider1 FLslider "kbeta", -1, 1, 0, 5, -1, 180, 20, 10, 10
FLpanelEnd
FLrun

instr 1
iamp = 0dbfs / 4 ; Peaks 12 dB below 0dbfs
print iamp

a1 noise iamp, gkbeta*0.9999
printk2 gkbeta*0.9999
outs a1,a1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one minute.
i 1 0 60

e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch  
Université de Bath, Codemist. Ltd.  
Bath, UK  
Décembre 2000

Nouveau dans la version 4.10 de Csound

# noteoff

noteoff — Envoie un message note off sur le port MIDI OUT.

## Description

Envoie un message note off sur le port MIDI OUT.

## Syntaxe

```
noteoff ichn, inum, ivel
```

## Initialisation

*ichn* -- numéro de canal MIDI (1-16)

*inum* -- numéro de note (0-127)

*ivel* -- vélocité (0-127)

## Exécution

*noteon* (note on au taux-i) et *noteoff* (note off au taux-i) sont les opcodes MIDI OUT les plus simples. *noteon* envoie un message note on sur le port MIDI OUT et *noteoff* envoie un message note off. Un opcode *noteon* doit toujours être suivi par un *noteoff* avec les mêmes numéros de canal et de note, dans le même instrument, sinon la note sera jouée indéfiniment.

Ces opcodes *noteon* et *noteoff* ne sont utiles que si l'on introduit une instruction *timeout* pour jouer une note MIDI avec une durée non nulle. Dans la plupart des cas, il vaut mieux utiliser *noteondur* et *noteondur2*.

## Voir aussi

*noteon*, *noteondur*, *noteondur2*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# noteon

noteon — Envoie un message note on sur le port MIDI OUT.

## Description

Envoie un message note on sur le port MIDI OUT.

## Syntaxe

```
noteon ichn, inum, ivel
```

## Initialisation

*ichn* -- numéro de canal MIDI (1-16)

*inum* -- numéro de note (0-127)

*ivel* -- vélocité (0-127)

## Exécution

*noteon* (note on au taux-i) et *noteoff* (note off au taux-i) sont les opcodes MIDI OUT les plus simples. *noteon* envoie un message note on sur le port MIDI OUT et *noteoff* envoie un message note off. Un opcode *noteon* doit toujours être suivi par un *noteoff* avec les mêmes numéros de canal et de note, dans le même instrument, sinon la note sera jouée indéfiniment.

Ces opcodes *noteon* et *noteoff* ne sont utiles que si l'on introduit une instruction *timeout* pour jouer une note MIDI avec une durée non nulle. Dans la plupart des cas, il vaut mieux utiliser *noteondur* et *noteondur2*.

## Voir aussi

*noteoff*, *noteondur*, *noteondur2*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# noteondur2

`noteondur2` — Envoie un message MIDI note on et note off ayant même numéro de canal, de note et vitesse.

## Description

Envoie un message MIDI note on et note off ayant même numéro de canal, de note et vitesse.

## Syntaxe

```
noteondur2 ichn, inum, ivel, idur
```

## Initialisation

*ichn* -- numéro de canal MIDI (1-16)

*inum* -- numéro de note (0-127)

*ivel* -- vitesse (0-127)

*idur* -- durée de la note en secondes.

## Exécution

*noteondur2* (note on au taux-i avec durée) envoie un message MIDI note on et note off ayant même numéro de canal, de note et vitesse. Le message note off est envoyé *idur* secondes après l'activation de *noteondur2*.

*noteondur* diffère de *noteondur2* en ce que *noteondur* tronque la durée de la note lorsque l'instrument courant est désactivé par la partition ou par le jeu en , tandis que *noteondur2* allonge le temps d'exécution de l'instrument courant jusqu'à ce que *idur* secondes se soient écoulées. Dans le jeu en , il est suggéré d'utiliser *noteondur* aussi pour des durées indéfinies, en donnant une grande valeur à *idur*.

Il peut y avoir n'importe quel nombre d'opcodes *noteondur2* dans le même instrument de Csound, ce qui permet de jouer des accords avec un seul instrument.

## Exemples

Voici un exemple de l'opcode *noteondur2*. Il utilise le fichier *noteondur2.csd* [examples/noteondur2.csd].

### Exemple 616. Exemple de l'opcode *noteondur2*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

Cet exemple génère des notes pour chaque note reçue sur l'entrée MIDI. Il génère des notes MIDI sur la sortie MIDI de Csound, si bien qu'il faut y connecter quelque chose.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```

; Audio out   Audio in   No messages
-odac         -iadc      -d          -M0  -Q1;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

instr 1

ifund notnum
ivel veloc
idur = 1

;chord with single key
noteondur2 1, ifund,  ivel, idur
noteondur2 1, ifund+3, ivel, idur
noteondur2 1, ifund+7, ivel, idur
noteondur2 1, ifund+9, ivel, idur

endin

</CsInstruments>
<CsScore>
; Dummy ftable
f 0 60
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*noteoff, noteon, noteondur, midion, midion2*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# noteondur

noteondur — Envoie un message MIDI note on et note off ayant même numéro de canal, de note et velocity.

## Description

Envoie un message MIDI note on et note off ayant même numéro de canal, de note et velocity.

## Syntaxe

```
noteondur ichn, inum, ivel, idur
```

## Initialisation

*ichn* -- numéro de canal MIDI (1-16)

*inum* -- numéro de note (0-127)

*ivel* -- vitesse (0-127)

*idur* -- durée de la note en secondes.

## Exécution

*noteondur* (note on au taux-i avec durée) envoie un message MIDI note on et note off ayant même numéro de canal, de note et velocity. Le message note off est envoyé *idur* secondes après l'activation de *noteondur*.

*noteondur* diffère de *noteondur2* en ce que *noteondur* tronque la durée de la note lorsque l'instrument courant est désactivé par la partition ou par le jeu en , tandis que *noteondur2* allonge le temps d'exécution de l'instrument courant jusqu'à ce que *idur* secondes se soient écoulées. Dans le jeu en , il est suggéré d'utiliser *noteondur* aussi pour des durées indéfinies, en donnant une grande valeur à *idur*.

Il peut y avoir n'importe quel nombre d'opcodes *noteondur* dans le même instrument de Csound, ce qui permet de jouer des accords avec un seul instrument.

## Exemples

Voici un exemple de l'opcode *noteondur*. Il utilise le fichier *noteondur.csd* [examples/noteondur.csd].

### Exemple 617. Exemple de l'opcode *noteondur*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

Cet exemple génère des notes pour chaque note reçue sur l'entrée MIDI. Il génère des notes MIDI sur la sortie MIDI de Csound, si bien qu'il faut y connecter quelque chose.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in    No messages
```



```

-odac          -iadc      -d          -M0  -Q1;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

instr 1  ;Turned on by MIDI notes on channel 1

    ifund    notnum
    ivel     veloc
    idur     = 1

    ;chord with single key
    noteondur 1, ifund,   ivel, idur
    noteondur 1, ifund+3, ivel, idur
    noteondur 1, ifund+7, ivel, idur
    noteondur 1, ifund+9, ivel, idur

endin

</CsInstruments>
<CsScore>
; Play Instrument #1 for 60 seconds.

i1 0 60

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*noteoff, noteon, noteondur2, midion, midion2*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# notnum

notnum — Donne un numéro de note à partir d'un évènement MIDI.

## Description

Donne un numéro de note à partir d'un évènement MIDI.

## Syntaxe

ival notnum

## Exécution

Donne la valeur de l'octet MIDI (0 - 127) représentant le numéro de note de l'évènement courant.

## Exemples

Voici un exemple de l'opcode notnum. Il utilise le fichier *notnum.csd* [examples/notnum.csd].

### Exemple 618. Exemple de l'opcode notnum.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -M0 -+rtmidi=virtual ;;realtime audio out with virtual MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

massign 1, 1 ; set MIDI channel 1 to play instr 1

instr 1

iNum notnum
print iNum
; Convert MIDI note number to Hz
iHz = (440.0*exp(log(2.0)*((iNum)-69.0)/12.0))
aosc oscil 0.6, iHz, 1
outs aosc, aosc

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave

f 0 60 ;play 60 seconds

e
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*aftouch, ampmidi, cpsmidi, cpsmidib, midictrl, octmidi, octmidib, pchbend, pchmidi, pchmidib, veloc*

## Crédits

Auteur : Barry L. Vercoe - Mike Berry  
MIT - Mills  
Mai 1997

Exemple écrit par David Akbari.

# nreverb

nreverb — Une réverbération constituée de 6 filtres en peigne passe-bas parallèles.

## Description

Réverbération constituée de 6 filtres en peigne passe-bas parallèles suivis de 5 filtres passe-tout en série. *nreverb* remplace *reverb2* (version 3.48) et ainsi les deux opcodes sont identiques.

## Syntaxe

```
ares nreverb asig, ktime, khdif [, iskip] [,inumCombs] [, ifnCombs] \  
    [, inumAlpas] [, ifnAlpas]
```

## Initialisation

*iskip* (facultatif, 0 par défaut) -- L'initialisation est ignorée si ce paramètre est présent et différent de zéro.

*inumCombs* (facultatif) -- nombre de constantes de filtre dans le filtre en peigne. S'il est omis, les valeurs par défaut sont les constantes de *nreverb*. Nouveau dans la version 4.09 de Csound.

*ifnCombs* - table de fonction contenant *inumCombs* valeurs temporelles du filtre en peigne, suivies du même nombre de valeurs de gain. La table ne doit pas être normalisée (utiliser un numéro de fgen négatif). Les valeurs temporelles positives sont en secondes. Les valeurs temporelles sont converties en interne en nombre d'échantillons, puis fixées au nombre premier supérieur le plus proche. Si le temps est négatif, il est directement interprété en trames d'échantillons, et aucun traitement n'est effectué (à part le changement de signe). Nouveau dans la version 4.09 de Csound.

*inumAlpas*, *ifnAlpas* (facultatif) -- comme *inumCombs/ifnCombs*, pour le filtre passe-tout. Nouveau dans Csound 4.09.

## Exécution

Le signal d'entrée *asig* est réverbéré pendant *ktime* secondes. Le paramètre *khdif* contrôle la diffusion des hautes fréquences. Les valeurs de *khdif* doivent être comprises entre 0 et 1. Si *khdif* vaut 0 toutes les fréquences décroissent à la même vitesse. Si *khdif* vaut 1, les hautes fréquences décroissent plus vite que les basses fréquences. Si *ktime* reçoit par inadvertance un nombre non positif, il est automatiquement réinitialisé à 0.01. (Nouveau dans la version 4.07 de Csound.)

A partir de la version 4.09 de Csound, *nreverb* peut lire n'importe quel nombre de filtres en peigne et passe-tout depuis une ftable.

## Exemples

Voici un exemple simple de l'opdoce *nreverb*. Il utilise le fichier *nreverb.csd* [examples/nreverb.csd].

### Exemple 619. Exemple simple de l'opdoce *nreverb*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o nreverb.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

gaout init 0

instr 1
a1 oscil 15000, 440, 1
out a1

gaout = gaout+a1
endin

instr 99

a2 nreverb gaout, 2, .3
out a2*.15 ;volume of reverb

gaout = 0
endin

</CsInstruments>
<CsScore>

; Table 1: an ordinary sine wave.
f 1 0 32768 10 1

i 1 0 .5
i 1 1 .5
i 1 2 .5
i 1 3 .5
i 1 4 .5
i 99 0 9
e

</CsScore>
</CsoundSynthesizer>

```

Voici un exemple de l'opcode `nreverb` utilisant une ftable pour les constantes de filtre. Il utilise les fichiers `nreverb_ftable.csd` [examples/nreverb\_ftable.csd] et `beats.wav` [examples/beats.wav].

**Exemple 620. Un exemple de l'opcode `nreverb` utilisant une ftable pour les constantes de filtre.**

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o nreverb_ftable.wav -W ;; for file output any platform

```

```

</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
  a1 soundin "beats.wav"
  a2 nreverb a1, 1.5, .75, 0, 8, 71, 4, 72
  out a1 + a2 * .4
endin

</CsInstruments>
<CsScore>

; freeverb time constants, as direct (negative) sample, with arbitrary gains
f71 0 16  -2  -1116 -1188 -1277 -1356 -1422 -1491 -1557 -1617  0.8  0.79  0.78  0.77  0.76  0.75  0.74

f72 0 16  -2  -556 -441 -341 -225  0.7  0.72  0.74  0.76

i1 0 3
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Paris Smaragdis (*reverb2*)  
 MIT, Cambridge  
 1995

Auteur : Richard Karpen (*nreverb*)  
 Seattle, Wash  
 1998

# nrpn

nrpn — Envoie un numéro de paramètre non référencés sur le port MIDI OUT.

## Description

Envoie un message NRPN (Numéro de Paramètre Non Référencé) sur le port MIDI OUT chaque fois qu'un des arguments d'entrée change.

## Syntaxe

**nrpn** kchan, kparmnum, kparmvalue

## Exécution

*kchan* -- canal MIDI (1-16)

*kparmnum* -- numéro du paramètre NRPN

*kparmvalue* -- valeur du paramètre NRPN

Cet opcode envoie un nouveau message lorsque la valeur MIDI traduite de l'un de ses arguments d'entrée change. Il opère au taux-k. Il est utile avec les instruments MIDI qui reconnaissent les NRPN (par exemple avec les cartes son récentes ayant un synthétiseur MIDI interne telles que SB AWE32, AWE64, GUS, etc, dans lesquelles chaque paramètre de patch peut être modifié durant l'exécution via NRPN).

## Exemples

Voici un exemple de l'opcode nrpn. Il utilise le fichier *nrpn.csd* [examples/nrpn.csd].

### Exemple 621. Exemple de l'opcode nrpn.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

Cet exemple génère des notes chaque fois qu'une note est reçue sur l'entrée MIDI. Comme il génère ces notes MIDI sur la sortie MIDI de Csound, il faut s'assurer d'y connecter quelque chose.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -Q1      ;;;realtime audio out with MIDI out
;-iadc        ;;;uncomment -iadc if realtime audio input is needed too
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
odbfs = 1

instr 1 ; change attack time of external synth

initc7 1, 6, 0 ; set controller 6 to 0
```

```
nrpn 1, 99, 1 ; set MSB
nrpn 1, 98, 99 ; set LSB
katt ctrl17 1, 6, 1, 127 ; DataEntMSB
idur = 2
noteondur2 1, 60, 100, idur ; play note on synth

endin
</CsInstruments>
<CsScore>

i 1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1998

Nouveau dans la version 3.492 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.



# nsamp

nsamp — Retourne le nombre d'échantillons chargés dans une table de fonction.

## Description

Retourne le nombre d'échantillons chargés dans une table de fonction.

## Syntaxe

**nsamp**(x) (arg de taux-i seulement)

## Exécution

Retourne le nombre d'échantillons chargés dans la table de fonction numéro *x* par *GEN01*. Utile lorsqu'un échantillon est plus court que la puissance de deux, taille de la table de fonction qui le contient. Nouveau dans la version 3.49 de Csound.

A partir de la version 5.02 de Csound, *nsamp* travaille avec les tables de fonction à longueur différée (voir *GEN01*).

*nsamp* diffère de *flen* en ce sens que *nsamp* donne le nombre de trames d'échantillon chargées, tandis que *flen* donne le nombre total d'échantillons. Par exemple, avec un fichier son stéréo de 10000 échantillons, *flen()* retournera 19999 (c'est-à-dire un total de 20000 échantillons mono, en excluant le point de garde), mais *nsamp()* retournera 10000.

## Exemples

Voici un exemple de l'opcode *nsamp*. Il utilise les fichiers *nsamp.csd* [examples/nsamp.csd] *kickroll.wav* [examples/kickroll.wav], et *fox.wav* [examples/fox.wav].

### Exemple 622. Exemple de l'opcode *nsamp*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o nsamp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
; slightly adapted example from Jonathan Murphy Dec 2006

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
```

```

ifn  =  p4 ; table number
ilen =  nsamp(ifn)
prints "actual numbers of samples = %d\n", ilen
itrns = 1 ; no transposition
ilps = 0 ; loop starts at index 0
ilpe = ilen ; ends at value returned by nsamp above
imode = 1 ; loops forward
istrt = 0 ; commence playback at index 0 samples
; lphasor provides index into f1
alphs lphasor itrns, ilps, ilpe, imode, istrt
atab tablei alphs, ifn
outs atab, atab

endin
</CsInstruments>
<CsScore>
f 1 0 262144 1 "kickroll.wav" 0 4 1 ;stereo file in table, with lots of zeroes
f 2 0 262144 1 "fox.wav" 0 4 1 ;mono file in table, with lots of zeroes

i1 0 10 1
i1 + 10 2
e

</CsScore>
</CsoundSynthesizer>

```

Comme le fichier audio stéréo « kickroll.wav » a 37792 échantillons, et que le fichier mono « fox.wav » a 121569 échantillons, sa sortie comprendra des lignes comme celles-ci :

```

actual numbers of samples = 37792
actual numbers of samples = 121569

```

## Voir aussi

*ftchnls, flen, flptim, ftsr*

## Crédits

Auteur : Gabriel Maldonado  
 Italie  
 Octobre 1998

Exemple écrit par Kevin Conder.

# nstance

*nstance* — Programme une nouvelle instance d'instrument et stocke le descripteur de l'instance dans une variable.

## Description

Programme une nouvelle instance d'instrument et retourne un descripteur utilisable ultérieurement pour faire référence directement à l'instance courante. Cet opcode est semblable à *schedule* avec cependant la possibilité supplémentaire de retrouver le descripteur de l'instance.

## Syntaxe

```
iHandle nstance insnum, iwhen, idur [, ip4] [, ip5] [...]  
iHandle nstance "insname", iwhen, idur [, ip4] [, ip5] [...]
```

## Initialisation

*iHandle* -- cette variable reçoit un descripteur de l'instance de l'évènement. On peut l'utiliser, par exemple, pour arrêter l'instance de l'instrument donné via l'opcode *turnoff*. Noter que le descripteur de l'instance n'est valide qu'une fois l'instrument initialisé.

*insnum* -- numéro de l'instrument. Equivalent à p1 dans une *instruction i* de partition. *insnum* doit être un nombre supérieur au numéro de l'instrument appelant.

« *insname* » -- Une chaîne de caractères (entre guillemets) représentant un instrument nommé.

*iwhen* -- date de début du nouvel évènement. Equivalent à p2 dans une *instruction i* de partition. *iwhen* ne doit pas être négatif. Si *iwhen* vaut zéro, *insnum* doit être supérieur ou égal au p1 de l'instrument courant.

*idur* -- durée de l'évènement. Equivalent à p3 dans une *instruction ide* partition.

*ip4, ip5, ...* -- Equivalent à p4, p5, etc., dans une *instruction ide* partition.

## Exécution

*nstance* ajoute un nouvel évènement de partition. Les arguments, options incluses, sont les mêmes que dans une partition. Le temps *iwhen* (p2) est mesuré à partir de l'instant de cet évènement.

Si la durée est nulle ou négative, le nouvel évènement est de type MIDI, et il hérite le sous-évènement de relachement (release) de l'instruction de programmation.



### Note

Noter que l'opcode *nstance* ne peut pas accepter de p-champs chaîne de caractères. Si vous devez passer des chaînes de caractères à l'instanciation d'un instrument, utilisez l'opcode *scoreline* ou *scoreline\_i*.

## Exemples

Voici un exemple de l'opcode *nstance*. Il utilise le fichier *nstance.csd* [examples/nstance.csd].

### Exemple 623. Exemple de l'opcode `nstance`.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o nstance.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - oscillator with a high note.
instr 1
; Play Instrument #2 at the same time.
iHandle nstance 2, 0, p3

; Play a high note.
a1 oscils 10000, 880, 1
out a1
endin

; Instrument #2 - oscillator with a low note.
instr 2
; Play a low note.
a1 oscils 10000, 220, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for half a second.
i 1 0 0.5
; Play Instrument #1 for half a second.
i 1 1 0.5
e

</CsScore>
</CsoundSynthesizer>
```

## See Also

*schedwhen*

## Voir aussi

*event, event\_i, schedwhen, schedkwhen, schedkwhennamed, scoreline, scoreline\_i*

Pour plus d'information sur cet opcode : [http://www.csoundjournal.com/issue15/phrase\\_loops.html](http://www.csoundjournal.com/issue15/phrase_loops.html), écrit par Jim Aikin.

## Crédits

Auteur : Victor Lazzarini  
Novembre 2013

# ntof

ntof — Convertit un nom de note en fréquence.

## Description

Opcode du greffon emugens.

Convertit un nom de note en fréquence (Hz). Permet d'inclure dans le nom de la note des microtons ou une déviation en cents.

## Syntaxe

```
kfreq ntof Snote
```

```
ifreq ntof Snote
```

## Exécution

*Snote* -- Nom de note

*kfreq* -- Fréquence



### Note

La fréquence retournée dépend de la valeur de la variable globale A4.



### Note

4C est le do médiant du piano.

## Exemples

Voici un exemple de l'opcode ntof. Il utilise le fichier *ntof.csd* [examples/ntof.csd].

### Exemple 624. Exemple de l'opcode ntof.

```
<CsoundSynthesizer>
<CsOptions>
--nosound
</CsOptions>
<CsInstruments>

/*

Example for ntof: notename to frequency

A notename is a string of the form

<octave><pitchclass>[cents]

Example notenames are:
```

```

"4A"      -> A in the 4th octave
"5C+31"   -> C5, 31 cents higher
"4Db-13"  -> D flat, 4th octave, 13 cents lower
"7F#+-"   -> F# quarter tone higher
"5G-'"    -> G5 a quarter tone lower

NB: the frequency returned by ntof depends on the value of A4

*/

A4 = 442

instr 1
  ifreq ntof "4A"
  print ifreq

  koctave = 1
  while (koctave <= 8) do
    Snote sprintfk "%dC", koctave
    kfreq ntof Snote
    printf "notename: %s  freq: %.2f \n", koctave, Snote, kfreq
    koctave += 1
  od

  turnoff
endin

</CsInstruments>
<CsScore>

i 1 0 1

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*mtof, ftom, mton mton*

## Crédits

Par : Eduardo Moguillansky 2019

Nouveau dans la 6.13

# ntom

ntom — Conversion d'un nom de note en numéro de note MIDI.

## Description

Convertit un nom de note en numéro de note MIDI. Les noms de notes peuvent comprendre des microtons ou une déviation en cents.

## Syntaxe

```
kmidi ntom Snote
```

```
imidi ntom Snote
```

## Exécution

*Snote* -- Nom de note.

*kmidi* -- Numéro de note MIDI.

Exemple de noms de note :

midi	nom de note
-----	
60	4C
60.4	4C+40
60.5	4C+
60.9	4Db-10
61	4C#
61.5	4D-



### Note

4C est le do central du piano.

## Exemples

Voici un exemple de l'opcode ntom. Il utilise le fichier *mton-ntom.csd* [examples/mton-ntom.csd].

### Exemple 625. Exemple de l'opcode ntom.

```
<CsoundSynthesizer>
<CsOptions>
--nosound
</CsOptions>
<CsInstruments>

instr 1
  S4 mton ntom( "7D+63" )
  puts S4, 1
```



```

S1 mton 60
printf_i "midi 60 = %s \n", 1, S1

S2 mton fton(442)
printf_i "442 Hz = %s \n", 1, S2

S3 = mton(48.25)
printf_i "midi 48.25 = %s \n", 1, S3

k1 = nton("4C")
printf_i "4C = midi %f \n", 1, k1

i2 nton "4E"
printf_i "4E = %f \n", 1, i2

S5 = mton(nton("4G+"))
printf_i "roundtrip 4G+: %s \n", 1, S5

turnoff
endin

instr 2
; test i-time and k-time execution
k1 = nton("4Eb-3l")
printf "4Eb-3l = %f \n", 1, k1

i0 nton "4C+"
printf_i "4C+ = %f \n", 1, i0

i1 = nton:i("4A")
printf_i "4A = %f \n", 1, i1
turnoff
endin

</CsInstruments>
<CsScore>

i 1 0 1
i 2 0 1

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*cpsmidinn, mtof, fton, mton*

## Crédits

Par : Eduardo Moguillansky 2017

Nouveau greffon dans la version 6.11

# nstrnum

nstrnum — Retourne le numéro d'un instrument nommé.

## Description

Retourne le numéro d'un instrument nommé.

## Syntaxe

```
insno nstrnum "name"
```

## Initialisation

*insno* -- le numéro de l'instrument nommé.

## Exécution

*"name"* -- le nom de l'instrument nommé.

Si aucun instrument n'existe avec le nom spécifié, une erreur d'initialisation survient, et la valeur -1 est retournée.

## Crédits

Auteur : Istvan Varga  
Nouveau dans la version 4.23  
Ecrit en 2002.

# nstrstr

nstrstr — Retourne le nom d'un instrument nommé à partie de son numéro.

## Description

Retourne le nom d'un instrument nommé à partie de son numéro ou une chaîne de caractères vide si une telle association n'existe pas.

## Syntaxe

Sname **nstrstr**insno

Sname **nstrstr**knsno

## Initialisation

*insno* -- le numéro d'instrument de l'instrument nommé.

## Exécution

*knsno* -- le numéro de l'instrument nommé.

*Sname* -- le nom de l'instrument nommé.

Si aucun instrument existant ne porte le nom spécifié, une chaîne de caractères vide est retournée.

## Crédits

Auteur : John ffitch  
Nouveau dans la version 6.13  
Ecrit en 2019.

# ntrpol

ntrpol — Calcule la valeur de la moyenne pondérée de deux signaux d'entrée.

## Description

Calcule la valeur de la moyenne pondérée (c'est-à-dire l'interpolation linéaire) de deux signaux d'entrée.

## Syntaxe

```
ares ntrpol asig1, asig2, kpoint [, imin] [, imax]
ires ntrpol isig1, isig2, ipoint [, imin] [, imax]
kres ntrpol ksig1, ksig2, kpoint [, imin] [, imax]
```

## Initialisation

*imin* -- valeur minimale pour *xpoint* (facultatif, 0 par défaut)

*imax* -- valeur maximale pour *xpoint* (facultatif, 1 par défaut)

## Exécution

*xsig1*, *xsig2* -- signaux d'entrée

*xpoint* -- point d'interpolation entre les deux valeurs

L'opcode *ntrpol* produit l'interpolation linéaire entre deux valeurs d'entrée. *xpoint* est la distance entre le point d'évaluation et la première valeur. Avec les valeurs par défaut de *imin* et de *imax* (0 and 1), une valeur de zéro indique aucune distance depuis la première valeur et une distance maximale à la seconde valeur. Avec une valeur de 0.5, *ntrpol* produit la valeur moyenne des deux entrées, indiquant exactement le point médian entre *xsig1* et *xsig2*. Une valeur de un indique la distance maximale de la première valeur et pas de distance avec la seconde valeur. La plage de valeurs de *xpoint* peut aussi être définie avec *imin* et *imax* pour rendre sa gestion plus facile.

Ces opcodes sont utiles pour réaliser un fondu-enchaîné de deux signaux.

## Exemples

Voici un exemple de l'opcode *ntrpol*. Il utilise le fichier *ntrpol.csd* [examples/ntrpol.csd].

### Exemple 626. Exemple de l'opcode ntrpol.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if real audio input is needed too
```

```
; For Non-realtime ouput leave only the line below:
; -o ntrpol.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSin ftgen 1, 0, 1024, 10, 1

instr 1

avco vco2 .5, 110 ;sawtooth wave
asin poscil .5, 220, giSin ;sine wave but octave higher
kx linseg 0, p3*.4, 1, p3*.6, 1 ;crossfade between saw and sine
asig ntrpol avco, asin, kx
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Gabriel Maldonado

Italie

Octobre 1998

Nouveau dans la version 3.49 de Csound.

# octave

octave — Calcule un facteur pour élever/abaisser une fréquence d'un certain nombre d'octaves.

## Description

Calcule un facteur pour élever/abaisser une fréquence d'un certain nombre d'octaves.

## Syntaxe

`octave(x)`

Cette fonction travaille aux taux-i, -k et -a.

## Initialisation

*x* -- une valeur exprimée en octaves.

## Exécution

La valeur retournée par la fonction *octave* est un facteur. On peut multiplier une fréquence par ce facteur pour l'élever/l'abaisser du nombre d'octaves spécifié.

## Exemples

Voici un exemple de l'opcode octave. Il utilise le fichier *octave.csd* [examples/octave.csd].

### Exemple 627. Exemple de l'opcode octave.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o octave.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

iroot = 440 ; root note is A above middle-C (440 Hz)
koc   lfo 5, 1, 5 ; generate sawtooth, go from 5 octaves higher to root
koc   int(koc) ; produce only whole numbers
kfactor = octave(koc) ; for octave
knew = iroot * kfactor
```

```

printk2 knew

asig pluck 1, knew, 1000, 0, 1
asig dcblock asig ;remove DC
    outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 5
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra ces lignes :

```

i1  3520.00000
i1  1760.00000
i1   880.00000
i1   440.00000
i1  7040.00000
i1  3520.00000
i1  1760.00000
i1   880.00000
i1   440.00000
.....

```

## Voir aussi

*cent, db, semitone*

## Crédits

Nouveau dans la version 4.16

# octcps

octcps — Convertit des cycles par seconde en valeur octave-point-partie-décimale.

## Description

Convertit des cycles par seconde en valeur octave-point-partie-décimale.

## Syntaxe

**octcps** (cps) (arguments de taux-i ou -k seulement)

où l'argument entre parenthèses peut être une expression.

## Exécution

*octcps* et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

**Tableau 16. Valeurs de Hauteur et de Fréquence**

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par constraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.





## Note

La conversion de *pch*, *oct*, ou *midim* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

## Exemples

Voici un exemple de l'opcode *octcps*. Il utilise le fichier *octcps.csd* [examples/octcps.csd].

### Exemple 628. Exemple de l'opcode *octcps*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o octcps.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Convert a cycles-per-second value into an
; octave value.
icps = 440
ioct = octcps(icps)

print ioct
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  ioct = 8.750
```

## Voir aussi

*cpsoct, cpspch, octpch, pchoct, cpsmidinn, octmidinn, pchmidinn*

## Crédits

Exemple écrit par Kevin Conder.

# octmidi

octmidi — Retourne le numéro de note, en unités octave-point-décimal, de l'évènement MIDI courant.

## Description

Retourne le numéro de note, en unités octave-point-décimal, de l'évènement MIDI courant.

## Syntaxe

ioct octmidi

## Exécution

Retourne le numéro de note de l'évènement MIDI courant, exprimé en unités octave-point-décimal, pour traitement local.



### octmidi vs. octmidinn

L'opcode *octmidi* ne produit des résultats significatifs qu'avec une note activée par le MIDI (soit en , soit depuis une partition MIDI avec l'option -F). Avec *octmidi*, la valeur du numéro de note MIDI provient de l'évènement MIDI qui est associé en interne avec l'instance de l'instrument. Au contraire, l'opcode *octmidinn* peut être utilisé dans n'importe quelle instance d'instrument de Csound, que celle-ci soit activée par un évènement MIDI, un évènement de partition, un évènement en ligne ou depuis un autre instrument. La valeur d'entrée de *octmidinn* peut provenir par exemple d'un p-champ dans une partition textuelle ou bien elle peut avoir été extraite au moyen de l'opcode *notnum* de l'évènement MIDI en qui a activé la note courante.

## Exemples

Voici un exemple de l'opcode octmidi. Il utilise le fichier *octmidi.csd* [examples/octmidi.csd].

### Exemple 629. Exemple de l'opcode octmidi.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac          -iadc      -d           -M0   ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o octmidi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  ; This example expects MIDI note inputs on channel 1
  i1 octmidi

  print i1
endin

</CsInstruments>
<CsScore>

;Dummy f-table to give time for real-time MIDI events
f 0 8000
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*aftouch, ampmidi, cpsmidi, cpsmidib, midictrl, notnum, octmidib, pchbend, pchmidi, pchmidib, veloc, cps-midinn, octmidinn, pchmidinn*

## Crédits

Auteur : Barry L. Vercoe - Mike Berry  
MIT - Mills  
Mai 1997

Exemple écrit par Kevin Conder.

# octmidib

octmidib — Retourne le numéro de note de l'évènement MIDI courant en le modifiant par la valeur courante de pitch-bend, exprimé en unités octave-point-décimal.

## Description

Retourne le numéro de note de l'évènement MIDI courant en le modifiant par la valeur courante de pitch-bend, exprimé en unités octave-point-décimal.

## Syntaxe

```
ioct octmidib [irange]
```

```
koct octmidib [irange]
```

## Initialisation

*irange* (facultatif) -- l'étendue du pitch-bend en demi-tons.

## Exécution

Retourne le numéro de note de l'évènement MIDI courant en le modifiant par la valeur courante de pitch-bend et exprime le résultat en unités octave-point-décimal. Disponible comme une valeur d'initialisation ou comme une valeur continue de taux-k.

## Exemples

Voici un exemple de l'opcode octmidib. Il utilise le fichier *octmidib.csd* [examples/octmidib.csd].

### Exemple 630. Exemple de l'opcode octmidib.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac          -iadc      -d          -M0   ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o octmidib.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; This example expects MIDI note inputs on channel 1
```

```
i1 octmidib

print i1
endin

</CsInstruments>
<CsScore>

;Dummy f-table to give time for real-time MIDI events
f 0 8000
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*aftouch, ampmidi, cpsmidi, cpsmidib, midictrl, notnum, octmidi, pchbend, pchmidi, pchmidib, veloc*

## Crédits

Auteur : Barry L. Vercoe - Mike Berry  
MIT - Mills  
Mai 1997

Exemple écrit par Kevin Conder.

# octmidinn

octmidinn — Convertit un numéro de note Midi en octave-point-partie-décimale.

## Description

Convertit un numéro de note Midi en octave-point-partie-décimale.

## Syntaxe

**octmidinn** (MidiNoteNumber) (arguments de taux-i ou -k seulement)

où l'argument entre parenthèses peut être une expression.

## Exécution

*octmidinn* est une fonction qui prend une valeur de taux-i ou de taux-k représentant un numéro de note Midi et qui retourne la valeur de hauteur équivalente dans le format octave-point-partie-décimale de Csound. Cette conversion suppose que le do médian (8.000 en *oct*) est la note Midi numéro 60. Les numéros de note Midi sont par définition des nombres entiers compris entre 0 et 127 mais des valeurs fractionnaires ou des valeurs en dehors de cet intervalle seront correctement interprétées.



### octmidinn vs. octmidi

L'opcode *octmidinn* peut être utilisé dans n'importe quelle instance d'instrument de Csound, que celle-ci soit activée depuis un évènement Midi, un évènement de partition, un évènement en ligne, ou depuis un autre instrument. La valeur d'entrée de *octmidinn* peut provenir par exemple d'un p-champ dans une partition textuelle ou bien avoir été retrouvée au moyen de l'opcode *notnum* à partir de l'évènement Midi en qui a activé la note courante. Le numéro de note Midi à convertir doit être spécifié comme une expression de taux-i ou de taux-k. D'un autre côté, l'opcode *octmidi* ne fournit des résultats significatifs qu'avec une note activée par le Midi (soit en temps réel soit à partir d'une partition Midi avec l'option -F). Avec *octmidi*, la valeur du numéro de note Midi provient de l'évènement Midi associé à l'instance d'instrument, et aucune source ni aucune expression ne peuvent être spécifiées pour cette valeur.

*octmidinn* et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

**Tableau 17. Valeurs de Hauteur et de Fréquence**

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme

deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par contraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.



## Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

## Exemples

Voici un exemple de l'opcode *octmidinn*. Il utilise le fichier *cpsmidinn.csd* [exemples/cpsmidinn.csd].

### Exemple 631. Exemple de l'opcode *octmidinn*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
; Prints a table showing the equivalents of all Midi
; note numbers from 0-127 in cycles-per-second,
; octave.decimal, and octave.pitchclass units.

<CsOptions>
; Select audio/midi flags here according to platform.
; This example produces no audio, so we render in
; non-realtime and turn off sound to disk:
-n
</CsOptions>
<CsInstruments>

instr 1
; i-time loop to print conversion table
imidiNN = 0
loop1:
  icps = cpsmidinn(imidiNN)
  ioct = octmidinn(imidiNN)
  ipch = pchmidinn(imidiNN)
```



```

    print    imidiNN, icps, ioct, ipch

    imidiNN = imidiNN + 1
    if (imidiNN < 128) igoto loop1
endin

instr 2
; test k-rate converters
kMiddleC = 60
kcps = cpsmidinn(kMiddleC)
koct = octmidinn(kMiddleC)
kpch = pchmidinn(kMiddleC)

printks "%d %f %f %f\n", 1.0, kMiddleC, kcps, koct, kpch
endin

</CsInstruments>
<CsScore>
i1 0 0
i2 0 0.1
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*cpsmidinn, pchmidinn, octmidi, notnum, cpspch, cpsoct, octcps, octpch, pchoct*

## Crédits

Dérivé à partir des convertisseurs de valeur originaux de Barry Vercoe.

Nouveau dans la version 5.07

# octpch

octpch — Convertit une valeur de classe de hauteur en octave-point-partie-décimale.

## Description

Convertit une valeur de classe de hauteur en octave-point-partie-décimale.

## Syntaxe

**octpch** (pch) (arguments de taux-i ou -k seulement)

où l'argument entre parenthèses peut être une expression.

## Exécution

*octpch* et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

**Tableau 18. Valeurs de Hauteur et de Fréquence**

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par constraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.



## Note

La conversion de *pch*, *oct*, ou *midim* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

## Exemples

Voici un exemple de l'opcode *octpch*. Il utilise le fichier *octpch.csd* [examples/octpch.csd].

### Exemple 632. Exemple de l'opcode *octpch*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o octpch.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Convert a pitch-class value into an
; octave-point-decimal value.
ipch = 8.09
ioct = octpch(ipch)

print ioct
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  ioct = 8.750
```

## Voir aussi

*cpsoct, cpspch, octcps, pchoct, cpsmidinn, octmidinn, pchmidinn*

## Crédits

Exemple écrit par Kevin Conder.

# olabuffer

olabuffer — Additionne les trames audio tuilées dans des tableaux de taux-k et retourne un signal audio.

## Description

Opcode du greffon `framebuffer`.

*olabuffer* prend des tableaux unidimensionnels de taux-k qui contiennent des trames audio séquentielles et les additionne sur la base d'un facteur de superposition, produisant un signal audio en sortie. C'est utile pour un traitement audio basé sur des trames comme l'analyse/resynthèse spectrale.

## Syntaxe

```
aout olabuffer kin, ioverlap
```

## Initialisation

*ioverlap* -- Le facteur de superposition par taille d'échantillons de trame dans le tableau de taux-k en entrée. Par exemple pour une taille de fenêtre de 1024 en entrée et une taille de saut de 256, le facteur de superposition vaut 4. Le facteur de superposition doit être supérieur ou égal à *ksmps* et doit aussi être un multiple entier du nombre d'échantillons dans le tableau de taux-k en entrée.

## Exécution

*aout* -- Le signal audio résultant de l'addition des trames en entrée. *kin* -- Un tableau de taux-k contenant les trames audio séquentielles.

## Exemples

Voici un exemple simple de l'opcode `olabuffer`. Il utilise le fichier *framebuffer.csd* [exemples/framebuffer.csd].

### Exemple 633. Exemple de l'opcode `olabuffer`.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>
nchnls = 2
0dbfs = 1
ksmps = 128
sr = 44100

instr 1
  isize init 1024
  ioverlaps init 4

  asig diskin2 "fox.wav", 1, 0, 1
```

```
kframe[] framebuffer asig, isize
kwindowedFrame[] window kframe, isize

aout olabuffer kwindowedFrame, ioverlaps
aout = aout / 2

outs aout, aout
endin

</CsInstruments>
<CsScore>
i 1 0 400
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*olabuffer shiftin shiftout*

## Crédits

Auteur : Edward Costello;  
NUIM, 2015

Nouveau dans la version 6.06

# opcode

opcode — Commence un bloc d'opcode défini par l'utilisateur.

## Définir des opcodes

Les instructions *opcode* et *endop* permettent de définir un nouvel opcode qui peut être utilisé de la même façon qu'un opcode original de Csound. Ces blocs d'opcode ressemblent beaucoup aux instruments (et sont, en fait, implémentés comme des instruments spéciaux), mais on ne peut pas les appeler comme des instruments normaux, par exemple avec des *instructions i*.

Un bloc d'opcode défini par l'utilisateur doit précéder l'instrument (ou l'opcode) depuis lequel on l'utilise. Mais un opcode peut aussi s'appeler lui-même. Cela permet une récursivité dont la profondeur n'est limitée que par la mémoire disponible. De plus, on peut, à titre expérimental, exécuter l'opcode défini à un taux de contrôle plus élevé que la valeur de *kr* spécifiée dans l'en-tête de l'orchestre.

Comme pour les instruments, les variables et les étiquettes d'un bloc d'opcode défini par l'utilisateur sont locales et ne sont pas visible depuis l'instrument appelant (de même que l'opcode n'a pas accès aux variables de l'instrument qui l'a appelé).

Cependant, certains paramètres sont copiés automatiquement à l'initialisation :

- tous les p-champs jusqu'à celui de numéro le plus élevé inclus, référencés dans l'instrument appelant
- le temps supplémentaire (voir aussi *xtratim*, *linsegr*, et les opcodes correspondants). Ceci peut affecter le fonctionnement de *linsegr/expsegr/linenr/envlpxr* dans le bloc d'opcode défini par l'utilisateur.
- les paramètres MIDI, s'il y en a.

Le drapeau de release (voir l'opcode *release*) est également copié durant l'exécution.

La modification de la durée de la note dans la définition de l'opcode en assignant une valeur à *p3*, ou l'utilisation de *ihold*, *turnoff*, *xtratim*, *linsegr*, ou d'autres opcodes similaires affecteront aussi l'instrument appelant. Les changements sur des contrôleurs MIDI (par exemple avec *ctrlinit*) s'appliqueront aussi à l'instrument qui a appelé l'opcode.

Utilisez l'opcode *setksmps* pour fixer la valeur locale de *ksmps*.

Les opcodes *xin* et *xout* copient les variables vers et depuis la définition de l'opcode, permettant la communication avec l'instrument appelant.

Les types des variables d'entrée et de sortie sont définis par les paramètres *intypes* et *outtypes*.



### Astuce

On peut créer des UDOs sans entrée ou sans sortie en remplaçant la chaîne de caractères correspondante par 0.



### Notes

- *xin* et *xout* ne doivent être appelés qu'une seule fois, et *xin* doit précéder *xout*, sinon une erreur d'initialisation et une désactivation de l'instrument courant peuvent se produire.
- Ces deux opcodes n'agissent qu'à l'initialisation. La copie durant l'exécution est réalisée par l'appel de l'opcode de l'utilisateur. Cela signifie que sauter *xin* ou *xout* avec *goto* n'a

aucun effet, alors que les sauter avec *igoto* affecte à la fois les opérations de l'initialisation et de l'exécution.

## Syntaxe

`opcode nom, outtypes, intypes`

## Initialisation

*nom* -- nom de l'opcode. Il est constitué de n'importe quelle combinaison de lettres, chiffres et traits de soulignement mais il ne doit pas commencer par un chiffre. Si un opcode du même nom existe déjà, il est redéfini (un avertissement est affiché dans ce cas). Certains mots réservés (comme *instr* et *endin*) ne peuvent pas être redéfinis.

*intypes* -- liste des types en entrée, toute combinaison des caractères a, k, O, P, V, K, i, o, p et j. Un caractère 0 unique peut être utilisé s'il n'y a pas d'argument en entrée. Il n'y a *pas* besoin d'apostrophes doubles et de délimiteurs (comme la virgule).

La signification des différent *intypes* est montrée dans le tableau suivant :

Type	Description	Types de Variable Autorisés	Mise à jour
a	variable de taux-a	taux-a	taux-a
i	variable de taux-i	taux-i	initialisation
j	facultatif de taux-i, -1 par défaut	taux-i, constante	initialisation
k	variable de taux-k	taux-k et -i, constante	taux-k
O	variable facultative de taux-k, valant 0 par défaut	taux-k et -i, constante	taux-k
P	variable facultative de taux-k, valant 1 par défaut	taux-k et -i, constante	taux-k
V	variable facultative de taux-k, valant 0.5 par défaut	taux-k et -i, constante	taux-k
K	taux-k avec initialisation	taux-k et -i, constante	taux-i et taux-k
o	facultatif à l'initialisation, 0 par défaut	taux-i, constante	initialisation
p	facultatif à l'initialisation, 1 par défaut	taux-i, constante	initialisation
S	variable chaîne de caractères	chaîne de caractères de taux-i	initialisation

Le nombre maximum d'arguments en entrée autorisé est 256.

*outtypes* -- liste des types en sortie. Le format est le même que celui utilisé pour *intypes*.

Voici les *outtypes* disponibles :



Type	Description	Types de Variable Autorisés	Mise à jour
a	variable de taux-a	taux-a	taux-a
i	variable de taux-i	taux-i	initialisation
k	variable de taux-k	taux-k	taux-k
K	taux-k avec initialisation	taux-k	taux-i et taux-k

Le nombre maximum d'arguments en sortie autorisé est 256.

*iksmips* (facultatif, 0 par défaut) -- fixe la valeur locale de *ksmps*. Doit être un nombre entier positif, et le *ksmps* de l'instrument appelant doit être un multiple entier de cette valeur. Par exemple, si *ksmps* vaut 10 dans l'instrument depuis lequel l'opcode a été appelé, les valeurs permises pour *iksmips* sont 1, 2, 5, et 10.

Si *iksmips* vaut zéro, le *ksmps* de l'instrument ou de l'opcode appelant est utilisé (c'est le comportement par défaut).



## Note

Le *ksmps* local est implémenté en divisant une période de contrôle en sous-périodes-k plus petites et en modifiant temporairement les variables globales internes de Csound. Ceci nécessite aussi la conversion du taux des arguments d'entrée et de sortie de taux-k (les variables d'entrée reçoivent la même valeur dans tous les sous-périodes-k, tandis que les valeurs de sortie ne sont écrites que pendant la dernière).



## Avertissement au sujet du *ksmps* local

Lorsque le *ksmps* local est différent du *ksmps* de l'orchestre (celui spécifié dans l'en-tête de l'orchestre), il ne faut pas utiliser d'opération globale de taux-a dans le bloc d'opcode défini par l'utilisateur.

Ceci comprend :

- tous les accès aux variables « ga »
- les opcodes zak de taux-a (*zar*, *zaw*, etc.)
- *tablera* et *tablewa* (ces deux opcodes peuvent fonctionner en fait, mais il faut prendre des précautions)
- La famille d'opcode *in* et *out* (ils lisent depuis et écrivent dans des tampons globaux de taux-a)

En général, il faut utiliser le *ksmps* local avec précaution car c'est une fonctionnalité expérimentale, bien qu'elle fonctionne correctement dans la plupart des cas.

L'instruction *setksmps* peut être utilisée pour fixer la valeur du *ksmps* local du bloc d'opcode défini par l'utilisateur. Elle a un paramètre de taux-i spécifiant la nouvelle valeur de *ksmps* (qui reste inchangée si l'on utilise zéro, voir aussi les notes au sujet de *iksmips* ci-dessus). *setksmps* doit être utilisé avant tout autre opcode (mais il est autorisé après *xin*), autrement des résultats imprévisibles peuvent se produire.

On peut lire les paramètres d'entrée avec l'opcode *xin*, et la sortie est écrite par l'opcode *xout*. On ne doit utiliser qu'une seule instance de ces unités, car *xout* écrase la sortie sans accumuler les valeurs. Le nombre et le type des arguments pour *xin* et *xout* doit être le même que dans la déclaration du bloc d'opcode défini par l'utilisateur (voir les tableaux ci-dessus).

Les arguments d'entrée et de sortie doivent se conformer à la définition à la fois en nombre (sauf si des entrées de taux-i facultatives sont utilisées) et en genre. Un paramètre d'entrée facultatif de taux-i (*ksmps*) est automatiquement ajouté à la liste des *intypes* et (comme pour *setksmps*) fixe la valeur du *ksmps* local.

## Exécution

La syntaxe d'un bloc d'opcode défini par l'utilisateur est la suivante :

```
opcode nom, outtypes, intypes
xinarg1 [, xinarg2] [, xinarg3] ... [xinargN] xin
[setksmps iksmps]
... the rest of the instrument's code.
xout xoutarg1 [, xoutarg2] [, xoutarg3] ... [xoutargN]
endop
```

Le nouvel opcode peut ensuite être utilisé avec la syntaxe usuelle :

```
[xoutarg1] [, xoutarg2] ... [xoutargN] nom [xinarg1] [, xinarg2] ... [xinargN] [, iksmps]
```



### Note

L'opcode est toujours appelé à la fois durant l'initialisation et durant l'exécution, même s'il n'y a pas d'arguments de taux-k ou -a. Si l'on sait que plusieurs opcodes définis par l'utilisateur n'ont pas d'effet durant l'exécution (taux-k) dans un instrument, on peut épargner du temps CPU en sautant ces groupes d'opcodes avec *kgoto*.

## Exemples

Voici un exemple d'opcode défini par l'utilisateur. Il utilise le fichier *opcode.csd* [examples/opcode\_example.csd].

### Exemple 634. Exemple d'opcode défini par l'utilisateur.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o opcode_example.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

/* example opcode 1: simple oscillator */

opcode Oscillator, a, kk

kamp, kcps      xin          ; read input parameters
a1             vco2 kamp, kcps ; sawtooth oscillator
```

```

        xout a1                ; write output

    endop

/* example opcode 2: lowpass filter with local ksmps */

    opcode Lowpass, a, akk

        setksmps 1            ; need sr=kr
    ain, ka1, ka2    xin        ; read input parameters
    aout    init 0            ; initialize output
    aout    = ain*ka1 + aout*ka2 ; simple tone-like filter
    xout aout                ; write output

    endop

/* example opcode 3: recursive call */

    opcode RecursiveLowpass, a, akkpp

    ain, ka1, ka2, idep, icnt    xin        ; read input parameters
    if (icnt >= idep) goto skip1 ; check if max depth reached
    ain    RecursiveLowpass ain, ka1, ka2, idep, icnt + 1
skip1:
    aout    Lowpass ain, ka1, ka2        ; call filter
    xout aout                ; write output

    endop

/* example opcode 4: de-click envelope */

    opcode DeClick, a, a

    ain    xin
    aenv    linseg 0, 0.02, 1, p3 - 0.05, 1, 0.02, 0, 0.01, 0
    xout ain * aenv            ; apply envelope and write output

    endop

/* instr 1 uses the example opcodes */

    instr 1

    kamp    = .7                ; amplitude
    kcps    expon 50, p3, 500    ; pitch
    a1      Oscillator kamp, kcps        ; call oscillator
    kflt    linseg 0.4, 1.5, 0.4, 1, 0.8, 1.5, 0.8 ; filter envelope
    a1      RecursiveLowpass a1, kflt, 1 - kflt, 10 ; 10th order lowpass
    a1      DeClick a1
    outs a1, a1

    endin

</CsInstruments>
<CsScore>

i 1 0 4
e5 ;extra second before quitting

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*endop, setksmps, xin, xout*

## Crédits

Auteur : Istvan Varga, 2002 ; basé sur du code de Matt J. Ingalls

Nouveau dans la version 4.22

# oscbnk

oscbnk — Mélange la sortie de n'importe quel nombre d'oscillateurs.

## Description

Ce générateur unitaire mélange la sortie de n'importe quel nombre d'oscillateurs. La fréquence, la phase et l'amplitude de chaque oscillateur peuvent être modulées par deux LFO (tous les oscillateurs ont un jeu de LFO séparé, avec différentes phase et fréquence) ; de plus, la sortie de chaque oscillateur peut être filtrée au travers d'un égaliseur paramétrique (aussi contrôlé par les LFO). Cet opcode trouve sa plus grande utilité dans des instruments de rendu d'ensemble (cordes, chœur, etc.).

Bien que les LFO fonctionnent au taux-k, les modulations d'amplitude, de phase et de filtrage sont interpolées en interne, et il est ainsi possible (et recommandé dans la plupart des cas) d'utiliser cette unité avec de faibles taux de contrôle (~1000 Hz) sans dégradation audible de la qualité.

La phase et la fréquence initiale de tous les oscillateurs et LFO peuvent être fixées par un générateur intégré de nombres aléatoires sur 31 bit amorçable par une « graine », ou spécifiées manuellement dans une table de fonction (GEN2).

## Syntaxe

```
ares oscbnk  kcps, kamd, kfmd, kpmd, iovrlap, iseed, kllminf, kllmaxf, \
               kl2minf, kl2maxf, ilfomode, keqminf, keqmaxf, keqminl, keqmaxl, \
               keqminq, keqmaxq, iegmode, kfn [, illfn] [, il2fn] [, iegffn] \
               [, iegqlfn] [, iegqfn] [, itabl] [, ioutfn]
```

## Initialisation

*iovrlap* -- Nombre d'oscillateurs.

*iseed* -- Valeur de la graine du générateur de nombres aléatoires (entier positif dans l'intervalle 1 à 2147483646 ( $2^{31} - 2$ )). Si *iseed*  $\leq 0$  la graine est l'heure courante.

*iegmode* -- Mode de l'égaliseur paramétrique

- -1 : désactive l'EQ (plus rapide)
- 0 : crête
- 1 : à plateau low shelf
- 2 : à plateau high shelf
- 3 : crête (filtrage sans interpolation)
- 4 : à plateau low shelf (sans interpolation)
- 5 : à plateau high shelf (sans interpolation)

Les modes sans interpolation sont plus rapides, et dans certains cas (par exemple filtre à plateau high shelf aux fréquences de coupure basses) également plus stables ; cependant, l'interpolation est utile pour éviter le « bruit de transition » aux faibles taux de contrôle.

*ilfomode* -- Type de la modulation par les LFO, somme de :

- 128 : LFO1 module la fréquence

- 64 : LFO1 module l'amplitude
- 32 : LFO1 module la phase
- 16 : LFO1 module l'EQ
- 8 : LFO2 module la fréquence
- 4 : LFO2 module l'amplitude
- 2 : LFO2 module la phase
- 1 : LFO2 module l'EQ

Si un LFO ne module rien, il n'est pas calculé, et le numéro de sa ftable (*il1fn* ou *il2fn*) peut être omis.

*il1fn* (facultatif : par défaut 0) -- Numéro de la table de fonction de LFO1. La forme d'onde dans cette table doit être normalisée (valeur absolue  $\leq 1$ ), et elle est lue avec une interpolation linéaire.

*il2fn* (facultatif : par défaut 0) -- Numéro de la table de fonction de LFO2. La forme d'onde dans cette table doit être normalisée (valeur absolue  $\leq 1$ ), et elle est lue avec une interpolation linéaire.

*ieqffn*, *ieqlfn*, *ieqqfn* (facultatif : par défaut 0) -- Tables de lecture pour la fréquence, le niveau et le Q de EQ (facultatif si EQ est désactivé). La position de lecture dans une table est 0 si le signal de modulation est inférieur ou égal à -1, (longueur de table / 2) si le signal de modulation vaut zero, et le point de garde si le signal de modulation est supérieur ou égal à 1. Ces tables doivent être normalisées dans l'intervalle 0 - 1, et ont un point de garde étendu (longueur de table = puissance de deux + 1). Toutes les tables sont lues avec une interpolation linéaire.

*itabl* (facultatif : par défaut 0) -- Table de fonction stockant les valeurs de phase et de fréquence pour tous les oscillateurs (facultatif). Les valeurs dans cette table sont dans l'ordre suivant (5 pour chaque oscillateur) :

phase de l'oscillateur, phase de lfo1, fréquence de lfo1, phase de lfo2, fréquence de lfo2, ...

Toutes les valeurs sont dans l'intervalle 0 à 1 ; si le nombre spécifié est supérieur à 1, il est ramené cycliquement (phase) ou limité (fréquence) à l'intérieur de l'intervalle permis. Une valeur négative (ou la fin de la table) utilisera la sortie du générateur de nombres aléatoires. La valeur aléatoire est toujours calculée (même si aucun nombre aléatoire n'est utilisé), si bien que le fait de basculer entre une valeur aléatoire et une valeur fixe n'altérera pas les autres valeurs.

*ioutfn* (facultatif : par défaut 0) -- Table de fonction pour écrire les valeurs de phase et de fréquence (facultatif). Le format est le même que celui de *itabl*. Cette table est utile lors de l'expérimentation avec des nombres aléatoires pour enregistrer les meilleures valeurs.

L'accès aux deux tables facultatives (*itabl* et *ioutfn*) n'a lieu que pendant l'initialisation. Il est utile de savoir cela, car les tables peuvent être réécrites en toute sécurité après l'initialisation de l'opcode, permettant le pré-calcul des paramètres pendant le temps-i et le stockage dans une table temporaire avant l'initialisation de *oscbnk*.

## Exécution

*ares* -- Signal de sortie.

*kcps* -- Fréquence de l'oscillateur en Hz.

*kamd* -- Profondeur de la modulation d'amplitude (0 - 1).

(sortie MA) = (entrée MA) \* ((1 - (prof MA)) + (prof MA) \* (modulateur))

Si *ilfomode* n'est pas réglé pour moduler l'amplitude, alors (sortie MA) = (entrée MA) quelque soit la valeur de *kamd*. Dans ce cas, *kamd* n'aura pas d'effet.

Note : La modulation d'amplitude est appliquée avant l'égaliseur paramétrique.

*kfmd* -- Profondeur de la MF (en Hz).

*kpm* -- Profondeur de la modulation de phase.

*kl1minf*, *kl1maxf* -- Fréquence minimale et maximale de LFO1 en Hz.

*kl2minf*, *kl2maxf* -- Fréquence minimale et maximale de LFO2 en Hz. (Note : il est permis d'avoir des fréquences nulles ou négatives pour l'oscillateur et les LFO.)

*keqminf*, *keqmaxf* -- Fréquence minimale et maximale de l'égaliseur paramétrique en Hz.

*keqminl*, *keqmaxl* -- Niveau minimum et maximum de l'égaliseur paramétrique.

*keqminq*, *keqmaxq* -- Q minimum et maximum de l'égaliseur paramétrique.

*kfn* -- Table de la forme d'onde de l'oscillateur. Le numéro de la table peut être changé au taux-k (c'est utile pour choisir parmi un ensemble de tables à bande limitée générées par GEN30, afin d'éviter les erreurs de repliement). La table est lue avec une interpolation linéaire.



## Note

*oscblk* utilise le même générateur de nombres aléatoires que *rnd31*. C'est pourquoi il est également recommandé de lire *sa documentation*.

## Exemples

Voici un exemple de l'opcode *oscblk*. Il utilise le fichier *oscblk.csd* [examples/oscblk.csd].

### Exemple 635. Exemple de l'opcode *oscblk*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o oscblk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Istvan Varga */
sr = 48000
kr = 750
ksmps = 64
nchnls = 2

ga01 init 0
ga02 init 0

/* sawtooth wave */
i_ ftgen 1, 0, 16384, 7, 1, 16384, -1
/* FM waveform */
i_ ftgen 3, 0, 4096, 7, 0, 512, 0.25, 512, 1, 512, 0.25, 512, \
```

```

0, 512, -0.25, 512, -1, 512, -0.25, 512, 0
/* AM waveform */
i_ ftgen 4, 0, 4096, 5, 1, 4096, 0.01
/* FM to EQ */
i_ ftgen 5, 0, 1024, 5, 1, 512, 32, 512, 1
/* sine wave */
i_ ftgen 6, 0, 1024, 10, 1
/* room parameters */
i_ ftgen 7, 0, 64, -2, 4, 50, -1, -1, -1, 11, \
    1, 26.833, 0.05, 0.85, 10000, 0.8, 0.5, 2, \
    1, 1.753, 0.05, 0.85, 5000, 0.8, 0.5, 2, \
    1, 39.451, 0.05, 0.85, 7000, 0.8, 0.5, 2, \
    1, 33.503, 0.05, 0.85, 7000, 0.8, 0.5, 2, \
    1, 36.151, 0.05, 0.85, 7000, 0.8, 0.5, 2, \
    1, 29.633, 0.05, 0.85, 7000, 0.8, 0.5, 2

/* generate bandlimited sawtooth waves */

i0 = 0
loop1:
imaxh = sr / (2 * 440.0 * exp(log(2.0) * (i0 - 69) / 12))
i_ ftgen i0 + 256, 0, 4096, -30, 1, 1, imaxh
i0 = i0 + 1
if (i0 < 127.5) igoto loop1

instr 1

p3 = p3 + 0.4

; note frequency
kcps = 440.0 * exp(log(2.0) * (p4 - 69) / 12)
; lowpass max. frequency
klpmaxf limit 64 * kcps, 1000.0, 12000.0
; FM depth in Hz
kfmd1 = 0.02 * kcps
; AM frequency
kamfr = kcps * 0.02
kamfr2 = kcps * 0.1
; table number
kfnum = (256 + 69 + 0.5 + 12 * log(kcps / 440.0) / log(2.0))
; amp. envelope
aenv linseg 0, 0.1, 1.0, p3 - 0.5, 1.0, 0.1, 0.5, 0.2, 0, 1.0, 0

/* oscillator / left */

a1 oscbnk kcps, 0.0, kfmd1, 0.0, 40, 200, 0.1, 0.2, 0, 0, 144, \
    0.0, klpmaxf, 0.0, 0.0, 1.5, 1.5, 2, \
    kfnum, 3, 0, 5, 5, 5
a2 oscbnk kcps, 1.0, kfmd1, 0.0, 40, 201, 0.1, 0.2, kamfr, kamfr2, 148, \
    0, 0, 0, 0, 0, 0, -1, \
    kfnum, 3, 4
a2 pareq a2, kcps * 8, 0.0, 0.7071, 2
a0 = a1 + a2 * 0.12
/* delay */
adel = 0.001
a01 vdelayx a0, adel, 0.01, 16
a_ oscili 1.0, 0.25, 6, 0.0
adel = adel + 1.0 / (exp(log(2.0) * a_) * 8000)
a02 vdelayx a0, adel, 0.01, 16
a0 = a01 + a02

ga01 = ga01 + a0 * aenv * 2500

/* oscillator / right */

; lowpass max. frequency

```



```

a1 oscbnk kcps, 0.0, kfmd1, 0.0, 40, 202, 0.1, 0.2, 0, 0, 144, \
    0.0, klpmaxf, 0.0, 0.0, 1.0, 1.0, 2, \
    kfnum, 3, 0, 5, 5, 5
a2 oscbnk kcps, 1.0, kfmd1, 0.0, 40, 203, 0.1, 0.2, kamfr, kamfr2, 148, \
    0, 0, 0, 0, 0, 0, -1, \
    kfnum, 3, 4
a2 pareq a2, kcps * 8, 0.0, 0.7071, 2
a0 = a1 + a2 * 0.12
/* delay */
adel = 0.001
a01 vdelayx a0, adel, 0.01, 16
a_ oscili 1.0, 0.25, 6, 0.25
adel = adel + 1.0 / (exp(log(2.0) * a_) * 8000)
a02 vdelayx a0, adel, 0.01, 16
a0 = a01 + a02

ga02 = ga02 + a0 * aenv * 2500

    endin

/* output / left */

    instr 81

    i1 = 0.000001
    aLl, aLh, aRl, aRh spat3di ga01 + i1*i1*i1*i1, -8.0, 4.0, 0.0, 0.3, 7, 4
    ga01 = 0
    aLl butterlp aLl, 800.0
    aRl butterlp aRl, 800.0

    outs aLl + aLh, aRl + aRh

    endin

/* output / right */

    instr 82

    i1 = 0.000001
    aLl, aLh, aRl, aRh spat3di ga02 + i1*i1*i1*i1, 8.0, 4.0, 0.0, 0.3, 7, 4
    ga02 = 0
    aLl butterlp aLl, 800.0
    aRl butterlp aRl, 800.0

    outs aLl + aLh, aRl + aRh

    endin

</CsInstruments>
<CsScore>

/* Written by Istvan Varga */
t 0 60

i 1 0 4 41
i 1 0 4 60
i 1 0 4 65
i 1 0 4 69

i 81 0 5.5
i 82 0 5.5
e

</CsScore>

```

</CsoundSynthesizer>

## Crédits

Auteur : Istvan Varga  
2001

Nouveau dans la version 4.15

Mis à jour en avril 2002 par Istvan Varga

# oscil1

oscil1 — Accède aux valeurs d'une table par échantillonnage incrémentiel.

## Description

Accède aux valeurs d'une table par échantillonnage incrémentiel.

## Syntaxe

```
kres oscil1 idel, kamp, idur [, ifn]
```

## Initialisation

*idel* -- délai en secondes avant que l'échantillonnage incrémentiel d'*oscil1* ne commence.

*idur* -- durée en secondes de l'unique passe d'échantillonnage dans la table d'*oscil1*. Une valeur négative provoque une lecture de la table de la fin vers le début.

*ifn* (facultatif) -- numéro de la table de fonction. *tablei*, *oscilli* nécessitent un point de garde. Vaut -1 par défaut ce qui indique une onde sinus.

## Exécution

*kamp* -- facteur d'amplitude.

*oscil1* accède aux valeurs en échantillonnant une fois la table de fonction à un taux déterminé par *idur*. Pendant les premières *idel* secondes, le point de lecture reste sur la première position de la table ; ensuite il traverse la table à vitesse constante, atteignant la fin au bout de *idur* secondes ; à partir de ce moment (c-à-d après *idel* + *idur* secondes) il reste sur la dernière position. Chaque valeur lue par échantillonnage est multipliée par le facteur d'amplitude *kamp* avant d'être écrite dans le résultat.

## Exemples

Voici un exemple de l'opcode *oscil1*. Il utilise le fichier *oscil1.csd* [examples/oscil1.csd].

### Exemple 636. Exemple de l'opcode *oscil1*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o oscilli.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
```

```
ksmps = 32
nchnls = 2
0dbfs = 1

instr      1

ipanfn = p4
asig      vco2 .3, 220
kpan      oscilli 0, 1, p3, ipanfn ;create panning &
kleft     = sqrt(kpan) ;start right away
kright    = sqrt(1-kpan)
outs      kleft*asig, kright*asig

endin
</CsInstruments>
<CsScore>
f 1 0 3 -7 .5 3 .5 ;remain in center (.5 CONSTANT)
f 2 0 129 7 1 129 0 ;left-->right
f 3 0 129 7 .5 32 1 64 0 33 .5 ;center-->left-->right-->center

i 1 0 2 1 ;use table 1
i 1 3 2 2 ;use table 2
i 1 6 2 3 ;use table 3

e
</CsScore>
</CsoundSynthesizer>
```

L'exemple produit la sortie suivante :

```
i1      0.50000
i1      0.20000
i1      0.80000
i1      0.10000
i1      0.90000
i1      0.00000
i1      1.00000
i1      0.50000
```

## Voir aussi

*table, tablei, table3, oscilli, osciln*

# oscil1i

oscil1i — Accède aux valeurs d'une table par échantillonnage incrémentiel avec interpolation linéaire.

## Description

Accède aux valeurs d'une table par échantillonnage incrémentiel avec interpolation linéaire.

## Syntaxe

```
kres oscil1i idel, kamp, idur [, ifn]
```

## Initialisation

*idel* -- délai en secondes avant que l'échantillonnage incrémentiel d'*oscil1i* ne commence.

*idur* -- durée en secondes de l'unique passe d'échantillonnage dans la table d'*oscil1i*. Une valeur négative provoque une lecture de la table de la fin vers le début.

*ifn* (facultatif) -- numéro de la table de fonction. *oscil1i* nécessite un point de garde. Vaut -1 par défaut ce qui indique une onde sinus.

## Exécution

*kamp* -- facteur d'amplitude

*oscil1i* est une unité avec interpolation dans laquelle la partie fractionnaire de l'index est utilisée pour interpoler entre les entrées adjacentes de la table. La régularité apportée par l'interpolation se paie par une légère augmentation du temps d'exécution (voir aussi *oscili*, etc.), mais sinon les unités avec ou sans interpolation sont interchangeables.

## Exemples

Voici un exemple de l'opcode *oscil1i*. Il utilise le fichier *oscil1i.csd* [examples/oscil1i.csd].

### Exemple 637. Exemple de l'opcode *oscil1i*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o oscil1i.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
```

```

0dbfs = 1

instr      1

  ipanfn = p4
  asig   vco2 .3, 220
  kpan   oscilli 0, 1, p3, ipanfn ;create panning &
  kleft  = sqrt(kpan) ;start right away
  kright = sqrt(1-kpan)
  outs kleft*asig, kright*asig

endin
</CsInstruments>
<CsScore>
f 1 0 3 -7 .5 3 .5 ;remain in center (.5 CONSTANT)
f 2 0 129 7 1 129 0 ;left-->right
f 3 0 129 7 .5 32 1 64 0 33 .5 ;center-->left-->right-->center

i 1 0 2 1 ;use table 1
i 1 3 2 2 ;use table 2
i 1 6 2 3 ;use table 3

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*table, tablei, table3, oscil1, osciln*

# oscil3

oscil3 — Un oscillateur simple avec interpolation cubique.

## Description

*oscil3* lit la table *ifn* séquentiellement et de manière répétitive à la fréquence *xcps*. L'amplitude est pondérée par *xamp*. La lecture des valeurs de phase internes de la table se fait avec interpolation cubique.

## Syntaxe

```
ares oscil3 xamp, xcps [, ifn, iphs]
kres oscil3 kamp, kcps [, ifn, iphs]
```

## Initialisation

*ifn* (facultatif) -- numéro de la table de fonction. Nécessite un point de garde pour la lecture cyclique. Vaut -1 par défaut ce qui indique une onde sinus.

*iphs* (facultatif) -- phase initiale de la lecture, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative l'initialisation de la phase sera ignorée. La valeur par défaut est 0.

## Exécution

*kamp*, *xamp* -- amplitude

*kcps*, *xcps* -- fréquence en cycles par seconde.

*oscil3* est identique à *oscili*, sauf qu'il utilise l'interpolation cubique.

La table *ifn* est parcourue par incrément modulo la longueur de la table et la valeur obtenue est multipliée par *amp*.

Si vous désirez changer la table de l'oscillateur avec un signal de taux-k, vous pouvez utiliser *oscilikt*.

## Exemples

Voici un exemple de l'opcode *oscil3*. Il utilise le fichier *oscil3.csd* [examples/oscil3.csd].

### Exemple 638. Exemple de l'opcode *oscil3*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o oscil3.wav -W ;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

  sr = 44100
  ksmps = 32
  nchnls = 2
  odbfs = 1

  instr 1

  kamp = .6
  kcps = 440
  ifn = p4

  asig oscil kamp, kcps, ifn
      outs asig,asig

  endin

  instr 2

  kamp = .6
  kcps = 440
  ifn = p4

  asig oscil3 kamp, kcps, ifn
      outs asig,asig

  endin
</CsInstruments>
<CsScore>
f1 0 128 10 1                                     ; Sine with a small amount of data
f2 0 128 10 1 0.5 0.3 0.25 0.2 0.167 0.14 0.125 .111 ; Sawtooth with a small amount of data
f3 0 128 10 1 0 0.3 0 0.2 0 0.14 0 .111           ; Square with a small amount of data
f4 0 128 10 1 1 1 1 0.7 0.5 0.3 0.1               ; Pulse with a small amount of data

i 1 0 2 1
i 2 3 2 1
i 1 6 2 2
i 2 9 2 2
i 1 12 2 3
i 2 15 2 3
i 1 18 2 4
i 2 21 2 4

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*oscil, oscili, oscilikt.*

## Crédits

Auteur : John ffitich

Nouveau dans la version 3.50 de Csound



# oscil

oscil — Un oscillateur simple.

## Description

*oscil* lit la table *ifn* séquentiellement et de manière répétitive à la fréquence *xcps*. L'amplitude est pondérée par *xamp*.

## Syntaxe

```
ares oscil xamp, xcps [, ifn, iphs]
```

```
kres oscil kamp, kcps [, ifn, iphs]
```

## Initialisation

*ifn* (facultatif) -- numéro de la table de fonction. Nécessite un point de garde pour la lecture cyclique. Vaut -1 par défaut ce qui indique une onde sinus.

*iphs* (facultatif, par défaut 0) -- phase initiale de la lecture, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative l'initialisation de la phase sera ignorée. La valeur par défaut est 0.

## Exécution

*kamp*, *xamp* -- amplitude

*kcps*, *xcps* -- fréquence en cycles par seconde.

L'opcode *oscil* génère des signaux de contrôle (ou audio) constitués de la valeur de *kamp* (*xamp*) fois la valeur de la lecture au taux de contrôle (ou au taux audio) d'une table de fonction stockée. La phase interne est simultanément incrémentée selon la valeur en entrée de *kcps* ou de *xcps*.

La table *ifn* est parcourue par incrément modulo la longueur de la table et la valeur obtenue est multipliée par *amp*.

Si vous désirez changer la table de l'oscillateur avec un signal de taux-k, vous pouvez utiliser *oscilkt*.

## Exemples

Voici un exemple de l'opcode *oscil*. Il utilise le fichier *oscil.csd* [examples/oscil.csd].

### Exemple 639. Exemple de l'opcode *oscil*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
```

```

; For Non-realtime ouput leave only the line below:
; -o oscil.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kamp = .6
kcps = 440
ifn = p4

asig oscil kamp, kcps, ifn
outs asig,asig

endin
</CsInstruments>
<CsScore>
f1 0 16384 10 1 ; Sine
f2 0 16384 10 1 0.5 0.3 0.25 0.2 0.167 0.14 0.125 .111 ; Sawtooth
f3 0 16384 10 1 0 0.3 0 0.2 0 0.14 0 .111 ; Square
f4 0 16384 10 1 1 1 1 0.7 0.5 0.3 0.1 ; Pulse

i 1 0 2 1
i 1 3 2 2
i 1 6 2 3
i 1 9 2 4

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*oscili, oscilikt, oscil3, poscil, poscil3*

# oscili

oscili — Un oscillateur simple avec interpolation linéaire.

## Description

*oscili* lit la table *ifn* séquentiellement et de manière répétitive à la fréquence *xcps*. L'amplitude est pondérée par *xamp*. La lecture des valeurs de phase internes de la table se fait avec interpolation linéaire.

## Syntaxe

```
ares oscili xamp, xcps [, ifn, iphs]
```

```
kres oscili kamp, kcps [, ifn, iphs]
```

## Initialisation

*ifn* (facultatif) -- numéro de la table de fonction. Nécessite un point de garde pour la lecture cyclique. Vaut -1 par défaut ce qui indique une onde sinusoïdale.

*iphs* (facultatif) -- phase initiale de la lecture, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative l'initialisation de la phase sera ignorée. La valeur par défaut est 0.

## Exécution

*kamp*, *xamp* -- amplitude

*kcps*, *xcps* -- fréquence en cycles par seconde.

*oscili* diffère de *oscil* en ce que la procédure standard d'utilisation d'une phase tronquée comme index de lecture est remplacée ici par une interpolation entre deux lectures successives. Les générateurs avec interpolation produiront un signal de sortie nettement plus propre, mais ils peuvent prendre jusqu'à deux fois plus de temps de calcul. On peut obtenir également ce type de précision sans le surcoût du calcul de l'interpolation en utilisant de grandes tables de fonction stockées de 2K, 4K ou 8K points, si l'on dispose de cet espace mémoire.

La table *ifn* est parcourue par incrément modulo la longueur de la table et la valeur obtenue est multipliée par *amp*.

Si vous désirez changer la table de l'oscillateur avec un signal de taux-k, vous pouvez utiliser *oscilkt*.

## Exemples

Voici un exemple de l'opcode *oscili*. Il utilise le fichier *oscili.csd* [examples/oscili.csd].

### Exemple 640. Exemple de l'opcode *oscili*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
```

```

<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o oscili.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kamp = .6
kcps = 440
ifn = p4

asig oscil kamp, kcps, ifn
outs asig,asig

endin

instr 2

kamp = .6
kcps = 440
ifn = p4

asig oscili kamp, kcps, ifn
outs asig,asig

endin
</CsInstruments>
<CsScore>
f1 0 128 10 1                                ; Sine with a small amount of data
f2 0 128 10 1 0.5 0.3 0.25 0.2 0.167 0.14 0.125 .111 ; Sawtooth with a small amount of data
f3 0 128 10 1 0 0.3 0 0.2 0 0.14 0 .111 ; Square with a small amount of data
f4 0 128 10 1 1 1 1 0.7 0.5 0.3 0.1 ; Pulse with a small amount of data

i 1 0 2 1
i 2 3 2 1
i 1 6 2 2
i 2 9 2 2
i 1 12 2 3
i 2 15 2 3
i 1 18 2 4
i 2 21 2 4

e
</CsScore>
</CsSoundSynthesizer>

```

## Voir aussi

*oscil, oscil3*

## Crédits

# oscilikt

oscilikt — Un oscillateur avec interpolation linéaire qui permet de changer le numéro de table au taux-k.

## Description

*oscilikt* ressemble beaucoup à *oscili*, mais il permet de changer le numéro de table au taux-k. Il est légèrement plus lent que *oscili* (spécialement avec des taux de contrôle élevés), mais en contrepartie il est plus précis car il utilise un accumulateur de phase sur 31 bit au lieu de celui sur 24 bit utilisé par *oscili*.

## Syntaxe

```
ares oscilikt xamp, xcps, kfn [, iphs] [, istor]
kres oscilikt kamp, kcps, kfn [, iphs] [, istor]
```

## Initialisation

*iphs* (facultatif, par défaut 0) -- phase initiale dans l'intervalle 0 à 1. Les autres valeurs sont ramenées cycliquement dans l'intervalle autorisé.

*istor* (facultatif, par défaut 0) -- ignorer l'initialisation.

## Exécution

*kamp*, *xamp* -- amplitude.

*kcps*, *xcps* -- fréquence en Hz. Zéro et les valeurs négatives sont permis. Cependant, la valeur absolue doit être inférieure à *sr* (et il est recommandé qu'elle soit inférieure à *sr/2*).

*kfn* -- numéro de la table de fonction. Peut varier au taux de contrôle (utile pour le « morphing » de formes d'onde, ou pour choisir parmi un ensemble de tables à bande de fréquence limitée générées par *GEN30*).

## Exemples

Voici un exemple de l'opcode *oscilikt*. Il utilise le fichier *oscilikt.csd* [examples/oscilikt.csd].

### Exemple 641. Exemple de l'opcode *oscilikt*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc          -d          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o oscilikt.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
```

```

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
    ; Generate a uni-polar (0-1) square wave.
    kamp1 init 1
    kcps1 init 2
    itype = 3
    ksquare lfo kamp1, kcps1, itype

    ; Use the square wave to switch between Tables #1 and #2.
    kamp2 init 20000
    kcps2 init 220
    kfn = ksquare + 1

    a1 oscilikt kamp2, kcps2, kfn
    out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine waveform.
f 1 0 4096 10 0 1
; Table #2: a sawtooth wave
f 2 0 3 -2 1 0 -1

; Play Instrument #1 for two seconds.
i 1 0 2

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*osciliktp* et *oscilikts*.

## Crédits

Auteur : Istvan Varga

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.22

# oscilikt

oscilikt — Un oscillateur avec interpolation linéaire qui permet la modulation de phase.

## Description

*oscilikt* permet la modulation de phase (qui est implémentée comme une modulation de fréquence au taux-*k*, en différenciant la phase en entrée). Le désavantage est qu'il n'y a pas de contrôle d'amplitude, et que la fréquence ne peut varier qu'au taux de contrôle. Cet opcode peut être plus rapide ou plus lent que *oscilikt*, en fonction du taux de contrôle.

## Syntaxe

```
ares oscilikt kcps, kfn, kphs [, istor]
```

## Initialisation

*istor* (facultatif, par défaut 0) -- ignorer l'initialisation.

## Exécution

*ares* -- signal de sortie au taux audio.

*kcps*, *xcps* -- fréquence en Hz. Zéro et les valeurs négatives sont permis. Cependant, la valeur absolue doit être inférieure à *sr* (et il est recommandé qu'elle soit inférieure à *sr/2*).

*kfn* -- numéro de la table de fonction. Peut varier au taux de contrôle (utile pour le « morphing » de formes d'onde, ou pour choisir parmi un ensemble de tables à bande de fréquence limitée générées par *GEN30*).

*kphs* -- phase (taux-*k*), l'intervalle attendu est 0 à 1. La valeur absolue de la différence entre les valeurs courante et précédente de *kphs* doit être inférieure à *ksmps*.

## Exemples

Voici un exemple de l'opcode *oscilikt*. Il utilise le fichier *oscilikt.csd* [examples/oscilikt.csd].

### Exemple 642. Exemple de l'opcode *oscilikt*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc          -d            ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o oscilikt.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
```

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1: oscilikt example
instr 1
  kphs line 0, p3, 4

  alx oscilikt 220.5, 1, 0
  aly oscilikt 220.5, 1, -kphs
  a1 = alx - aly

  out a1 * 14000
endin

</CsInstruments>
<CsScore>

; Table #1: Sawtooth wave
f 1 0 3 -2 1 0 -1

; Play Instrument #1 for four seconds.
i 1 0 4
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*oscilikt* et *oscilikts*.

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.22



# oscilikts

oscilikts — Un oscillateur avec interpolation linéaire et statut de synchronisation qui permet de changer le numéro de table au taux-k.

## Description

*oscilikts* est pareil à *oscilikt*. Sauf qu'il a une entrée de synchronisation que l'on peut utiliser pour réinitialiser l'oscillateur à une valeur de phase de taux-k. Il est plus lent que *oscilikt* et que *osciliktp*.

## Syntaxe

```
ares oscilikts xamp, xcps, kfn, async, kphs [, istor]
```

## Initialisation

*istor* (facultatif, par défaut 0) -- ignorer l'initialisation.

## Exécution

*xamp* -- amplitude.

*kcps*, *xcps* -- fréquence en Hz. Zéro et les valeurs négatives sont permis. Cependant, la valeur absolue doit être inférieure à *sr* (et il est recommandé qu'elle soit inférieure à *sr/2*).

*kfn* -- numéro de la table de fonction. Peut varier au taux de contrôle (utile pour le « morphing » de formes d'onde, ou pour choisir parmi un ensemble de tables à bande de fréquence limitée générées par *GEN30*).

*async* -- n'importe quelle valeur positive réinitialise la valeur de la phase de *oscilikts* à *kphs*. Zero ou des valeurs négatives n'ont aucun effet.

*kphs* -- fixe la phase, initialement et lorsqu'elle est réinitialisée avec *async*.

## Exemples

Voici un exemple de l'opcode *oscilikts*. Il utilise le fichier *oscilikts.csd* [examples/oscilikts.csd].

### Exemple 643. Exemple de l'opcode *oscilikts*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o oscilikts.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1: oscilikts example.
instr 1
  ; Frequency envelope.
  kfrq expon 400, p3, 1200
  ; Phase.
  kphs line 0.1, p3, 0.9

  ; Sync 1
  atmp1 phasor 100
  ; Sync 2
  atmp2 phasor 150
  async diff 1 - (atmp1 + atmp2)

  a1 oscilikts 14000, kfrq, 1, async, 0
  a2 oscilikts 14000, kfrq, 1, async, -kphs

  out a1 - a2
endin

</CsInstruments>
<CsScore>

; Table #1: Sawtooth wave
f 1 0 3 -2 1 0 -1

; Play Instrument #1 for four seconds.
i 1 0 4
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*oscilikt* et *osciliktp*.

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.22

# osciln

osciln — Lit des valeurs dans une table à une fréquence définie par l'utilisateur.

## Description

Lit des valeurs dans une table à une fréquence définie par l'utilisateur. On peut également écrire cet opcode comme *oscilx*.

## Syntaxe

```
ares osciln kamp, ifrq, ifn, itimes
```

## Initialisation

*ifrq, itimes* -- taux de lecture et nombre de passages à travers la table.

*ifn* -- numéro de la table de fonction.

## Exécution

*kamp* -- facteur d'amplitude

*osciln* parcourera plusieurs fois la table stockée en prélevant un échantillon *ifrq* fois par seconde, après quoi il retournera des zéros. Il génère seulement des signaux audio, avec les valeurs de sortie pondérées par *kamp*.

## Exemples

Voici un exemple de l'opcode *osciln*. Il utilise le fichier *osciln.csd* [examples/osciln.csd].

### Exemple 644. Exemple de l'opcode *osciln*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;;realtime audio out
;-iadc    ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o osciln.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gione ftgen 1, 0, 1024, 7, 0, 1, 1, 1024, 0
gitwo ftgen 2, 0, 1024, 7, 0, 512, 1, 512, 0
```

```
instr 1 ;very simple waveguide system

ifn      = p4
ipitch   = p5
itimes   = p6
iperiod  = 1000/ipitch

afeed    init    0
aimpl    osciln  1, ipitch, ifn, itimes    ;use as excitation signal
arefl    tone    aimpl + afeed, 4000
aout     atone    arefl, 5000
afeed    vdelay  arefl, iperiod, 10
          outs    aout*3, aout*3

endin
</CsInstruments>
<CsScore>

i 1 0  4 1 110 1 ;use different tables,
i 1 5  4 2 110 1 ;& different pitch
i 1 10 4 1 110 10 ;& different number of times the table is read
i 1 15 4 2 110 10
i 1 20 6 1 880 1
i 1 25 3 2 880 1
i 1 30 3 1 880 10
i 1 35 3 2 880 10

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*table, tablei, table3, oscill, oscilli*

# oscils

oscils — Un oscillateur sinus simple et rapide.

## Description

Oscillateur sinus simple et rapide, qui utilise seulement une multiplication et deux additions pour générer un échantillon en sortie, et qui ne nécessite pas de table de fonction.

## Syntaxe

```
ares oscils iamp, icps, iphs [, iflg]
```

## Initialisation

*iamp* -- amplitude en sortie.

*icps* -- fréquence en Hz (peut être nulle ou négative, cependant la valeur absolue doit être inférieure à  $sr/2$ ).

*iphs* -- phase initiale entre 0 et 1.

*iflg* -- somme des valeurs suivantes :

- 2 : utiliser la double précision même si Csound a été compilé pour utiliser des floats. Ceci améliore la qualité (spécialement dans le cas d'une longue exécution), mais le temps de calcul peut varier du simple au double.
- 1 : ignorer l'initialisation.

## Exécution

*ares* -- sortie audio

## Exemples

Voici un exemple de l'opcode oscils. Il utilise le fichier *oscils.csd* [examples/oscils.csd].

### Exemple 645. Exemple de l'opcode oscils.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o oscils.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
```

```
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

iflg = p4
asig oscils .7, 220, 0, iflg
    outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 2 0
i 1 3 2 2 ;double precision
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Istvan Varga  
Janvier 2002

Nouveau dans la version 4.18

## oscilx

oscilx — Identique à l'opcode osciln.

## Description

Voir l'opcode *osciln*.

# OSCBundle

OSCBundle — Envoie des données à d'autres processus en utilisant le protocole OSC en assemblant les messages dans un paquet.

## Description

Utilise le protocole OSC pour envoyer un ou plusieurs messages à d'autres processus OSC à l'écoute, dans un paquet unique. Contrairement à OSCsend il peut être utilisé pour envoyer plusieurs messages en même temps, mais seuls les types numériques standard d'OSC sont permis.

## Syntaxe

```
OSCBundle kwhen, ihost, iport, \  
          Sdest[], Stype[],kArgs[][][], isize
```

## Initialisation

*ihost* -- a string that is the intended host computer domain name. An empty string is interpreted as the current computer.

*iport* -- le numéro du port utilisé pour la communication.

*isize* -- taille maximale de paquet en octets, 65536 par défaut.

## Exécution

*kwhen* -- un paquet de messages est envoyé chaque fois que cette valeur change. Un message sera toujours envoyé lors du premier appel.

*Sdest*[] -- un tableau de chaînes de caractères contenant l'adresse de destination pour chaque message. Sa longueur doit être conforme à celle de *Stype*[].

*Stype*[] -- un tableau de chaînes de caractères contenant les types de chaque message. Seuls les types numériques ('i' pour entiers et 'f' pour flottants) sont supportés. Sa longueur doit être conforme à celle de *Sdest*[].

*kArgs*[][] -- un tableau bidimensionnel contenant les arguments pour chaque message. Sa dimension 1 (nombre de lignes) doit correspondre à celles de *Sdest*[] et de *Stype*[] . Chaque ligne doit contenir les arguments pour les types de chaque message. Si une ligne contient moins d'arguments, les places des données restantes sont remplies avec des zéros. Les arguments en sus de ceux que chaque chaîne de caractères demande sont ignorés.

## Exemple

Voici un exemple de l'opcode OSCBundle. Il utilise le fichier *oscbundle.csd* [examples/oscbundle.csd].

### Exemple 646. Exemple de l'opcode OSCBundle.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.



```

<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

instr 1
Sdest[] init 2
Stype[] init 2
kdata[][] init 2, 2

Sdest[0] = "/test/floats"
Sdest[1] = "/test/ints"
Stype[0] = "ff"
Stype[1] = "ii"

kdata fillarray 1,2,3,4

OSCbundle 1, "localhost", 7000, Sdest, Stype, kdata
endin

</CsInstruments>
<CsScore>
i1 0 1
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*OSClisten, OSCinit*

## Crédits

Auteur : Victor Lazzarini  
2018

# OSCcount

OSCcount — Donne le nombre de messages OSC pas encore lus.

## Description

Opcode du greffon osc.

Donne le nombre de messages OSC pas encore lus mais reçus par les récepteurs courants.

## Syntaxe

`kans OSCcount`

## Exécution

*kans* -- contient le nombre de messages acceptés par Csound sur n'importe quelle adresse mais pas encore présentés à un appel à OSClisten.

On peut utiliser cet opcode lorsqu'il y a plusieurs récepteurs et que l'on décide de les tester pour savoir s'il y a des messages.

## Exemple

Cet exemple montre une paire de nombres en virgule flottante reçus sur le port 7770.

```
sr = 44100
ksmps = 100
nchnls = 2

gihandle OSCinit 7770

instr 1
  kf1 init 0
  kf2 init 0
  kk OSCcount
nxtmsg:

if (kk == 0) goto ex
  kr OSClisten gihandle, "/foo/bar", "ff", kf1, kf2
  printk 0,kf1
  printk 0,kf2
  kk -= 1
  kgoto nxtmsg
ex:
  endin
```

## Voir aussi

*OSCsend, OSCinit OSClisten*

Pour plus d'information sur cet opcode : [http://www.youtube.com/watch?v=JX1C3TqP\\_9Y](http://www.youtube.com/watch?v=JX1C3TqP_9Y), par Andrés Cabrera

## Crédits

Auteur : John ffitch  
2018 nouveau dans Csound 6.12

# OSCinit

OSCinit — Démarre l'écoute des messages OSC sur un port particulier.

## Description

Opcodes du greffon osc.

Démarre un processus d'écoute qui peut être utilisé par *OSClisten*.

## Syntaxe

```
ihandle OSCinit iport
```

## Initialisation

*ihandle* -- identifiant retourné que l'on peut passer à n'importe quel nombre d'opcodes *OSClisten* pour recevoir des messages sur ce port.

*iport* -- le port sur lequel on écoute.

## Exécution

Le module d'écoute fonctionne en tâche de fond. Voir *OSClisten* pour les détails.

## Exemples

Cet exemple montre une paire de nombres en virgule flottante reçus sur le port 7770.

```
sr = 44100
ksmps = 100
nchnls = 2

gihandle OSCinit 7770

instr 1
  kf1 init 0
  kf2 init 0
nxtmsg:
  kk OSClisten gihandle, "/foo/bar", "ff", kf1, kf2
if (kk == 0) goto ex
  printk 0,kf1
  printk 0,kf2
  kgoto nxtmsg
ex:
  endin
```

## Voir aussi

*OSClisten*, *OSCsend*

Plus d'information sur cet opcode : [http://www.youtube.com/watch?v=JX1C3TqP\\_9Y](http://www.youtube.com/watch?v=JX1C3TqP_9Y), par Andrés Cabrera.

## Crédits

Auteur : John ffitch  
2005

# OSCinitM

OSCinitM — Démarre l'écoute des messages OSC multidiffusés sur un port particulier.

## Description

Opcodes du greffon osc.

Démarre un processus d'écoute de multidiffusion qui peut être utilisé par *OSClisten*.

## Syntaxe

```
ihandle OSCinitM Sgroup, iport
```

## Initialisation

*Sgroup* -- chaîne de caractères donnant l'adresse IP du groupe de multidiffusion.

*ihandle* -- identifiant retourné que l'on peut passer à n'importe quel nombre d'opcodes *OSClisten* pour recevoir des messages sur ce port.

*iport* -- le port sur lequel on écoute.

## Exécution

Le module d'écoute fonctionne en tâche de fond. Voir *OSClisten* pour les détails.

## Exemple

Cet exemple montre une paire de nombres en virgule flottante reçus sur le port 7770.

```
sr = 44100
ksmps = 100
nchnls = 2

gihandle OSCinitM "225.0.0.1", 7770

instr 1
  kf1 init 0
  kf2 init 0
nxtmsg:
  kk OSClisten gihandle, "/foo/bar", "ff", kf1, kf2
if (kk == 0) goto ex
  printk 0,kf1
  printk 0,kf2
  kgoto nxtmsg
ex:
  endin
```

## Voir aussi

*OSClisten*, *OSCsend* *OSCinit*

## Crédits

Auteur : John ffitch  
2016

# OSClisten

OSClisten — Ecoute les messages OSC sur un chemin particulier.

## Description

Opcodes du greffon osc.

Cherche à chaque cycle-k si un message OSC a été envoyé à un certain chemin d'un certain type.

## Syntaxe

```
kans OSClisten ihandle, idest, itype [, xdata1, xdata2, ...]  
kans, kdata[] OSClisten ihandle, idest, itype
```

## Initialisation

*ihandle* -- un identifiant retourné par un appel antérieur à *OSCinit*, pour associer *OSClisten* avec un numéro de port particulier.

*idest* -- une chaîne de caractères représentant l'adresse de destination. Elle est formatée comme un chemin préfixé par une barre oblique, avec d'éventuels sous-répertoires séparés par des barres obliques. Csound traite les messages entrants qui correspondent à cette adresse.

*itype* -- une chaîne de caractères indiquant le type des arguments optionnels à lire. La chaîne peut contenir les caractères "acdfhisAG" qui signifient audio, caractère, double, flottant, entier sur 64 bit, entier sur 32 bit, chaîne de caractères, tableau scalaire et ftable. Tous les types sauf 'asA' nécessitent une variable de taux-k ; 's' nécessite une variable chaîne de caractères, 'a' une variable audio et 'A' un tableau de taux-i ou de taux-k. Pour le type 'G' on peut utiliser une variable ou une constante.

Un identifiant est inséré dans le module d'écoute (voir *OSCinit*) pour intercepter les messages conformes à ce modèle.

## Exécution

*kans* -- fixé à 1 si un nouveau message a été reçu, ou 0 dans le cas contraire. Si plusieurs messages sont reçus dans une seule période de contrôle, les messages sont mis dans un tampon, et *OSClisten* peut être rappelé jusqu'à ce que 0 soit retourné.

S'il y avait un message les variables *xdata* reçoivent les valeurs en entrée, selon l'interprétation du paramètre *itype*. Noter que bien que les variables *xdata* soient situées à droite de l'opérateur, ce sont des sorties, et elles doivent donc être des variables de type a, ga, k, gk, S, gS, k[] ou gk[] et peut-être nécessiter une déclaration avec *init* ou = dans le cas des variables chaîne de caractères, avant l'appel à *OSClisten*.

## Exemples

L'exemple montre une paire de nombres en virgule flottante reçus sur le port 7770.

```
sr = 44100  
ksmps = 100
```



```
nchnls = 2

gihandle OSCinit 7770

instr 1
  kf1 init 0
  kf2 init 0
nxtmsg:
  kk OSClisten gihandle, "/foo/bar", "ff", kf1, kf2
if (kk == 0) goto ex
  printk 0,kf1
  printk 0,kf2
  kgoto nxtmsg
ex:
  endin
```

Ci-dessous deux fichiers .csd démontrent l'utilisation des opcodes OSC. Ils utilisent les fichiers *OSCmidisend.csd* [exemples/OSCmidisend.csd] et *OSCmidircv.csd* [exemples/OSCmidircv.csd].

### Exemple 647. Exemples des opcodes OSC.

Les deux fichiers .csd suivants démontrent l'utilisation des opcodes OSC dans Csound. Le premier fichier, *OSCmidisend.csd* [exemples/OSCmidisend.csd], transforme des messages MIDI reçus en données OSC. Le second fichier, *OSCmidircv.csd* [exemples/OSCmidircv.csd], peut prendre ces messages OSC et les interpréter pour générer du son à partir des messages de note, et stocker les valeurs de contrôleur. Il utilise le contrôleur 7 pour modifier le volume. Noter que ces fichiers sont conçus pour se trouver sur la même machine, mais si une adresse d'hôte différente (dans la macro IPADDRESS) est utilisée, ils peuvent se trouver sur différentes machines d'un réseau, ou connectées via l'internet.

Fichier CSD pour envoyer des messages OSC :

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
</CsOptions>
<CsInstruments>

  sr      = 44100
  ksmpps  = 128
  nchnls  = 1

; Example by David Akbari 2007
; Modified by Jonathan Murphy
; Use this file to generate OSC events for OSCmidircv.csd

#define IPADDRESS # "localhost" #
#define PORT      # 47120 #

turnon 1000

  instr 1000

    kst, kch, kd1, kd2  midiin

    OSCsend  kst+kch+kd1+kd2, $IPADDRESS, $PORT, "/midi", "iiii", kst, kch, kd1, kd2

  endin
```

```
</CsInstruments>
<CsScore>
f 0 3600 ;Dummy f-table
e
</CsScore>
</CsoundSynthesizer>
```

Fichier CSD pour recevoir des messages OSC :

```
<CsoundSynthesizer>
; network_recv.csd
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 128
nchnls  = 1

; Example by Jonathan Murphy and Andres Cabrera 2007
; Use file OSCmidisend.csd to generate OSC events for this file

0dbfs   = 1

gilisten  OSCinit  47120

gisin     ftgen     1, 0, 16384, 10, 1
givel     ftgen     2, 0, 128, -2, 0
gicc      ftgen     3, 0, 128, -7, 100, 128, 100 ;Default all controllers to 100

;Define scale tuning
giji_12   ftgen     202, 0, 32, -2, 12, 2, 256, 60, 1, 16/15, 9/8, 6/5, 5/4, 4/3, 7/5, \
              3/2, 8/5, 5/3, 9/5, 15/8, 2

#define DEST #"/midi"#
; Use controller number 7 for volume
#define VOL #7#

turnon 1000

instr 1000

kst      init      0
kch      init      0
kd1      init      0
kd2      init      0

next:

kk       OSClisten  gilisten, $DEST, "iiii", kst, kch, kd1, kd2

if (kk == 0) goto done

printks "kst = %i, kch = %i, kd1 = %i, kd2 = %i\\n", \
        0, kst, kch, kd1, kd2

if (kst == 176) then
;Store controller information in a table
tablew   kd2, kd1, gicc
endif

if (kst == 144) then
```

```

;Process noteon and noteoff messages.
kkey      = kd1
kvel      = kd2
kcps      cpstun    kvel, kkey, giji_12
kamp      = kvel/127

if (kvel == 0) then
    turnoff2 1001, 4, 1
elseif (kvel > 0) then
    event     "i", 1001, 0, -1, kcps, kamp
endif
endif

kgoto next ;Process all events in queue

done:
    endin

instr 1001 ;Simple instrument

icps      init      p4
kvol      table     $VOL, gicc ;Read MIDI volume from controller table
kvol      = kvol/127

aenv      linsegr    0, .003, p5, 0.03, p5 * 0.5, 0.3, 0
aosc      oscil      aenv, icps, gisin

    out      aosc * kvol
    endin

</CsInstruments>
<CsScore>
f 0 3600 ;Dummy f-table
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*OSCsend, OSCinit*

Plus d'information sur cet opcode : [http://www.youtube.com/watch?v=JX1C3TqP\\_9Y](http://www.youtube.com/watch?v=JX1C3TqP_9Y), par Andrés Cabrera.

## Crédits

Auteur : John ffitich

2005

Exemples par David Akbari, Andrés Cabrera et Jonathan Murphy 2007

Les types aAG sont nouveaux dans in Csound 6.07

# OSCraw

OSCraw — Ecoute tous les messages OSC sur un port donné.

## Description

A chaque cycle-k cherche si un message OSC a été reçu sur un port donné.

## Syntaxe

```
Smess[],klen OSCraw iport
```

## Initialisation

*iport* -- port sur lequel les messages seront reçus.

## Exécution

*Smess[]* -- un tableau de chaînes de caractères contenant les composantes du messages : adresse, types et éléments de données. Si le tableau n'existe pas (c'est-à-dire n'a pas été initialisé), il est créé avec 32 éléments. Les tableaux ne peuvent grossir avec la taille du message, si bien que si un message de plus de 32 éléments (c'est-à-dire 30 éléments de données séparés) est attendu, un tableau plus grand doit être initialisé avant utilisation. Si un message a plus d'éléments que le tableau ne peut en contenir, il est tronqué.

*klen* -- nombre d'éléments placés dans le tableau de sortie. Vaut 0 si aucun message n'a été reçu et au moins 2 si un message a été reçu (adresse et types sont les éléments minimaux).

### Exemple 648. Exemple.

Le fichier CSD suivant montre comment un message est reçu par OSCraw :

```
<CsoundSynthesizer>
<CsOptions>
-o dac
</CsOptions>
<CsInstruments>

instr 1
  kwhen init 0
  kmetro metro 1
  kwhen += kmetro
  OSCsend kwhen, "127.0.0.1",7771, "/foo/bar", "f", kwhen
endin

instr 2
top:
  Smess[],ka OSCraw 7771
  kn = 0
  while kn < ka do
    printf " : %s ", kn+1, Smess[kn]
    kn += 1
  od
  printf "%d items\n", ka, kn
  if ka > 0 kgoto top
endin
```

```
</CsInstruments>
<CsScore>
i1 0 10
i2 0 10
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*OSCsend, OSCinit*

Plus d'information sur cet opcode: [http://www.youtube.com/watch?v=JX1C3TqP\\_9Y](http://www.youtube.com/watch?v=JX1C3TqP_9Y), par Andrés Cabrera

## Crédits

Auteurs: Oeyvind Brandtsegg, Victor Lazzarini  
2017

Les types aAG sont nouveaux dans Csound 6.07

# OSCsend

OSCsend — Envoie des données à d'autres processus au moyen du protocole OSC.

## Description

Utilise le protocole OSC pour envoyer un message à d'autres processus d'écoute OSC.

## Syntaxe

```
OSCsend kwhen, ihost, iport, idestination[, itype , xdata1, xdata2, ...]
```

## Initialisation

*ihost* -- une chaîne de caractères donnant le nom de domaine de l'ordinateur hôte destinataire. Une chaîne vide est interprétée comme l'ordinateur courant.

*iport* -- le numéro du port utilisé pour la communication.

*idestination* -- une chaîne de caractères indiquant l'adresse de destination. Elle prend la forme d'un nom de fichier avec des répertoires. Csound ne fait que transmettre cette chaîne au code brut envoyé sans faire d'interprétation.

*itype* -- une chaîne de caractères indiquant le type des arguments facultatifs qui sont lus au taux-k. La chaîne peut contenir les caractères "abcdfilmstAG" pour audio, booléen, caractère, double, flottant, entier sur 32 bit, entier sur 64 bit, MIDI, chaîne de caractères, repère temporel, tableau de taux-k et ftable. Le message OSC peut n'avoir aucun type, auquel cas il se réduira à l'adresse de destination seule.

## Exécution

*kwhen* -- un message est envoyé chaque fois que cette valeur change. Un message sera toujours envoyé au premier appel.

Les données proviennent des valeurs de taux-k ou de taux-a qui suivent la chaîne de formatage. De même que pour le format dans printf, la série de caractères détermine l'interprétation des arguments. Noter qu'un repère temporel prend deux arguments.

## Exemples

L'exemple montre un simple instrument qui, lorsqu'il est appelé, envoie un groupe de trois messages à un ordinateur nommé "xenakis", sur le port 7770, à lire par un processus dont l'adresse est /foo/bar.

```
instr 1
  OSCsend 1, "xenakis.cs.bath.ac.uk", 7770, "/foo/bar", "sis", "FOO", 42, "bar"
endin
```

Voir la notice d'*OSClisten* pour un exemple d'envoi/réception en utilisant OSC.

## Voir aussi

*OSClisten*, *OSCinit*

Plus d'information sur cet opcode : [http://www.youtube.com/watch?v=JX1C3TqP\\_9Y](http://www.youtube.com/watch?v=JX1C3TqP_9Y), par Andrés Cabre-ra.

## Crédits

Auteur : John ffitch  
2005

Les types de données a, A et G ont été ajoutés dans la version 6.07 ; ils envoient les données via des blobs et ne sont réellement utiles qu'aux autres instances de Csound ou à des écouteurs particuliers.

# out32

out32 — Ecrit des données audio sur 32 canaux vers un périphérique externe ou un flot.

## Description

Ecrit des données audio sur 32 canaux vers un périphérique externe ou un flot.

## Syntaxe

```
out32 asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8, asig10, \  
      asig11, asig12, asig13, asig14, asig15, asig16, asig17, asig18, \  
      asig19, asig20, asig21, asig22, asig23, asig24, asig25, asig26, \  
      asig27, asig28, asig29, asig30, asig31, asig32
```

## Exécution

*out32* sort 32 canaux d'audio.

## Voir aussi

*outc, outch, outx, outz*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.07 de Csound



# out

out — Ecrit des données audio vers un périphérique externe ou un flot.

## Description

Ecrit des données audio vers un périphérique externe ou un flot, soit depuis des variables audio, soit depuis un tableau audio.

## Syntaxe

```
out asig1[, asig2,...]
```

```
out aarray
```

## Exécution

Envoie des échantillons audio dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Dans le cas d'un tableau, chaque élément du tableau est envoyé sur le canal correspondant. C'est un moyen d'utiliser de nombreux canaux. La version sans tableau est limitée à 1999.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*.

## Exemples

Voici un exemple de l'opcode out. Il utilise le fichier *out.csd* [examples/out.csd].

### Exemple 649. Exemple de l'opcode out.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o out.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 1
0dbfs = 1

instr 1

kamp = .6
kcps = 440
```

```

ifn  = p4

asig oscil kamp, kcps, ifn
    out asig ;one channel

endin
</CsInstruments>
<CsScore>
f1 0 16384 10 1                ; Sine
f2 0 16384 10 1 0.5 0.3 0.25 0.2 0.167 0.14 0.125 .111 ; Sawtooth

i 1 0 2 1
i 1 3 2 2

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*outh, outho, outq, outq1, outq2, outq3, outq4, outs, outs1, outs2, soundout*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
 MIT, Mills College  
 1993-1997  
 Author: John fitch  
 NUIM, 2013

Original dans Csound v1

Variante pour tableau ajoutée dans la version 6.01

# outc

outc — Ecrit des données audio sur un nombre arbitraire de canaux vers un périphérique externe ou un flot.

## Description

Ecrit des données audio sur un nombre arbitraire de canaux vers un périphérique externe ou un flot.

## Syntaxe

```
outc asig1 [, asig2] [...]
```

## Exécution

*outc* écrit autant de canaux que de variables fournies. Tous les canaux dépassant *nchnls* sont ignorés. Des zéros sont ajoutés si nécessaire.

## Exemples

Voici un exemple de l'opcode outc. Il utilise le fichier *outc.csd* [examples/outc.csd].

### Exemple 650. Exemple de l'opcode outc.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outc.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 5
0dbfs = 1

instr 1

asig vco2 .05, 30 ; sawtooth waveform at low volume

kcut line 100, p3, 30 ; Vary cutoff frequency
kresonance = 7
inumlayer = 2
asig lowresx asig, kcut, kresonance, inumlayer
; output same sound to 5 channels
    outc asig,asig,asig,asig,asig

endin
</CsInstruments>
<CsScore>
```

```
i 1 0 30  
e  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*out32, outch, outx, outz*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.07 de Csound

# outch

outch — Ecrit des données audio multi-canaux sous contrôle de l'utilisateur, vers un périphérique externe ou un flot.

## Description

Ecrit des données audio multi-canaux sous contrôle de l'utilisateur, vers un périphérique externe ou un flot.

## Syntaxe

```
outch kchan1, asig1 [, kchan2] [, asig2] [...]
```

## Exécution

*outch* envoie *asig1* sur le canal déterminé par *kchan1*, *asig2* sur le canal déterminé par *kchan2*, etc.



### Note

Le plus grand numéro de paramètre *kchanX* pour *outch* dépend de *nchnls*. Si *kchanX* est supérieur à *nchnls*, *asigX* sera silencieux. Noter que *outch* donnera dans ce cas un avertissement mais pas d'erreur.

## Exemples

Voici un exemple de l'opcode outch. Il utilise le fichier *outch.csd* [examples/outch.csd].

### Exemple 651. Exemple de l'opcode outch.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outch.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 5
0dbfs = 1

instr 1

asig vco2 .05, 100 ; sawtooth waveform at low volume

kcut line 100, p3, 30 ; Vary cutoff frequency
kresonance = .7
inumlayer = 3
```

```

asig lowresx asig, kcut, kresonance, inumlayer

klfo lfo 4, .5, 4
klfo = klfo+1 ; offset of 1
printks "signal is sent to channel %d\\n", .1, klfo
      outch klfo,asig

endin
</CsInstruments>
<CsScore>

i 1 0 30
e
</CsScore>
</CsoundSynthesizer>

signal is sent to channel 5
signal is sent to channel 4
signal is sent to channel 3
signal is sent to channel 2
signal is sent to channel 1
signal is sent to channel 5
.....

```

Voici un autre exemple de l'opcode outch. Il utilise le fichier *outch-2.csd* [examples/outch.csd].

### Exemple 652. Un autre exemple de l'opcode outch.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outch-2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 4
0dbfs = 1

seed      0

instr 1 ;random movements between 4 speakers with outch

ichn1     random    1, 4.999 ;channel to start
ichn2     random    1, 4.999 ;channel to end
prints     "Moving from speaker %d to speaker %d\\n", int(ichn1), int(ichn2)
asamp      soundin   "fox.wav"
kmov       linseg    0, p3, 1
a1, a2     pan2      asamp, kmov
           outch      int(ichn1), a1, int(ichn2), a2

endin

</CsInstruments>
<CsScore>
r 5
i 1 0 3
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*out32, outc, outx, outz*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/issue16/audiorouting.html> [<http://www.csoundjournal.com/issue16/audiorouting.html>], écrit par Andreas Russo.

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.07 de Csound

# outh

outh — Ecrit des données audio sur 6 canaux vers un périphérique externe ou un flot.

## Description

Ecrit des données audio sur 6 canaux vers un périphérique externe ou un flot.

## Syntaxe

```
outh asig1, asig2, asig3, asig4, asig5, asig6
```

## Exécution

Envoie des échantillons sur 6 canaux dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*.

## Voir aussi

*out, outo, outq, outq1, outq2, outq3, outq4, outs, outs1, outs2, soundout*

## Crédits

Auteur : John ffitch

Introduit avant la version 3



# outiat

outiat — Envoie des messages MIDI aftertouch au taux-i.

## Description

Envoie des messages MIDI aftertouch au taux-i.

## Syntaxe

```
outiat ichn, ivalue, imin, imax
```

## Initialisation

*ichn* -- numéro de canal MIDI (1-16)

*ivalue* -- valeur en virgule flottante

*imin* -- valeur minimale en virgule flottante (convertie en valeur entière MIDI 0)

*imax* -- valeur maximale en virgule flottante (convertie en valeur entière MIDI 127 (7 bit))

## Exécution

*outiat* envoie des messages aftertouch au taux-i. Il ne fonctionne qu'avec les instruments MIDI qui les reconnaissent. Il peut piloter une valeur différente de paramètre pour chaque note active.

Il peut échelonner un argument de taux-i en virgule flottante selon les valeurs *imin* et *imax*. Par exemple avec *imin* = 1.0 et *imax* = 2.0, lorsque l'argument *ivalue* reçoit la valeur 2.0, l'opcode envoie la valeur 127 sur le périphérique MIDI OUT. Lorsque l'argument *ivalue* reçoit la valeur 1.0, il envoie la valeur 0. Les opcodes de taux-i n'envoient leur message que pendant l'initialisation de l'instrument.

## Exemples

Voici un exemple de l'opcode *outiat*. Il utilise le fichier *outiat.csd* [examples/outiat.csd].

### Exemple 653. Exemple de l'opcode *outiat*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -Q1 -M0   ;;realtime audio out and midi in and out
;-iadc         ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outiat.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 10
```

```
nchnls = 2

instr 1

ikey notnum
ivel  veloc

ivib = 25 ;low value.
outiat 1, ivib, 0, 127 ;assign aftertouch on
print ivib ;external synth for example to
midion 1, ikey, ivel ;change depth of filter modulation

endin
</CsInstruments>
<CsScore>
f0 30 ;play for 30 seconds

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*outic14, outic, outipat, outipb, outipc, outkat, outkc14, outkc, outkpat, outkpb, outkpc*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# outic14

outic14 — Envoie une sortie de contrôleur MIDI sur 14 bit au taux-i.

## Description

Envoie une sortie de contrôleur MIDI sur 14 bit au taux-i.

## Syntaxe

```
outic14 ichn, imsb, ilsb, ivalue, imin, imax
```

## Initialisation

*ichn* -- numéro de canal MIDI (1-16)

*imsb* -- octet de poids fort du numéro de contrôleur lorsque l'on utilise des paramètres sur 14 bit ((0-127)

*ilsb* -- octet de poids faible du numéro de contrôleur lorsque l'on utilise des paramètres sur 14 bit ((0-127)

*ivalue* -- valeur en virgule flottante

*imin* -- valeur minimale en virgule flottante (convertie en valeur entière MIDI 0)

*imax* -- valeur maximale en virgule flottante (convertie en valeur entière MIDI 16383 (14-bit))

## Exécution

*outic14* envoie au taux-i une paire de messages de contrôleur. Cet opcode peut envoyer des paramètres sur 14 bit vers les instruments MIDI qui les reconnaissent. Le premier message de contrôle contient l'octet de poids fort de l'argument *ivalue* tandis que le second message contient l'octet de poids faible. *imsb* et *ilsb* sont respectivement les octets de poids fort et de poids faible du numéro de contrôleur.

Cet opcode peut piloter une valeur différente de paramètre pour chaque note active.

Il peut échelonner un argument de taux-i en virgule flottante selon les valeurs *imin* et *imax*. Par exemple avec *imin* = 1.0 et *imax* = 2.0, lorsque l'argument *ivalue* reçoit la valeur 2.0, l'opcode envoie la valeur 16383 sur le périphérique MIDI OUT. Lorsque l'argument *ivalue* reçoit la valeur 1.0, il envoie la valeur 0. Les opcodes de taux-i n'envoient leur message que pendant l'initialisation de l'instrument.

## Voir aussi

*outiat*, *outic*, *outipat*, *outipb*, *outipc*, *outkat*, *outkc14*, *outkc*, *outkpat*, *outkpb*, *outkpc*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# outic

outic — Envoie une sortie de contrôleur MIDI au taux-i.

## Description

Envoie une sortie de contrôleur MIDI au taux-i.

## Syntaxe

`outic ichn, inum, ivalue, imin, imax`

## Initialisation

*ichn* -- numéro de canal MIDI (1-16)

*inum* -- numéro du contrôleur (0-127 par exemple 1 = Mollette de Modulation, 2 = Contrôleur de Souffle, etc.)

*ivalue* -- valeur en virgule flottante

*imin* -- valeur minimale en virgule flottante (convertie en valeur entière MIDI 0)

*imax* -- valeur maximale en virgule flottante (convertie en valeur entière MIDI 127 (7 bit))

## Exécution

*outic* envoie au taux-i des messages de contrôleur sur le périphérique MIDI OUT. Il ne fonctionne qu'avec les instruments MIDI qui les reconnaissent. Il peut piloter une valeur différente de paramètre pour chaque note active.

Il peut échelonner un argument de taux-i en virgule flottante selon les valeurs *imin* et *imax*. Par exemple avec *imin* = 1.0 et *imax* = 2.0, lorsque l'argument *ivalue* reçoit la valeur 2.0, l'opcode envoie la valeur 127 sur le périphérique MIDI OUT. Lorsque l'argument *ivalue* reçoit la valeur 1.0, il envoie la valeur 0. Les opcodes de taux-i n'envoient leur message que pendant l'initialisation de l'instrument.

## Exemples

Voici un exemple de l'opcode *outic*. Il utilise le fichier *outic.csd* [examples/outic.csd].

### Exemple 654. Exemple de l'opcode *outic*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -Q1 -M0 ;;;realtime audio out -+rtmidi=virtual
;-iadc ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outic.wav -W ;;; for file output any platform
```

```
</CsOptions>
<CsInstruments>

  sr = 44100
  ksmpr = 32
  nchnls = 2

  instr 1

  ikey notnum
  ivel veloc
  kbrt = 40 ;set controller 74 (=brightness)
  outic 1, 74, kbrt, 0, 127 ;so filter closes a bit
  midion 1, ikey, ivel ;play external synth

endin
</CsInstruments>
<CsScore>
f0 30 ;runs 30 seconds

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*outiat, outic14, outipat, outipb, outipc, outkat, outkc14, outkc, outkpat, outkpb, outkpc*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# outipat

outipat — Envoie des messages MIDI d'aftertouch polyphonique au taux-i.

## Description

Envoie des messages MIDI d'aftertouch polyphonique au taux-i.

## Syntaxe

```
outipat ichn, inotenum, ivalue, imin, imax
```

## Initialisation

*ichn* -- numéro de canal MIDI (1-16)

*inotenum* -- numéro de note MIDI (utilisé dans les messages d'aftertouch polyphonique)

*ivalue* -- valeur en virgule flottante

*imin* -- valeur minimale en virgule flottante (convertie en valeur entière MIDI 0)

*imax* -- valeur maximale en virgule flottante (convertie en valeur entière MIDI 127 (7 bit))

## Exécution

*outipat* envoie des messages MIDI d'aftertouch polyphonique au taux-i. Il ne fonctionne qu'avec les instruments MIDI qui les reconnaissent. Il peut piloter une valeur différente de paramètre pour chaque note active.

Il peut échelonner un argument de taux-i en virgule flottante selon les valeurs *imin* et *imax*. Par exemple avec *imin* = 1.0 et *imax* = 2.0, lorsque l'argument *ivalue* reçoit la valeur 2.0, l'opcode envoie la valeur 127 sur le périphérique MIDI OUT. Lorsque l'argument *ivalue* reçoit la valeur 1.0, il envoie la valeur 0. Les opcodes de taux-i n'envoient leur message que pendant l'initialisation de l'instrument.

## Voir aussi

*outiat*, *outic14*, *outic*, *outipb*, *outipc*, *outkat*, *outkc14*, *outkc*, *outkpat*, *outkpb*, *outkpc*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# outipb

outipb — Envoie des messages MIDI de pitch-bend au taux-i.

## Description

Envoie des messages MIDI de pitch-bend au taux-i.

## Syntaxe

```
outipb ichn, ivalue, imin, imax
```

## Initialisation

*ichn* -- numéro de canal MIDI (1-16)

*ivalue* -- valeur en virgule flottante

*imin* -- valeur minimale en virgule flottante (convertie en valeur entière MIDI 0)

*imax* -- valeur maximale en virgule flottante (convertie en valeur entière MIDI 127 (7 bit))

## Exécution

*outipb* envoie des messages MIDI de pitch-bend au taux-i. Il ne fonctionne qu'avec les instruments MIDI qui les reconnaissent. Il peut piloter une valeur différente de paramètre pour chaque note active.

Il peut échelonner un argument de taux-i en virgule flottante selon les valeurs *imin* et *imax*. Par exemple avec *imin* = 1.0 et *imax* = 2.0, lorsque l'argument *ivalue* reçoit la valeur 2.0, l'opcode envoie la valeur 127 sur le périphérique MIDI OUT. Lorsque l'argument *ivalue* reçoit la valeur 1.0, il envoie la valeur 0. Les opcodes de taux-i n'envoient leur message que pendant l'initialisation de l'instrument.

## Exemples

Voici un exemple de l'opcode *outipb*. Il utilise le fichier *outipb.csd* [examples/outipb.csd].

### Exemple 655. Exemple de l'opcode *outipb*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -Q1 -M0   ;;realtime audio out and midi in and out
;-iadc         ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outipb.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 10
```



```

nchnls = 2

instr 1

ikey notnum
ivel veloc

ipb = 10 ;a little out of tune
outipb 1, ipb, 0, 127 ;(= pitchbend)
midion 1, ikey, ivel ;of external synth

endin
</CsInstruments>
<CsScore>
f0 30

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*outiat, outic14, outic, outipat, outipc, outkat, outkc14, outkc, outkpat, outkpb, outkpc*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# outipc

outipc — Envoie des messages MIDI de changement de programme au taux-i.

## Description

Envoie des messages MIDI de changement de programme au taux-i.

## Syntaxe

```
outipc ichn, iprog, imin, imax
```

## Initialisation

*ichn* -- numéro de canal MIDI (1-16)

*iprog* -- numéro de changement de programme en virgule flottante

*imin* -- valeur minimale en virgule flottante (convertie en valeur entière MIDI 0)

*imax* -- valeur maximale en virgule flottante (convertie en valeur entière MIDI 127 (7 bit))

## Exécution

*outipc* envoie des messages MIDI de changement de programme au taux-i. Il ne fonctionne qu'avec les instruments MIDI qui les reconnaissent. Il peut piloter une valeur différente de paramètre pour chaque note active.

Il peut échelonner un argument de taux-i en virgule flottante selon les valeurs *imin* et *imax*. Par exemple avec *imin* = 1.0 et *imax* = 2.0, lorsque l'argument *ivalue* reçoit la valeur 2.0, l'opcode envoie la valeur 127 sur le périphérique MIDI OUT. Lorsque l'argument *ivalue* reçoit la valeur 1.0, il envoie la valeur 0. Les opcodes de taux-i n'envoient leur message que pendant l'initialisation de l'instrument.

## Exemples

Voici un exemple de l'opcode outipc. Il utilise le fichier *outipc.csd* [examples/outipc.csd].

### Exemple 656. Exemple de l'opcode outipc.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -Q1 -M0   ;;realtime audio out and midi in and out
;-iadc          ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outipc.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
sr = 44100
ksmps = 32
nchnls = 2

instr 1

outipc 1, 80, 0, 127 ;program change --> 80
ikey notnum
ivel veloc
midion 1, ikey, ivel ;play external synth

endin
</CsInstruments>
<CsScore>
f0 30 ;runs 30 seconds

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*outiat, outic14, outic, outipat, outipb, outkat, outkc14, outkc, outkpat, outkpb, outkpc*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# outkat

outkat — Envoie des messages MIDI aftertouch au taux-k.

## Description

Envoie des messages MIDI aftertouch au taux-k.

## Syntaxe

`outkat kchn, kvalue, kmin, kmax`

## Exécution

*kchn* -- numéro de canal MIDI (1-16)

*kvalue* -- valeur en virgule flottante

*kmin* -- valeur minimale en virgule flottante (convertie en valeur entière MIDI 0)

*kmax* -- valeur maximale en virgule flottante (convertie en valeur entière MIDI 127)

*outkat* envoie des messages aftertouch au taux-k. Il ne fonctionne qu'avec les instruments MIDI qui les reconnaissent. Il peut piloter une valeur différente de paramètre pour chaque note active.

Il peut échelonner un argument de taux-k en virgule flottante selon les valeurs *kmin* et *kmax*. Par exemple avec *kmin* = 1.0 et *kmax* = 2.0, lorsque l'argument *kvalue* reçoit la valeur 2.0, l'opcode envoie la valeur 127 sur le périphérique MIDI OUT. Lorsque l'argument *kvalue* reçoit la valeur 1.0, il envoie la valeur 0. Les opcodes de taux-k envoient un message chaque fois que la valeur MIDI traduite de l'argument *kvalue* change.

## Exemples

Voici un exemple de l'opcode outkat. Il utilise le fichier *outkat.csd* [examples/outkat.csd].

### Exemple 657. Exemple de l'opcode outkat.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -Q1 -M0 ;;realtime audio out and midi in and out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outkat.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 10
nchnls = 2
```

```
instr 1

ikey notnum
ivel veloc

kvib linseg 100, .5, 120 ;vary aftertouch in .5 second
kvbr = int(kvib) ;whole numbers only
outkat 1, kvbr, 0, 127 ;assign aftertouch on
printk2 kvbr ;external synth for example to
midion 1, ikey, ivel ;change depth of filter modulation

endin
</CsInstruments>
<CsScore>
f0 30 ;play for 30 seconds

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*outiat, outic14, outic, outipat, outipb, outipc, outkc14, outkc, outkpat, outkpb, outkpc*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# outkc14

outkc14 — Envoie une sortie de contrôleur MIDI sur 14 bit au taux-k.

## Description

Envoie une sortie de contrôleur MIDI sur 14 bit au taux-k.

## Syntaxe

```
outkc14 kchn, kmsb, klsb, kvalue, kmin, kmax
```

## Exécution

*kchn* -- numéro de canal MIDI (1-16)

*kmsb* -- octet de poids fort du numéro de contrôleur lorsque l'on utilise des paramètres sur 14 bit ((0-127)

*klsb* -- octet de poids faible du numéro de contrôleur lorsque l'on utilise des paramètres sur 14 bit ((0-127)

*kvalue* -- valeur en virgule flottante

*kmin* -- valeur minimale en virgule flottante (convertie en valeur entière MIDI 0)

*kmax* -- valeur maximale en virgule flottante (convertie en valeur entière MIDI 16383 (14-bit))

*outkc14* envoie au taux-k une paire de messages de contrôleur. Cet opcode peut envoyer des paramètres sur 14 bit vers les instruments MIDI qui les reconnaissent. Le premier message de contrôle contient l'octet de poids fort de l'argument *kvalue* tandis que le second message contient l'octet de poids faible. *kmsb* et *klsb* sont respectivement les octets de poids fort et de poids faible du numéro de contrôleur.

Cet opcode peut piloter une valeur différente de paramètre pour chaque note active.

Il peut échelonner un argument de taux-k en virgule flottante selon les valeurs *kmin* et *kmax*. Par exemple avec *kmin* = 1.0 et *kmax* = 2.0, lorsque l'argument *kvalue* reçoit la valeur 2.0, l'opcode envoie la valeur 16383 sur le périphérique MIDI OUT. Lorsque l'argument *kvalue* reçoit la valeur 1.0, il envoie la valeur 0. Les opcodes de taux-k envoient un message chaque fois que la valeur MIDI traduite de l'argument *kvalue* change.

## Voir aussi

*outiat*, *outic14*, *outic*, *outipat*, *outipb*, *outipc*, *outkat*, *outkc*, *outkpat*, *outkpb*, *outkpc*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# outkc

outkc — Envoie des messages de contrôleur MIDI au taux-k.

## Description

Envoie des messages de contrôleur MIDI au taux-k.

## Syntaxe

**outkc** *kchn*, *knum*, *kvalue*, *kmin*, *kmax*

## Exécution

*kchn* -- numéro de canal MIDI (1-16)

*knum* -- numéro du contrôleur (0-127 par exemple 1 = Mollette de Modulation, 2 = Contrôleur de Souffle, etc.)

*kvalue* -- valeur en virgule flottante

*kmin* -- valeur minimale en virgule flottante (convertie en valeur entière MIDI 0)

*kmax* -- valeur maximale en virgule flottante (convertie en valeur entière MIDI 127 (7 bit))

*outkc* envoie au taux-k des messages de contrôleur sur le périphérique MIDI OUT. Il ne fonctionne qu'avec les instruments MIDI qui les reconnaissent. Il peut piloter une valeur différente de paramètre pour chaque note active.

Il peut échelonner un argument de taux-i en virgule flottante selon les valeurs *kmin* et *kmax*. Par exemple avec *kmin* = 1.0 et *kmax* = 2.0, lorsque l'argument *kvalue* reçoit la valeur 2.0, l'opcode envoie la valeur 127 sur le périphérique MIDI OUT. Lorsque l'argument *kvalue* reçoit la valeur 1.0, il envoie la valeur 0. Les opcodes de taux-k envoient un message chaque fois que la valeur MIDI traduite de l'argument *kvalue* change.

## Exemples

Voici un exemple de l'opcode outkc. Il utilise le fichier *outkc.csd* [examples/outkc.csd].

### Exemple 658. Exemple de l'opcode outkc.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -Q1 -M0   ;;realtime audio out and midi in and out
;-iadc         ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outkc.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
sr = 44100
ksmps = 32
nchnls = 2

instr 1

ikey notnum
ivel veloc

kcut linseg 100, .5, 20 ;vary controller in .5 second
kbrt = int(kcut) ;whole numbers only
outkc 1, 74, kbrt, 0, 127 ;controller 74 (= brightness)
midion 1, ikey, ivel ;of external synth

endin
</CsInstruments>
<CsScore>
f0 30 ;runs 30 seconds

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*outiat, outic14, outic, outipat, outipb, outipc, outkat, outkc14, outkpat, outkpb, outkpc*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.



# outkpat

outkpat — Envoie des messages MIDI d'aftertouch polyphonique au taux-k.

## Description

Envoie des messages MIDI d'aftertouch polyphonique au taux-k.

## Syntaxe

**outkpat** *kchn*, *knotenum*, *kvalue*, *kmin*, *kmax*

## Exécution

*kchn* -- numéro de canal MIDI (1-16)

*knotenum* -- numéro de note MIDI (utilisé dans les messages d'aftertouch polyphonique)

*kvalue* -- valeur en virgule flottante

*kmin* -- valeur minimale en virgule flottante (convertie en valeur entière MIDI 0)

*kmax* -- valeur maximale en virgule flottante (convertie en valeur entière MIDI 127 (7 bit))

*outkpat* envoie des messages MIDI d'aftertouch polyphonique au taux-k. Il ne fonctionne qu'avec les instruments MIDI qui les reconnaissent. Il peut piloter une valeur différente de paramètre pour chaque note active.

Il peut échelonner un argument de taux-k en virgule flottante selon les valeurs *kmin* et *kmax*. Par exemple avec *kmin* = 1.0 et *kmax* = 2.0, lorsque l'argument *kvalue* reçoit la valeur 2.0, l'opcode envoie la valeur 127 sur le périphérique MIDI OUT. Lorsque l'argument *kvalue* reçoit la valeur 1.0, il envoie la valeur 0. Les opcodes de taux-k envoient un message chaque fois que la valeur MIDI traduite de l'argument *kvalue* change.

## Voir aussi

*outiat*, *outic14*, *outic*, *outipat*, *outipb*, *outipc*, *outkat*, *outkc14*, *outkc*, *outkpb*, *outkpc*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# outkpb

outkpb — Envoie des messages MIDI de pitch-bend au taux-k.

## Description

Envoie des messages MIDI de pitch-bend au taux-k.

## Syntaxe

`outkpb kchn, kvalue, kmin, kmax`

## Exécution

*kchn* -- numéro de canal MIDI (1-16)

*kvalue* -- valeur en virgule flottante

*kmin* -- valeur minimale en virgule flottante (convertie en valeur entière MIDI 0)

*kmax* -- valeur maximale en virgule flottante (convertie en valeur entière MIDI 127 (7 bit))

*outkpb* envoie des messages MIDI de pitch-bend au taux-k. Il ne fonctionne qu'avec les instruments MIDI qui les reconnaissent. Il peut piloter une valeur différente de paramètre pour chaque note active.

Il peut échelonner un argument de taux-k en virgule flottante selon les valeurs *kmin* et *kmax*. Par exemple avec *kmin* = 1.0 et *kmax* = 2.0, lorsque l'argument *kvalue* reçoit la valeur 2.0, l'opcode envoie la valeur 127 sur le périphérique MIDI OUT. Lorsque l'argument *kvalue* reçoit la valeur 1.0, il envoie la valeur 0. Les opcodes de taux-k envoient un message chaque fois que la valeur MIDI traduite de l'argument *kvalue* change.

## Exemples

Voici un exemple de l'opcode outkpb. Il utilise le fichier *outkpb.csd* [examples/outkpb.csd].

### Exemple 659. Exemple de l'opcode outkpb.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -Q1 -M0 ;;realtime audio out and midi in and out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outkpb.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 10
nchnls = 2
```

```
instr 1

ikey notnum
ivel veloc

kpch linseg 100, 1, 0 ;vary in 1 second
kpb = int(kpch) ;whole numbers only
outkpb 1, kpb, 0, 127 ;(= pitchbend)
midion 1, ikey, ivel ;of external synth

endin
</CsInstruments>
<CsScore>
f0 30

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*outiat, outic14, outic, outipat, outipb, outipc, outkat, outkc14, outkc, outkpat, outkpc*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# outkpc

outkpc — Envoie des messages MIDI de changement de programme au taux-k.

## Description

Envoie des messages MIDI de changement de programme au taux-k.

## Syntaxe

```
outkpc kchn, kprog, kmin, kmax
```

## Exécution

*kchn* -- numéro de canal MIDI (1-16)

*kprog* -- numéro de changement de programme en virgule flottante

*kmin* -- valeur minimale en virgule flottante (convertie en valeur entière MIDI 0)

*kmax* -- valeur maximale en virgule flottante (convertie en valeur entière MIDI 127 (7 bit))

*outkpc* envoie des messages MIDI de changement de programme au taux-k. Il ne fonctionne qu'avec les instruments MIDI qui les reconnaissent. Il peut piloter une valeur différente de paramètre pour chaque note active.

Il peut échelonner un argument de taux-k en virgule flottante selon les valeurs *kmin* et *kmax*. Par exemple avec *kmin* = 1.0 et *kmax* = 2.0, lorsque l'argument *kvalue* reçoit la valeur 2.0, l'opcode envoie la valeur 127 sur le périphérique MIDI OUT. Lorsque l'argument *kvalue* reçoit la valeur 1.0, il envoie la valeur 0. Les opcodes de taux-k envoient un message chaque fois que la valeur MIDI traduite de l'argument *kvalue* change.

## Exemples

Voici un exemple de l'opcode outkpc. Il utilise le fichier *outkpc.csd* [examples/outkpc.csd].

### Exemple 660. Exemple de l'opcode outkpc.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

Cet exemple génère un changement de programme et une note sur le port de sortie MIDI de Csound chaque fois qu'une note est reçu sur le canal 1. Il faut que quelque chose soit connecté sur le port MIDI de sortie de Csound pour entendre le résultat.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc     -d         -M0  -Q1  ;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>
```

```

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

kprogram init 0

instr 1 ;Triggered by MIDI notes on channel 1

    ifund    notnum
    ivel    veloc
    idur = 1

; Sends a MIDI program change message according to
; the triggering note's velocity
outkpc      1 ,ivel ,0 ,127

noteondur   1 ,ifund ,ivel ,idur

endin

</CsInstruments>
<CsScore>
; Dummy ftable
f 0 60
</CsScore>
</CsoundSynthesizer>

```

Voici un autre exemple de l'opcode outkpc. Il utilise le fichier *outkpc\_fltk.csd* [examples/outkpc\_fltk.csd].

### Exemple 661. Exemple de l'opcode outkpc utilisant FLTK.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d           -M0  -Q1;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Example by Giorgio Zucco 2007

FLpanel "outkpc",200,100,90,90;start of container
gkpg, gihandle FLcount "Midi-Program change",0,127,1,5,1,152,40,16,23,-1
FLpanelEnd

FLrun

instr 1

ktrig changed gkpg
outkpc      ktrig,gkpg,0,127

endin

```

```
</CsInstruments>
<CsScore>
; Run instrument 1 for 60 seconds
i 1 0 60
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*outiat, outic14, outic, outipat, outipb, outipc, outkat, outkc14, outkc, outkpat, outkpb*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# outleta

outleta — Envoie un signal de taux-a depuis un port nommé d'un instrument.

## Description

Opcodes du greffon signalflowgraph.

Envoie un signal de taux-a depuis un port nommé d'un instrument.

## Syntaxe

```
outleta Sname, asignal
```

## Initialisation

*Sname* -- Nom sous forme de chaîne de caractères du port sortant. Le nom du connecteur sortant est qualifié implicitement par le nom ou le numéro de l'instrument, si bien qu'il est permis d'utiliser le même nom de connecteur sortant dans plus d'un instrument (mais par contre on ne peut pas utiliser deux fois le même nom de connecteur sortant dans un instrument).

## Exécution

*asignal* -- signal audio en sortie.

Durant l'exécution, le signal de taux-a passant par le connecteur sortant est envoyé à chaque instance d'un instrument contenant un connecteur entrant auquel ce connecteur sortant a été relié au moyen de l'opcode connect. Les signaux de tous les connecteurs sortants reliés à un connecteur entrant sont additionnés dans le connecteur entrant.

## Exemples

Voici un exemple de l'opcode outleta. Il utilise le fichier *outleta.csd* [examples/outleta.csd].

### Exemple 662. Exemple de l'opcode outleta.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outleta.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
```

```

connect "1", "Outl", "reverby", "InL"
connect "1", "Outr", "reverby", "InR"

alwayson "reverby", 1

instr 1

aIn diskin2 "fox.wav", 1
    outleta "Outl", aIn
    outleta "Outr", aIn

endin

instr reverby

aInL   inleta "InL"
aInR   inleta "InR"

al, ar reverbbsc aInL, aInR, 0.7, 21000
ifxlev = 0.5
al      = (aInL*ifxlev)+(al*(1-ifxlev))
ar      = (aInR*ifxlev)+(ar*(1-ifxlev))
    outs al, ar

endin
</CsInstruments>
<CsScore>

i 1 0 3
e4
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*outletk, outletkid, outletf, inleta, inletk, inletkid, inletf, connect, alwayson, figenonce.*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/issue13/signalFlowGraphOp-codes.html> [http://www.csoundjournal.com/issue13/signalFlowGraphOp-codes.html], écrit par Michael Gogins.

## Crédits

Par Michael Gogins, 2009



# outletf

outletf — Envoie un signal de taux-f (fsig) depuis un port nommé d'un instrument.

## Description

Opcodes du greffon signalflowgraph.

Envoie un signal de taux-f (fsig) depuis un port nommé d'un instrument.

## Syntaxe

```
outletf Sname, fsignal
```

## Initialisation

*Sname* -- Nom sous forme de chaîne de caractères du port sortant. Le nom du connecteur sortant est qualifié implicitement par le nom ou le numéro de l'instrument, si bien qu'il est permis d'utiliser le même nom de connecteur sortant dans plus d'un instrument (mais par contre on ne peut pas utiliser deux fois le même nom de connecteur sortant dans un instrument).

## Exécution

*fsignal* -- signal de taux-f (fsig) en sortie.

Durant l'exécution, le signal de taux-f passant par le connecteur sortant est envoyé à chaque instance d'un instrument contenant un connecteur entrant auquel ce connecteur sortant a été relié au moyen de l'opcode connect. Les signaux de tous les connecteurs sortants reliés à un connecteur entrant sont additionnés dans le connecteur entrant.

## Voir aussi

*outleta, outletk, outletkid, inleta, inletk, inletkid, inletf, connect, alwayson, ftgenonce.*

## Crédits

Par Michael Gogins, 2009

# outletk

outletk — Envoie un signal de taux-k depuis un port nommé d'un instrument.

## Description

Opcodes du greffon signalflowgraph.

Envoie un signal de taux-k depuis un port nommé d'un instrument.

## Syntaxe

```
outletk Sname, ksignal
```

## Initialisation

*Sname* -- Nom sous forme de chaîne de caractères du port sortant. Le nom du connecteur sortant est qualifié implicitement par le nom ou le numéro de l'instrument, si bien qu'il est permis d'utiliser le même nom de connecteur sortant dans plus d'un instrument (mais par contre on ne peut pas utiliser deux fois le même nom de connecteur sortant dans un instrument).

## Exécution

*ksignal* -- signal de taux-k en sortie.

Durant l'exécution, le signal de taux-k passant par le connecteur sortant est envoyé à chaque instance d'un instrument contenant un connecteur entrant auquel ce connecteur sortant a été relié au moyen de l'opcode connect. Les signaux de tous les connecteurs sortants reliés à un connecteur entrant sont additionnés dans le connecteur entrant.

## Exemples

Voici un exemple de l'opcode outletk. Il utilise le fichier *outletk.csd* [examples/outletk.csd].

### Exemple 663. Exemple de l'opcode outletk.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o inletk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
```

```

connect "bend", "bendout", "guitar", "bendin"

instr bend

kbend line p4, p3, p5
    outletk "bendout", kbend
endin

instr guitar

kbend inletk "bendin"
kpch pow 2, kbend/12
    printk2 kpch
asig oscili .4, 440*kpch, 1
    outs asig, asig
endin

</CsInstruments>
<CsScore>
f1 0 1024 10 1

i"guitar" 0 5 8.00
i"bend" 3 .2 -12 12
i"bend" 4 .1 -17 40
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*outleta, outletf, outletkid, inleta, inletk, inletkid, inletf, connect, alwayson, ftgenonce.*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/issue13/signalFlowGraphOp-codes.html> [<http://www.csoundjournal.com/issue13/signalFlowGraphOp-codes.html>], écrit par Michael Gogins.

## Crédits

Par Michael Gogins, 2009

# outletkid

outletkid — Envoie un signal de taux-k depuis un port nommé d'un instrument.

## Description

Opcode du greffon signalflowgraph.

Envoie un signal de taux-k depuis un port nommé d'un instrument.

## Syntaxe

```
outletkid Sname, SinstanceID, ksignal
```

## Initialisation

*Sname* -- Nom sous forme de chaîne de caractères du port sortant. Le nom du connecteur sortant est qualifié implicitement par le nom ou le numéro de l'instrument, si bien qu'il est permis d'utiliser le même nom de connecteur sortant dans plus d'un instrument (mais par contre on ne peut pas utiliser deux fois le même nom de connecteur sortant dans un instrument).

*SinstanceID* -- Nom sous forme de chaîne de caractères de l'ID de l'instance du port sortant. Cela permet au port entrant de distinguer différentes instances du port sortant, par exemple une instance du port sortant pourrait être créée par une note spécifiant un ID d'instance et une autre instance pourrait être créée par une note spécifiant un autre ID. On pourrait utiliser ceci pour situer différentes instances d'un instrument à différents points d'un espace Ambisonic dans un processeur d'effet de spatialisation.

## Exécution

*ksignal* -- signal de taux-k en sortie.

Durant l'exécution, le signal de taux-k passant par le connecteur sortant est envoyé à chaque instance d'un instrument contenant un connecteur entrant auquel ce connecteur sortant a été relié au moyen de l'opcode connect. Les signaux de tous les connecteurs sortants reliés à un connecteur entrant, mais seulement ceux qui ont un ID d'instance qui correspond, sont additionnés dans le connecteur entrant.

## Voir aussi

*outleta, outletf, inleta, inletk, inletkid, inletf, connect, alwayson, ftgenonce.*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/issue13/signalFlowGraphOp-codes.html> [http://www.csoundjournal.com/issue13/signalFlowGraphOp-codes.html], écrit par Michael Gogins.

## Crédits

Par Michael Gogins, 2009

# outletv

outletv — Envoie un signal, tableau de taux-a, depuis un port nommé d'un instrument.

## Description

Opcodes du greffon signalflowgraph.

Envoie un signal, tableau de taux-a, depuis un port nommé d'un instrument.

## Syntaxe

```
outletv Sname, array
```

## Initialisation

*Sname* -- Nom sous forme de chaîne de caractères du port sortant. Le nom du connecteur sortant est qualifié implicitement par le nom ou le numéro de l'instrument, si bien qu'il est permis d'utiliser le même nom de connecteur sortant dans plus d'un instrument (mais par contre on ne peut pas utiliser deux fois le même nom de connecteur sortant dans un instrument).

## Exécution

*array* -- tableau du signal audio en sortie.

Durant l'exécution, le tableau du signal de taux-a passant par le connecteur sortant est envoyé à chaque instance d'un instrument contenant un connecteur entrant auquel ce connecteur sortant a été relié au moyen de l'opcode connect. Les signaux de tous les connecteurs sortants reliés à un connecteur entrant sont additionnés dans le connecteur entrant. Les ports peuvent avoir n'importe quel nombre de canaux, mais le port du connecteur d'entrée doit avoir le même nombre de canaux que les ports des connecteurs de sortie.

## Voir aussi

*outleta outletf outletk outletkid inleta inletf inletk inletkid inletv connect alwayson fgenonce*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/issue13/signalFlowGraphOp-codes.html> [<http://www.csoundjournal.com/issue13/signalFlowGraphOp-codes.html>] , written by Michael Gogins

## Crédits

Par Michael Gogins 2009

# outo

outo — Ecrit des données audio sur 8 canaux vers un périphérique externe ou un flot.

## Description

Ecrit des données audio sur 8 canaux vers un périphérique externe ou un flot.

## Syntaxe

```
outo asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8
```

## Exécution

Envoie des échantillons sur 8 canaux dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*.

## Voir aussi

*out, outh, outq, outq1, outq2, outq3, outq4, outs, outs1, outs2, soundout*

## Crédits

Auteur : John ffitich

Nouveau après la 3.30

# outq1

outq1 — Écrit des échantillons sur le canal quadro n°1 d'un périphérique externe ou d'un flot.

## Description

Écrit des échantillons sur le canal quadro n°1 d'un périphérique externe ou d'un flot.

## Syntaxe

```
outq1 asig
```

## Exécution

Envoie des échantillons dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*. On peut choisir des opcodes pour envoyer le son sur un canal particulier : *outs1* envoie vers le canal stéréo n°1, *outq3* vers le canal quadro n°3, etc.

## Exemples

Voici un exemple de l'opcode outq1. Il utilise le fichier *outq1.csd* [examples/outq1.csd].

### Exemple 664. Exemple de l'opcode outq1.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outq1.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 4
0dbfs = 1

instr 1

asig vco2 .05, 30 ; sawtooth waveform at low volume

kcut line 60, p3, 300 ; Vary cutoff frequency
kresonance = 7
inumlayer = 2
asig lowresx asig, kcut, kresonance, inumlayer
```

```
    outq1 asig ; output channel 1

endin
</CsInstruments>
<CsScore>

i 1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*out, outh, outh, outq, outq2, outq3, outq4, outs, outs1, outs2, soundout*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997



# outq2

outq2 — Ecrit des échantillons sur le canal quadro n°2 d'un périphérique externe ou d'un flot.

## Description

Ecrit des échantillons sur le canal quadro n°2 d'un périphérique externe ou d'un flot.

## Syntaxe

```
outq2 asig
```

## Exécution

Envoie des échantillons dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*. On peut choisir des opcodes pour envoyer le son sur un canal particulier : *outs1* envoie vers le canal stéréo n°1, *outq3* vers le canal quadro n°3, etc.

## Exemples

Voici un exemple de l'opcode outq2. Il utilise le fichier *outq2.csd* [examples/outq2.csd].

### Exemple 665. Exemple de l'opcode outq2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outq2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 4
0dbfs = 1

instr 1

asig vco2 .05, 30 ; sawtooth waveform at low volume

kcut line 300, p3, 60 ; Vary cutoff frequency
kresonance = 7
inumlayer = 2
asig lowresx asig, kcut, kresonance, inumlayer
```

```
    outq2 asig ; output channel 2

endin
</CsInstruments>
<CsScore>

i 1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*out, outh, outh, outq, outq1, outq3, outq4, outs, outs1, outs2, soundout*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

# outq3

outq3 — Ecrit des échantillons sur le canal quadro n°3 d'un périphérique externe ou d'un flot.

## Description

Ecrit des échantillons sur le canal quadro n°3 d'un périphérique externe ou d'un flot.

## Syntaxe

```
outq3 asig
```

## Exécution

Envoie des échantillons dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*. On peut choisir des opcodes pour envoyer le son sur un canal particulier : *outs1* envoie vers le canal stéréo n°1, *outq3* vers le canal quadro n°3, etc.

## Exemples

Voici un exemple de l'opcode outq3. Il utilise le fichier *outq3.csd* [examples/outq3.csd].

### Exemple 666. Exemple de l'opcode outq3.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outq3.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 4
0dbfs = 1

instr 1

asig vco2 .05, 30 ; sawtooth waveform at low volume

kcut line 30, p3, 100 ; Vary cutoff frequency
kresonance = 7
inumlayer = 2
asig lowresx asig, kcut, kresonance, inumlayer
```

```
    outq3 asig ; output channel 3

endin
</CsInstruments>
<CsScore>

i 1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*out, outh, outh, outq, outq1, outq2, outq4, outs, outs1, outs2, soundout*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

# outq4

outq4 — Ecrit des échantillons sur le canal quadro n°4 d'un périphérique externe ou d'un flot.

## Description

Ecrit des échantillons sur le canal quadro n°4 d'un périphérique externe ou d'un flot.

## Syntaxe

```
outq4 asig
```

## Exécution

Envoie des échantillons dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*. On peut choisir des opcodes pour envoyer le son sur un canal particulier : *outs1* envoie vers le canal stéréo n°1, *outq3* vers le canal quadro n°3, etc.

## Exemples

Voici un exemple de l'opcode outq4. Il utilise le fichier *outq4.csd* [examples/outq4.csd].

### Exemple 667. Exemple de l'opcode outq4.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outq4.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 4
0dbfs = 1

instr 1

asig vco2 .05, 30 ; sawtooth waveform at low volume

kcut line 100, p3, 30 ; Vary cutoff frequency
kresonance = 7
inumlayer = 2
asig lowresx asig, kcut, kresonance, inumlayer
```

```
    outq4 asig ; output channel 4

endin
</CsInstruments>
<CsScore>

i 1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*out, outh, outh, outq, outq1, outq2, outq3, outs, outs1, outs2, soundout*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

# outq

outq — Ecrit des données audio sur 4 canaux vers un périphérique externe ou un flot.

## Description

Ecrit des données audio sur 4 canaux vers un périphérique externe ou un flot.

## Syntaxe

```
outq asig1, asig2, asig3, asig4
```

## Exécution

Envoie des échantillons sur 4 canaux dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*. On peut choisir des opcodes pour envoyer le son sur un canal particulier : *outs1* envoie vers le canal stéréo n°1, *outq3* vers le canal quadra n°3, etc.

## Exemples

Voici un exemple de l'opcode outq. Il utilise le fichier *outq.csd* [examples/outq.csd].

### Exemple 668. Exemple de l'opcode outq.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outq.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 4
0dbfs = 1

instr 1

asig vco2 .01, 110 ; sawtooth waveform at low volume

;filter the first channel
kcut1 line 60, p3, 300 ; Vary cutoff frequency
kresonance1 = 3
inumlayer1 = 3
```

```

asig1 lowresx asig, kcut1, kresonance1, inumlayer1

;filter the second channel
kcut2 line 300, p3, 60 ; Vary cutoff frequency
kresonance2 = 3
inumlayer2 = 3
asig2 lowresx asig, kcut2, kresonance2, inumlayer2

;filter the third channel
kcut3 line 30, p3, 100; Vary cutoff frequency
kresonance3 = 6
inumlayer3 = 3
asig3 lowresx asig, kcut3, kresonance3, inumlayer3
asig3 = asig3*.1 ; lower volume

;filter the fourth channel
kcut4 line 100, p3, 30; Vary cutoff frequency
kresonance4 = 6
inumlayer4 = 3
asig4 lowresx asig, kcut4, kresonance4, inumlayer4
asig4 = asig4*.1 ; lower volume

    outq asig1, asig2, asig3, asig4; output channels 1, 2, 3 & 4

endin
</CsInstruments>
<CsScore>

i 1 0 5
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*out, outh, outh, outq1, outq2, outq3, outq4, outs, outs1, outs2, soundout*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
 MIT, Mills College  
 1993-1997



# outrg

outrg — Permet la sortie dans un ensemble de canaux contigus sur le périphérique de sortie audio.

## Description

*outrg* sort les données audio dans un ensemble de canaux contigus sur le périphérique de sortie audio.

## Syntaxe

```
outrg kstart, aout1 [,aout2, aout3, ..., aoutN]
```

## Exécution

*kstart* - le numéro du premier canal du périphérique de sortie où écrire (les numéros des canaux commencent à 1, qui est le premier canal).

*aout1, aout2, ... aoutN* - les arguments contenant les données audio à sortir sur les canaux correspondants.

*outrg* permet la sortie vers un ensemble de canaux contigus du périphérique de sortie audio. *kstart* indique le premier canal où écrire (le canal 1 étant le premier canal). Il faut s'assurer que le nombre obtenu en ajoutant à *kstart* le nombre de canaux à écrire - 1 est  $\leq nchnls$ .

## Exemples

Voici un exemple de l'opcode *outrg*. Il utilise le fichier *outrg.csd* [examples/outrg.csd].

### Exemple 669. Exemple de l'opcode *outrg*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outrg.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 4      ;quad
0dbfs = 1

instr 1

kleft init 1
asig vco2 .5, 220 ;sawtooth
idur = p3/(nchnls-1)
knext init idur
kpos init 0
krate init 1/idur
```

```

kbase init 0
ktime timeinsts
if ktime>=knext then
    kleft = kleft + 1
    knext = knext + idur
    kpos = 0
    kbase = ktime
else
    kpos = (ktime-kbase)/idur
endif
printks "speaker %d position %f\n", 0, kleft, kpos
a1,a2 pan2 asig, kpos
outrg kleft, a1, a2
kpos = kbase/idur
endin

</CsInstruments>
<CsScore>

i 1 0 10
e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

speaker 1 position 0.000200
speaker 1 position 0.000400
speaker 1 position 0.000600
.....
speaker 1 position 1.000000
speaker 2 position 0.000000
speaker 2 position 0.000200
....
speaker 2 position 0.999800
speaker 3 position 0.000000
speaker 3 position 0.000200
....
speaker 3 position 0.999600
speaker 4 position 0.000000
.....

```

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# outs1

outs1 — Ecrit des échantillons vers le canal stéréo n°1 d'un périphérique externe ou d'un flot.

## Description

Ecrit des échantillons vers le canal stéréo n°1 d'un périphérique externe ou d'un flot.

## Syntaxe

```
outs1 asig
```

## Exécution

Envoie des échantillons dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*. On peut choisir des opcodes pour envoyer le son sur un canal particulier : *outs1* envoie vers le canal stéréo n°1, *outq3* vers le canal quadra n°3, etc.

## Exemples

Voici un exemple de l'opcode *outs1*. Il utilise le fichier *outs1.csd* [examples/outs1.csd].

### Exemple 670. Exemple de l'opcode *outs1*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outs1.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

asig vco2 .01, 110 ; sawtooth waveform at low volume
kcut line 60, p3, 300 ; Vary cutoff frequency
kresonance = 3
inumlayer = 3
asig lowresx asig, kcut, kresonance, inumlayer
outs1 asig ; output stereo channel 1 only
```

```
endin
</CsInstruments>
<CsScore>

i 1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*out, outh, outh, outq, outq1, outq2, outq3, outq4, outs, outs2, soundout*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

# outs2

outs2 — Ecrit des échantillons vers le canal stéréo n°2 d'un périphérique externe ou d'un flot.

## Description

Ecrit des échantillons vers le canal stéréo n°2 d'un périphérique externe ou d'un flot.

## Syntaxe

```
outs2 asig
```

## Exécution

Envoie des échantillons dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*. On peut choisir des opcodes pour envoyer le son sur un canal particulier : *outs1* envoie vers le canal stéréo n°1, *outq3* vers le canal quadro n°3, etc.

## Exemples

Voici un exemple de l'opcode outs2. Il utilise le fichier *outs2.csd* [examples/outs2.csd].

### Exemple 671. Exemple de l'opcode outs2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outs2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

asig vco2 .01, 110 ; sawtooth waveform at low volume
kcut line 300, p3, 60 ; Vary cutoff frequency
kresonance = 3
inumlayer = 3
asig lowresx asig, kcut, kresonance, inumlayer
outs2 asig ; output stereo channel 2 only
```

```
endin
</CsInstruments>
<CsScore>

i 1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*out, outh, outo, outq, outq1, outq2, outq3, outq4, outs, outs1, soundout*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

# outs

outs — Ecrit des données audio stéréo vers un périphérique externe ou un flot.

## Description

Ecrit des données audio stéréo vers un périphérique externe ou un flot.

## Syntaxe

```
outs asig1, asig2
```

## Exécution

Envoie des échantillons stéréo dans un tampon accumulateur de sortie (créé au début de l'exécution) qui sert à collecter la sortie de tous les instruments actifs avant que le son ne soit écrit sur disque. Il peut y avoir n'importe quel nombre de ces unités de sortie dans un instrument.

Le type (mono, stéréo, quadra, hexa ou octo) doit concorder avec *nchnls*. Mais à partir de la version 3.50, Csound essaiera de changer un opcode incorrect pour satisfaire l'instruction *nchnls*. On peut choisir des opcodes pour envoyer le son sur un canal particulier : *outs1* envoie vers le canal stéréo n°1, *outq3* vers le canal quadra n°3, etc.

## Exemples

Voici un exemple de l'opcode outs. Il utilise le fichier *outs.csd* [examples/outs.csd].

### Exemple 672. Exemple de l'opcode outs.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o outs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

asig vco2 .01, 110 ; sawtooth waveform at low volume
;filter a channel
kcut1 line 60, p3, 300 ; Vary cutoff frequency
kresonance1 = 3
inumlayer1 = 3
asig1 lowresx asig, kcut1, kresonance1, inumlayer1
```

```
;filter the other channel
kcut2 line 300, p3, 60 ; Vary cutoff frequency
kresonance2 = 3
inumlayer2 = 3
asig2 lowresx asig, kcut2, kresonance2, inumlayer2

    outs asig1, asig2 ; output both channels 1 & 2

endin
</CsInstruments>
<CsScore>

i 1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*out, outh, outho, outq, outq1, outq2, outq3, outq4, outs1, outs2, soundout*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997



# outvalue

outvalue — Envoie un signal de taux-k ou de taux-i ou une chaîne de caractères vers un canal défini par l'utilisateur.

## Description

Envoie un signal de taux-k ou de taux-i ou une chaîne de caractères vers un canal défini par l'utilisateur.

## Syntaxe

```
outvalue "channel name", ivalue
outvalue "channel name", kvalue
outvalue "channel name", "string"
```

## Exécution

*"channel name"* -- Un entier ou une chaîne de caractères (entre guillemets) représentant le canal.

*ivalue, kvalue* -- La valeur envoyée vers le canal.

*string* -- La constante ou la variable chaîne de caractères envoyée vers le canal.

## Exemples

Voici un exemple de l'opcode outvalue. Il utilise le fichier *outvalue.csd* [examples/outvalue.csd].

### Exemple 673. Exemple de l'opcode outvalue.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
;run this example in CsoundQt, a Csound editor that provides widgets
;make the Widgets-panel visible, by clicking the Widgets symbol in the menu or pressing (Alt+1).

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
; written by Alex Hofmann

instr 1 ;move fader

kMoveUp linseg 0, 3, 1, 1, 1, 0.5, 0
outvalue "movefader", kMoveUp
endin

</CsInstruments>
<CsScore>
i 1 0 5
```

```
e  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*invalue*

## Crédits

Auteur : Matt Ingalls

Version de taux-i ajoutée dans Csound 6.04

# outx

outx — Ecrit des données audio sur 16 canaux vers un périphérique externe ou un flot.

## Description

Ecrit des données audio sur 16 canaux vers un périphérique externe ou un flot.

## Syntaxe

```
outx asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8, \  
    asig9, asig10, asig11, asig12, asig13, asig14, asig15, asig16
```

## Exécution

*outx* sort 16 canaux d'audio.

## Voir aussi

*out32, outc, outch, outz*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.07 de Csound

# outz

outz — Ecrit des données audio multi-canaux depuis un tableau ZAK vers un périphérique externe ou un flot.

## Description

Ecrit des données audio multi-canaux depuis un tableau ZAK vers un périphérique externe ou un flot.

## Syntaxe

```
outz ksig1
```

## Exécution

*outz* envoie en sortie *nchnls* de données audio d'un tableau ZAK.

## Voir aussi

*out32*, *outc*, *outch*, *outx*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.06 de Csound

# p5gconnect

p5gconnect — Lit les données d'un contrôleur P5 Glove.

## Description

Opcode du greffon p5g.

Ouvre et interroge au taux-k un contrôleur P5 Glove (gant de réalité virtuelle).

## Syntaxe

p5gconnect

## Initialisation

L'opcode détecte un P5 Glove connecté à l'ordinateur par USB et lance un thread d'écoute pour interroger ce périphérique.

## Exécution

A chaque cycle de contrôle, le gant est interrogé sur sa position et sur l'état des doigts et des boutons. Ces valeurs sont lues par l'opcode *p5gdata*.

## Exemples

Voici un exemple de des opcodes p5g. Il utilise le fichier *p5g.csd* [exemples/p5g.csd].

### Exemple 674. Exemple de des opcodes p5g.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
--rtaudio=alsa -o dac:hw:0
</CsOptions>
<CsInstruments>
nchnls = 1
ksmps = 1000

#define P5G_BUTTONS      #0#
#define P5G_BUTTON_A    #1#
#define P5G_BUTTON_B    #2#
#define P5G_BUTTON_C    #4#
#define P5G_JUSTPUSH     #8#
#define P5G_JUSTPU_A     #9#
#define P5G_JUSTPU_B    #10#
#define P5G_JUSTPU_C    #12#
#define P5G_RELEASED     #16#
#define P5G_RELSED_A     #17#
#define P5G_RELSED_B     #18#
#define P5G_RELSED_C     #20#
#define P5G_FINGER_INDEX #32#
```

```

#define P5G_FINGER_MIDDLE #33#
#define P5G_FINGER_RING #34#
#define P5G_FINGER_PINKY #35#
#define P5G_FINGER_THUMB #36#
#define P5G_DELTA_X #37#
#define P5G_DELTA_Y #38#
#define P5G_DELTA_Z #39#
#define P5G_DELTA_XR #40#
#define P5G_DELTA_YR #41#
#define P5G_DELTA_ZR #42#
#define P5G_ANGLES #43#

gka init 0
gkp init 0

instr 1
    p5gconnect
    ka p5gdata $P5G_JUSTPU_A.
    kc p5gdata $P5G_BUTTON_C.
    ; If the A button is just pressed then activate a note
    if (ka==0) goto ee
    event "i", 2, 0, 2

ee:
    gka p5gdata $P5G_DELTA_X.
    gkp p5gdata $P5G_DELTA_Y.
    printk2 gka
    printk2 gkp
    if (kc==0) goto ff
    printks "turning off (%d)\n", 0, kc
    turnoff
ff:
endin

instr 2
    a1 oscil ampbfs(gkp), 440+100*gka, 1
    ; a1 oscil 10000, 440, 1
    out a1
endin

</CsInstruments>

<CsScore>
f1 0 4096 10 1
i1 0 300

</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

*p5gdata*,

## Crédits

Auteur : John ffitch  
 Codemist Ltd  
 2009

Nouveau dans la version 5.12

# p5gdata

p5gdata — Lit les champs de données d'un P5 Glove externe.

## Description

Opcode du greffon p5g.

Lit les champs de données d'un P5 Glove externe.

## Syntaxe

```
kres p5gdata kcontrol
```

## Initialisation

Cet opcode doit être utilisé en conjonction avec un opcode *p5gconnect* actif.

## Exécution

*kcontrol* -- le code du contrôle à lire

A chaque accès, un élément de données particulier du P5 Glove est lu. Les contrôles actuellement implémentés sont donnés ci-dessous, avec le nom de macro défini dans le fichier *p5g\_mac* :

0 (P5G\_BUTTONS) : retourne un motif de bit pour tous les boutons qui ont été pressés.

1 (P5G\_BUTTON\_A) : retourne 1 si le bouton a été pressé, sinon 0.

2 (P5G\_BUTTON\_B) : comme ci-dessus.

4 (P5G\_BUTTON\_C) : comme ci-dessus.

8 (P5G\_JUSTPUSH) : retourne un motif de bit pour tous les boutons qui viennent juste d'être pressés.

9 (P5G\_JUSTPU\_A) : retourne 1 si le bouton A vient juste d'être pressé.

10 (P5G\_JUSTPU\_B) : comme ci-dessus.

12 (P5G\_JUSTPU\_C) : comme ci-dessus.

16 (P5G\_RELEASED) : retourne un motif de bit pour tous les boutons qui viennent d'être relâchés.

17 (P5G\_RELSED\_A) : retourne 1 si le bouton A vient juste d'être relâché.

18 (P5G\_RELSED\_B) : comme ci-dessus.

20 (P5G\_RELSED\_C) : comme ci-dessus.

32 (P5G\_FINGER\_INDEX) : retourne la valeur de repliement de l'index.

33 (P5G\_FINGER\_MIDDLE) : comme ci-dessus.

34 (P5G\_FINGER\_RING) : comme ci-dessus.

35 (P5G\_FINGER\_PINKY) : comme ci-dessus avec le petit doigt.

36 (P5G\_FINGER\_THUMB): comme ci-dessus.

37 (P5G\_DELTA\_X) : la position X du gant.

38 (P5G\_DELTA\_Y) : la position Y du gant.

39 (P5G\_DELTA\_Z) : la position Z du gant.

40 (P5G\_DELTA\_XR) : le changement de l'axe des X (angle).

41 (P5G\_DELTA\_YR) : comme ci-dessus.

42 (P5G\_DELTA\_ZR) : comme ci-dessus.

43 (P5G\_ANGLES) : l'angle général.

## Exemples

Voir l'exemple de l'opcode *p5gconnect*.

## Voir aussi

*p5gconnect*,

## Crédits

Auteur : John ffitch  
Codemist Ltd  
2009

Nouveau version 5.12



# p

p — Montre la valeur contenu dans un p-champ donné.

## Description

Montre la valeur contenu dans un p-champ donné.

## Syntaxe

**p**(x)

Cette fonction tourne au taux-i et au taux-k.

## Initialisation

x -- le numéro du p-champ.

## Exécution

La valeur retournée par la fonction *p* est la valeur contenue dans un p-champ.

## Exemples

Voici un exemple de l'opcode p. Il utilise le fichier *p.csd* [examples/p.csd].

### Exemple 675. Exemple de l'opcode p.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o p.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Get the value in the fourth p-field, p4.
i1 = p(4)

print i1
endin
```

```
</CsInstruments>
<CsScore>

; p4 = value to be printed.
; Play Instrument #1 for one second, p4 = 50.375.
i 1 0 1 50.375
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celle-ci :

```
instr 1:  i1 = 50.375
```

## Crédits

Exemple écrit par Kevin Conder.

# pan2

pan2 — Distribue un signal audio sur deux canaux.

## Description

Distribue un signal audio sur deux canaux avec choix de la méthode.

## Syntaxe

```
a1, a2 pan2 asig, xp [, imode]
```

## Initialisation

*imode* (facultatif) -- mode de l'algorithme de positionnement stéréophonique. 0 pour un panoramique à puissance égale (harmonique), 1 pour la méthode de la racine carrée, 2 pour un panoramique simplement linéaire et 3 pour un autre panoramique à puissance égale (basé sur un UDO). La valeur par défaut est 0.

## Exécution

*pan2* prend en entrée le signal *asig* et le distribue sur ses deux sorties (essentiellement des haut-parleurs stéréo) en fonction du contrôle *xp* qui peut être de taux-k ou de taux-a. Une valeur de zéro pour *xp* indique complètement à gauche et 1 indique complètement à droite.

## Exemples

Voici un exemple de l'opcode *pan2*. Il utilise le fichier *pan2.csd* [examples/pan2.csd].

### Exemple 676. Exemple de l'opcode *pan2*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if real audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pan2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 2^10, 10, 1

instr 1

kline line 0, p3, 1      ; straight line
ain oscili .6, 440, giSine ; audio signal..
```

```
aL,aR pan2 ain, kline ; sent across image
outs aL, aR

endin
</CsInstruments>
<CsScore>
i1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch  
Université de Bath, Codemist Ltd.  
Bath, UK  
Septembre 2007

Nouveau dans la version 5.07

# pan

pan — Distribue un signal audio sur quatre canaux.

## Description

Distribue un signal audio sur quatre canaux avec contrôle de la localisation.

## Syntaxe

```
a1, a2, a3, a4 pan asig, kx, ky, ifn [, imode] [, ioffset]
```

## Initialisation

*ifn* -- numéro de la table de fonction d'un modèle décrivant l'augmentation d'amplitude dans le canal d'un haut-parleur lorsque le son se déplace vers celui-ci en provenance d'un haut-parleur voisin. Nécessite un point de garde.

*imode* (facultatif) -- mode des valeurs de position *kx*, *ky*. 0 signifie un indice brut, 1 signifie que les entrées sont normalisées (0 - 1). La valeur par défaut est 0.

*ioffset* (facultatif) -- indicateur de translation pour *kx*, *ky*. 0 implique que l'origine se trouve au canal 3 (arrière-gauche) ; 1 indique un glissement des axes au centre de la quadraphonie. La valeur par défaut est 0.

## Exécution

*pan* distribue son signal d'entrée *asig* sur quatre sorties (essentiellement des haut-parleurs quadraphoniques) en fonction des contrôles *kx* et *ky*. Avec une entrée normalisée (*imode*=1) et sans translation, les quatre positions de sortie sont dans l'ordre : avant-gauche à (0, 1), avant-droite à (1, 1), arrière-gauche à l'origine (0, 0) et arrière-droite à (1, 0). Dans la notation (*kx*, *ky*), les coordonnées *kx* et *ky*, chacune variant entre 0 et 1, contrôlent la position du son en largeur et en profondeur.

Le mouvement entre les haut-parleurs se fait par variation d'amplitude, contrôlée par la table de fonction *ifn*. Comme *kx* varie entre 0 et 1, la force du signal de droite augmentera de la valeur la plus à gauche dans la table jusqu'à la valeur la plus à droite, tandis que celle du signal de gauche progressera de la valeur de la table la plus à droite jusqu'à la plus à gauche. Pour un simple panoramique linéaire, la table peut contenir la fonction linéaire de 0 à 1. On obtiendra un panoramique plus correct maintenant une puissance constante en mémorisant le premier quadrant d'une sinusoïde. Comme *pan* pondère et tronque *kx* et *ky* lors de la lecture de la table, il vaut mieux utiliser une table de taille moyenne (disons 8193).

Les valeurs *kx*, *ky* ne sont pas restreintes à 0 - 1. Un mouvement circulaire passant par les quatre haut-parleurs (à l'intérieur) aura un diamètre de racine de deux, et pourra être défini par un cercle de rayon  $R = \text{racine } 1/2$  dont le centre se trouve en (0.5, 0.5). *kx*, *ky* seront alors donnés par  $R\cos(\text{angle})$ ,  $R\sin(\text{angle})$ , avec une origine implicite en (0.5, 0.5) (c'est-à-dire *ioffset* = 1). Les valeurs brutes non pondérées opèrent de la même manière. Les sons peuvent ainsi être positionnés partout dans le plan polaire ou cartésien ; les points se trouvant hors du carré des haut-parleurs sont projetés correctement sur le périmètre du carré pour un auditeur situé au centre.

## Exemples

Voici un exemple de l'opcode *pan*. Il utilise le fichier *pan.csd* [examples/pan.csd].

### Exemple 677. Exemple de l'opcode pan.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if real audio input is needed too
; For Non-realtime output leave only the line below:
; -o pan.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 4
0dbfs = 1

instr 1

kcps = p4
k1 phasor kcps ; "fraction" of circle - controls speed of rotation - can be negative
k2 tablei k1, 1, 1 ; sin of angle (sinusoid in f1)
k3 tablei k1, 1, 1, .25, 1 ; cos of angle (sin offset 1/4 circle)
arnd randomi 400, 1000, 50 ; produce random values
asig poscil .7, arnd, 1 ; audio signal..

a1,a2,a3,a4 pan asig, k2/2, k3/2, 2, 1, 1 ; sent in a circle (f2=1st quad sin)
outq a1, a2, a3, a4

endin
</CsInstruments>
<CsScore>

f1 0 8192 10 1
f2 0 8193 9 .25 1 0

i1 0 10 .2 ;move to the tight
i1 11 10 -.2 ;move to the left
e

</CsScore>
</CsoundSynthesizer>
```

## pareq

pareq — Implémentation des filtres égaliseurs paramétrique de Zoelzer.

### Description

Implémentation des filtres égaliseurs paramétrique de Zoelzer, avec quelques modifications par l'auteur.

La formule du filtre low shelf est :

```
omega = 2*pi*f/sr
K      = tan(omega/2)

b0      = 1 + sqrt(2*V)*K + V*K^2
b1      = 2*(V*K^2 - 1)
b2      = 1 - sqrt(2*V)*K + V*K^2

a0      = 1 + K/Q + K^2
a1      = 2*(K^2 - 1)
a2      = 1 - K/Q + K^2
```

La formule du filtre high shelf est :

```
omega = 2*pi*f/sr
K      = tan((pi-omega)/2)

b0      = 1 + sqrt(2*V)*K + V*K^2
b1      = -2*(V*K^2 - 1)
b2      = 1 - sqrt(2*V)*K + V*K^2

a0      = 1 + K/Q + K^2
a1      = -2*(K^2 - 1)
a2      = 1 - K/Q + K^2
```

La formule du filtre peak est :

```
omega = 2*pi*f/sr
K      = tan(omega/2)

b0 = 1 + V*K/2 + K^2
b1 = 2*(K^2 - 1)
b2 = 1 - V*K/2 + K^2

a0 = 1 + K/Q + K^2
a1 = 2*(K^2 - 1)
a2 = 1 - K/Q + K^2
```

### Syntaxe

```
ares pareq asig, kc, kv, kq [, imodel] [, iskip]
```

## Initialisation

*imode* (facultatif, 0 par défaut) -- mode opératoire

- 0 = Peak
- 1 = Low Shelf
- 2 = High Shelf

*iskip* (facultatif, 0 par défaut) -- s'il est différent de zéro, l'initialisation du filtre est ignorée. (Nouveau dans les versions 4.23f13 et 5.0 de Csound)

## Exécution

*kc* -- fréquence centrale dans le mode peak, fréquence de coupure dans le mode shelf.

*kv* -- importance du renforcement ou de l'atténuation. Une valeur inférieure à 1 produit une atténuation. Une valeur supérieure à 1 produit un renforcement. La valeur 1 donne une réponse plate.

*kq* -- Q du filter (racine carrée de 0.5 ne produit pas de résonance)

*asig* -- le signal entrant

## Exemples

Voici un exemple de l'opcode *pareq*. Il utilise le fichier *pareq.csd* [exemples/pareq.csd].

### Exemple 678. Exemple de l'opcode *pareq*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pareq.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

instr 15
  ifc      =      p4                ; Center / Shelf
  kq       =      p5                ; Quality factor sqrt(.5) is no resonance
  kv       =      ampdb(p6)         ; Volume Boost/Cut
  imode    =      p7                ; Mode 0=Peaking EQ, 1=Low Shelf, 2=High Shelf
  kfc      linseg ifc*2, p3, ifc/2
  asig     rand 5000                ; Random number source for testing
  aout     pareq asig, kfc, kv, kq, imode ; Parametric equalization
  outs     aout, aout              ; Output the results
endin
```



```
</CsInstruments>
<CsScore>

; SCORE:
;   Sta  Dur  Fcenter  Q          Boost/Cut(dB)  Mode
i15 0    1    10000    .2          12             1
i15 +    .    5000    .2          12             1
i15 .    .    1000    .707       -12             2
i15 .    .    5000    .1         -12             0
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Hans Mikelson  
Décembre 1998

Nouveau dans la version 3.50 de Csound.

# partials

partials — Analyse spectrale par suivi des partiels.

## Description

L'opcode *partials* prend en entrée deux flots de signal PV contenant les signaux AMP\_FREQ et AMP\_PHASE (comme ils sont générés par exemple par *pvsifd* ou dans le premier cas par *pvsanal*) et réalise une estimation et un suivi de partiels selon la méthode décrite dans Lazzarini et al, "Time-stretching using the Instantaneous Frequency Distribution and Partial Tracking", Proc. of ICMC05, Barcelone. Il génère un flot de signal PV TRACKS, contenant l'amplitude, la fréquence, la phase et un ID de piste pour chaque piste en sortie. Ce type de signal contient un nombre variable de pistes de sortie, limité par le nombre total de bins d'analyse contenus dans les entrées ( $\text{tailleTFR}/2 + 1$  bin). La seconde entrée (AMP\_PHASE) est facultative, car elle peut prendre le même signal que la première entrée. Cependant, dans ce cas, toute l'information de phase vaut NULL et l'on ne peut pas réaliser de resynthèse en utilisant l'information de phase.

## Syntaxe

```
ftrks partials ffr, fphs, kthresh, kminpts, kmaxgap, imaxtracks
```

## Exécution

*ftrks* -- flot pv de sortie au format TRACKS

*ffr* -- flot pv d'entrée au format AMP\_FREQ

*fphs* -- flot pv d'entrée au format AMP\_PHASE

*kthresh* -- seuil d'analyse, compris entre -1 et 1. S'il n'est pas négatif, le seuil d'analyse est relatif à la magnitude maximale dans chaque trame d'analyse ( $kthresh * max\_magnitude$ ). S'il est négatif, la valeur de seuil maximale est relative à 0dbfs ( $kthresh * 0dbfs$ ). Les pistes de niveau inférieur au seuil sont ignorées.

*kminpoints* -- nombre minimal de points temporels pour qu'une crête détectée engendre une piste (1 est le minimum). Comme cet opcode travaille avec des flots de signal, les nombres plus importants augmentent le délai entre l'entrée et la sortie car il faut attendre que le nombre minimum de points nécessaires soient acquis.

*kmaxgap* -- écart maximum entre les points temporels pour la poursuite de la piste ( $> 0$ ). Les pistes sans suite après *kmaxgap* sont ignorées.

*imaxtracks* -- nombre maximal de pistes d'analyse (nombre de bins  $\geq imaxtracks$ )

## Exemples

Voici un exemple de l'opcode partials. Il utilise le fichier *partials.csd* [examples/partial.csd].

### Exemple 679. Exemple de l'opcode partials.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if real audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o partials.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ain diskin2 "fox.wav", 1
fs1,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst partials fs1,fsi2,.003,1,3,500 ; partial tracking
aout resyn fst, 1, 1.5, 500, 1 ; resynthesis (up a 5th)
outs aout, aout

endin
</CsInstruments>
<CsScore>
f 1 0 4096 10 1

i 1 0 2.8
e
</CsScore>
</CsoundSynthesizer>

```

L'exemple ci-dessus montre le suivi de partiel d'un signal d'analyse ifd et sa resynthèse additive par phase cubique avec transposition de hauteur.

## Crédits

Auteur : Victor Lazzarini  
Juin 2005

Nouveau greffon dans la version 5.

Novembre 2004.

# partikkel

*partikkel* — Synthétiseur granulaire avec un contrôle "par grain" grâce à ses nombreux paramètres. Il a une entrée sync pour synchroniser son horloge interne de distribution des grains avec une horloge externe.

## Description

*partikkel* a été conçu après la lecture du livre de Curtis Road "Microsound", et le but était de créer un opcode capable de réaliser toutes les variétés temporelles de synthèse granulaire décrites dans ce livre. L'idée étant que la plupart des techniques ne diffèrent que par les valeurs des paramètres, et que si l'on a un opcode unique qui peut produire toutes les variétés de synthèse granulaire, l'interpolation entre ces techniques devient possible. La synthèse granulaire est parfois appelée synthèse par particules et il m'a semblé approprié de nommer l'opcode *partikkel* afin de le distinguer des autres opcodes granulaires.

Certains des paramètres d'entrée de *partikkel* sont des numéros de table, pointant sur des tables dans lesquelles sont mémorisées des valeurs pour les changements de paramètre "par grain". *partikkel* peut utiliser une période d'une forme d'onde ou des formes d'onde complexes (par exemple un son échantillonné) comme source de forme d'onde pour les grains. Chaque grain est constitué du mélange de 4 formes d'onde source. On peut accorder séparément la fréquence de base de chacune des 4 formes d'onde source. La modulation de fréquence à l'intérieur de chaque grain est activée via une entrée audio auxiliaire (*awavfm*). La synthèse par trainlet (un trainlet est un bref train d'impulsions) est possible, et les trainlets peuvent être mélangés avec des grains basés sur des tables d'onde. On peut utiliser jusqu'à 8 sorties audio séparées. Des valeurs de sortie fractionnaires distribuent un grain entre deux sorties et une table de "loi de panoramique" peut être conçue pour la répartition de l'amplitude entre une paire de sorties.

## Syntaxe

```
a1 [, a2, a3, a4, a5, a6, a7, a8] partikkel agrainfreq, \
    kdistribution, idisttab, async, kenv2amt, ienv2tab, ienv_attack, \
    ienv_decay, ksustain_amount, ka_d_ratio, kduration, kamp, igainmask, \
    kwavfreq, ksweepshape, iwavfreqstarttab, iwavfreqendtab, awavfm, \
    ifmampstab, kfmenv, icosine, ktraincps, knumpartials, kchroma, \
    ichannelmask, krandommask, kwaveform1, kwaveform2, kwaveform3, \
    kwaveform4, iwaveampstab, asamplepos1, asamplepos2, asamplepos3, \
    asamplepos4, kwavekey1, kwavekey2, kwavekey3, kwavekey4, imax_grains \
    [, iopcode_id, ipanlaws]
```

## Initialisation

*idisttab* -- numéro d'une table de fonction, distribution des déplacements aléatoires du grain dans le temps. Les valeurs de la table sont interprétées comme la "quantité de déplacement" pondérée par 1/(rythme des grains). Cela signifie qu'une valeur de 0,5 dans la table déplacera un grain de la moitié de la période du rythme des grains. Les valeurs de la table sont lues aléatoirement, et pondérées par *kdistribution*. Pour obtenir des résultats stochastiques réalistes, il vaut mieux ne pas utiliser une taille de table trop petite, car cela limite le nombre des valeurs de déplacement possibles. On peut l'exploiter à d'autres fins, par exemple utiliser des valeurs de déplacement quantifiées pour travailler avec des décalages contrôlés à partir de la période du rythme des grains. Si *kdistribution* est négatif, les valeurs de la table seront lues séquentiellement. On peut sélectionner une table par défaut au moyen du numéro de table -1, pour lequel *idisttab* fournit une distribution nulle (pas de déplacement).

*ienv\_attack* -- numéro d'une table de fonction, forme de l'attaque du grain. Il faut un point de garde d'extension. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *ienv\_attack* fournit une fenêtre rectangulaire (pas d'enveloppe).

*ienv\_decay* -- numéro d'une table de fonction, forme de la chute du grain. Il faut un point de garde d'extension. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *ienv\_decay* fournit une fenêtre rectangulaire (pas d'enveloppe).

*ienv2tab* -- numéro d'une table de fonction, enveloppe additionnelle appliquée au grain après les enveloppes d'attaque et de chute. On peut l'utiliser par exemple pour la synthèse par formant fof. Il faut un point de garde d'extension. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *ienv2tab* fournit une fenêtre rectangulaire (pas d'enveloppe).

*icosine* -- numéro d'une table de fonction, devant contenir un cosinus, utilisée pour les trainlets. La table doit avoir une taille d'au moins 2048 pour obtenir des trainlets de bonne qualité.

*igainmasks* -- numéro d'une table de fonction, gain par grain. La suite des valeurs dans la table a la signification suivante : la valeur d'indice 0 est le point de début d'une boucle de lecture des valeurs, la valeur d'indice 1 étant le point de fin de cette boucle. Les entrées aux autres indices contiennent les valeurs de gain (normalement dans l'intervalle 0 - 1, mais d'autres valeurs sont permises, les valeurs négatives inversant la phase de la forme d'onde du grain) pour une suite de grains ; ces valeurs sont lues au rythme des grains, ce qui permet une correspondance exacte de "gain par grain". Les points du début et de la fin de la boucle sont basés sur zéro avec une origine à l'indice 2, par exemple une valeur de début de boucle de 0 et une valeur de fin de boucle de 3 provoqueront la lecture des valeurs d'indice 2, 3, 4, 5 dans une boucle évoluant au rythme des grains. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *igainmasks* désactive le masquage du gain (tous les grains reçoivent un masque de gain égal à 1).

*ichannelmasks* -- numéro d'une table de fonction, voir *igainmasks* pour une description de la façon dont les valeurs sont lues dans la table. L'intervalle des valeurs va de 0 à N, où N est le nombre de canaux de sortie. Une valeur de zéro enverra le grain sur la sortie audio 1 de l'opcode. On peut utiliser des valeurs non entières, par exemple 3,5 répartira le grain également entre les sorties 4 et 5. La valeur de channelmask boucle entre la dernière et la première sorties, si bien qu'une valeur de N-0,5 mélange le grain également entre la dernière et la première sortie. Si l'on désire une autre loi de panoramique entre les sorties, on peut la décrire dans la table *ipanlaws*. L'utilisateur doit éviter les dépassements de niveau. L'opcode plantera si des valeurs dépassent le niveau maximal. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *ichannelmasks* désactive le masquage des canaux (tous les grains reçoivent un masque de canal de 0 et sont envoyés sur la sortie audio 1 de *partikkel*).

*iwavfreqstarttab* -- numéro d'une table de fonction, voir *igainmasks* pour une description de la façon dont les valeurs sont lues dans la table. Multiplicateur de la fréquence de départ de chaque grain. La hauteur glissera de la fréquence de départ jusqu'à la fréquence de fin suivant une droite ou une courbe fixée par *ksweepshape*. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *iwavfreqstarttab* fournit un multiplicateur de 1, désactivant toute modification de la fréquence de départ.

*iwavfreqendtab* -- numéro d'une table de fonction, voir *iwavfreqstarttab*. Multiplicateur de la fréquence de fin de chaque grain. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *iwavfreqendtab* fournit un multiplicateur de 1, désactivant toute modification de la fréquence de fin.

*ifmamptab* -- numéro d'une table de fonction, voir *igainmasks* pour une description de la façon dont les valeurs sont lues dans la table. Indice de modulation de fréquence par grain. Le signal *awayfm* sera multiplié par les valeurs lues dans cette table. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *ifmamptab* fournit 1 comme indice de modulation, activant la modulation de fréquence pour tous les grains.

*iwaveamptab* -- numéro d'une table de fonction, les indices sont parcourus de la même manière que pour *igainmasks*. La valeur d'indice 0 sert de point de début de boucle et la valeur d'indice 1 de point de fin. Les autres indices sont lus par groupes de 5, dans lesquels chaque valeur représente une valeur de gain pour chacune des 4 formes d'onde source, et la cinquième valeur représente l'amplitude de trainlet. On peut choisir une table par défaut en utilisant -1 comme numéro de ftable, pour lequel *iwaveamptab* fournit

un mélange égal des 4 formes d'onde source (chacune avec une amplitude de 0,5) et une amplitude de trainlet nulle.

Le calcul des trainlets étant très gourmand en ressources CPU, on peut éviter la plupart des calculs de trainlet en fixant *ktrainamp* à zéro. Les trainlets sont normalisés au niveau de crête (*ktrainamp*), en compensation des variations d'amplitude causées par les variations de *kpartials* et de *kchroma*.

*imax\_grains* -- nombre maximum de grains par k-période. Une grande valeur ne devrait pas affecter l'exécution, le dépassement de cette valeur conduira à l'effacement des grains les "plus anciens".

*iopcode\_id* -- identificateur de l'opcode, liant une instance de *partikkel* à une instance de *partikkelsync*, laquelle fournira en sortie des impulsions de déclenchement synchronisées pour le distributeur de grains de *partikkel*. La valeur par défaut est zéro, ce qui signifie aucune connexion à une instance de *partikkelsync*.

*ipanlaws* -- numéro de table de fonction. La table décrit la courbe de panoramique utilisée pour des valeurs fractionnaires de channelmask. Les valeurs fractionnaires de channelmask mélangent un grain sur deux sorties voisines, avec le gain relatif fixé par la valeur fractionnaire. Par défaut (si aucune table *ipanlaws* n'est décrite), une répartition linéaire du gain est utilisée, si bien qu'une valeur de channelmask de par exemple 1,5 distribue le grain avec un gain de 0,5 sur la sortie 2 et un gain de 0,5 sur la sortie 3. La table *ipanlaws* peut être utilisée pour décrire d'autres courbes de contrôle du gain (lois de panoramique). La table doit contenir huit courbes de contrôle de gain, chacune contrôlant le panoramique entre deux sorties voisines. Les courbes doivent apparaître l'une après l'autre dans la table, de manière concaténée. On peut utiliser GEN18 pour créer cette table à partir de tables de courbe de panoramique séparées (voir l'exemple ci-dessous). La première courbe décrit la loi de panoramique entre les sorties 1 et 2, la suivante entre les sorties 2 et 3, et ainsi de suite. La dernière courbe décrit la loi de panoramique entre la dernière et la première sortie. La table est indexée par la valeur de channelmask de façon à ce qu'une des sorties d'une paire gouvernée par la loi de panoramique utilise l'index ( $\text{tablesize}/8 * \text{channelmask}$ ) tandis que l'autre sortie lit les valeurs à l'index ( $\text{tablesize}/8 * (\text{int}(\text{channelmask} + 1) - \text{frac}(\text{channelmask}))$ ). Cela signifie que si la valeur de la loi de panoramique à mi-chemin entre ces deux masques de canal vaut par exemple 0,7 (ce qui donnerait approximativement une répartition égale de puissance), alors chacune de ces deux sorties utilisera 0,7 comme valeur du gain.

## Exécution

*xgrainfreq* -- nombre de grains par seconde. On peut spécifier une valeur nulle, ce qui déléguera la distribution des grains à l'entrée de synchronisation.

*async* -- entrée de synchronisation. Les valeurs entrées sont ajoutées à la phase de l'horloge interne du distributeur de grains, ce qui permet une synchronisation de tempo avec une horloge externe. Comme c'est un signal de taux-a, les entrées sont généralement des impulsions de longueur 1/sr. À l'aide de telles impulsions on peut "faire bouger" la phase interne en avant ou en arrière, ce qui permet une synchronisation plus ou moins forte. Des valeurs d'entrée négatives décrémentent la phase interne, tandis que des valeurs positives dans l'intervalle de 0 à 1 incrémentent la phase interne. Une valeur d'entrée de 1 forcera toujours *partikkel* à générer un grain. Si la valeur reste à 1, l'horloge interne du distributeur de grain marquera une pause mais tous les grains en cours d'exécution continueront jusqu'à leur terme.

*kdistribution* -- distribution périodique ou stochastique des grains, 0 = périodique. Le déplacement stochastique de grain est de l'ordre de *kdistribution/grainrate* secondes. Le profil de la distribution stochastique (distribution aléatoire) peut être fixé dans la table *idisttab*. Si *kdistribution* est négatif, le résultat est un déplacement temporel déterministe comme décrit par *idisttab* (lecture séquentielle des valeurs de déplacement). Le déplacement de grain maximum est limité dans tous les cas à 10 secondes, et un grain conservera les valeurs (durée, hauteur, etc) reçues lors de sa première génération (avant le déplacement temporel). Comme le déplacement de grain dépend du taux de grains, ce déplacement est indéfini pour un taux de grain de 0Hz et *kdistribution* est complètement désactivé dans ce cas.

*kenv2amt* -- dosage de l'enveloppe secondaire dans l'enveloppe de chaque grain. L'intervalle va de 0 à 1, où 0 signifie pas d'enveloppe secondaire (fenêtre rectangulaire), 0,5 provoquera une interpolation entre une fenêtre rectangulaire et la forme fixée par *ienv2tab*.

*ksustain\_amount* -- durée d'entretien exprimée comme une fraction de la durée du grain. C-à-d la proportion entre le temps d'enveloppe (attaque + chute) et le temps d'entretien. Le niveau d'entretien est celui de la dernière valeur de la ftable *ienv\_attack*.

*ka\_d\_ratio* -- proportion entre le temps d'attaque et le temps de chute. Par exemple, avec *ksustain\_amount* à 0,5 et *ka\_d\_ratio* à 0,5, l'enveloppe d'attaque de chaque grain prendra 25% de la durée du grain, l'amplitude maximale (entretien) sera tenue pendant 50% de la durée du grain, et l'enveloppe de chute prendra les 25% restants de la durée du grain.

*kduration* -- durée du grain en millisecondes.

*kamp* -- facteur de pondération de l'amplitude en sortie de l'opcode. Multiplié par l'amplitude de chaque grain lue à partir de *igainmasks*. La lecture de la forme d'onde source dans les grains peut consommer un nombre significatif de cycles CPU, spécialement si la durée de grain est longue, de nombreux grains se chevauchant. Si *kamp* vaut zéro la lecture de la forme d'onde dans les grains n'aura pas lieu (et aucun son ne sera évidemment généré). On peut utiliser cette possibilité comme une méthode de court-circuit "logiciel" si l'on veut garder l'opcode actif mais silencieux pendant un certain temps.

*kwavfreq* -- facteur de transposition. Multiplié par les valeurs de transposition de départ et de fin lues à partir de *iwavfreqstarttab* et de *iwavfreqendtab*.

*ksweepshape* -- forme de la progression de la transposition, contrôle la courbure de la progression de la transposition. Dans l'intervalle de 0 à 1. Avec les valeurs faibles, la transposition sera maintenue plus longtemps près de la valeur de départ puis ira rapidement vers la valeur de fin, tandis qu'avec les valeurs fortes la transposition ira tout de suite rapidement vers la valeur de fin. Une valeur de 0,5 donnera une progression linéaire. La valeur 0 supprimera la progression et ne gardera que la fréquence de départ, tandis que la valeur 1 supprimera la progression et ne gardera que la fréquence de fin. Le générateur de la progression peut être légèrement imprécis lorsqu'il atteint la fréquence finale si l'on utilise une courbe raide avec des grains très longs.

*awavfm* -- entrée audio pour la modulation de fréquence du grain.

*kfmenv* -- numéro d'une table de fonction, enveloppe du signal modulateur de la modulation de fréquence provoquant un changement de l'indice de modulation sur toute la durée du grain.

*ktraincps* -- fréquence fondamentale des trainlets.

*knumpartials* -- nombre de partiels dans les trainlets.

*kchroma* -- couleur spectrale des trainlets. Une valeur de 1 donne une amplitude égale à chaque partiel, des valeurs plus grandes réduiront l'amplitude des partiels inférieurs tout en renforçant l'amplitude des partiels supérieurs.

*krandommask* -- masquage aléatoire (escamotage) de grains individuels. Dans l'intervalle de 0 à 1, où la valeur 0 signifie pas de masquage (tous les grains sont joués), et la valeur 1 escamote tous les grains.

*kwaveform1* -- numéro de la table pour la forme d'onde source 1.

*kwaveform2* -- numéro de la table pour la forme d'onde source 2.

*kwaveform3* -- numéro de la table pour la forme d'onde source 3.

*kwaveform4* -- numéro de la table pour la forme d'onde source 4.

*asamplepos1* -- position de départ pour la lecture de la forme d'onde source 1 (comprise entre 0 et 1).

*asamplepos2* -- position de départ pour la lecture de la forme d'onde source 2.

*asamplepos3* -- position de départ pour la lecture de la forme d'onde source 3.

*asamplepos4* -- position de départ pour la lecture de la forme d'onde source 4.

*kwavekey1* -- hauteur originale de la forme d'onde source 1. On peut l'utiliser pour transposer chaque forme d'onde source indépendamment.

*kwavekey2* -- comme *kwavekey1*, mais pour la forme d'onde source 2.

*kwavekey3* -- comme *kwavekey1*, mais pour la forme d'onde source 3.

*kwavekey4* -- comme *kwavekey1*, mais pour la forme d'onde source 4.

## Exemples

Voici un exemple de l'opcode *partikkel*. Il utilise le fichier *partikkel.csd* [examples/partikkel.csd].

### Exemple 680. Exemple de l'opcode *partikkel*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if real audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o partikkel.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 20
nchnls = 2

giSine ftgen 0, 0, 65537, 10, 1
giCosine ftgen 0, 0, 8193, 9, 1, 1, 90

instr 1

kgrainfreq = 200 ; 4 grains per second
kdistribution = 0 ; periodic grain distribution
idisttab = -1 ; (default) flat distribution used for grain distribution
async = 0 ; no sync input
kenv2amt = 0 ; no secondary enveloping
ienv2tab = -1 ; default secondary envelope (flat)
ienv_attack = -1 ; ; default attack envelope (flat)
ienv_decay = -1 ; ; default decay envelope (flat)
ksustain_amount = 0.5 ; time (in fraction of grain dur) at sustain level for each grain
ka_d_ratio = 0.5 ; balance between attack and decay time
kduration = (0.5/kgrainfreq)*1000 ; set grain duration relative to grain rate
kamp = 5000 ; amp
igainmasks = -1 ; (default) no gain masking
kwavfreq = 440 ; fundamental frequency of source waveform
ksweepshape = 0 ; shape of frequency sweep (0=no sweep)
iwavfreqstarttab = -1 ; default frequency sweep start (value in table = 1, which give no frequency modif)
iwavfreqendtab = -1 ; default frequency sweep end (value in table = 1, which give no frequency modifi)
awavfm = 0 ; no FM input
ifmampmtab = -1 ; default FM scaling (=1)
kfmenv = -1 ; default FM envelope (flat)
```



```

icosine = giCosine ; cosine ftable
kTrainCps = kgrainfreq ; set trainlet cps equal to grain rate for single-cycle trainlet in each grain
knumpartials = 3 ; number of partials in trainlet
kchroma = 1 ; balance of partials in trainlet
ichannelmasks = -1 ; (default) no channel masking, all grains to output 1
krandommask = 0 ; no random grain masking
kwaveform1 = giSine ; source waveforms
kwaveform2 = giSine ;
kwaveform3 = giSine ;
kwaveform4 = giSine ;
iwaveamptab = -1 ; (default) equal mix of all 4 source waveforms and no amp for trainlets
asamplepos1 = 0 ; phase offset for reading source waveform
asamplepos2 = 0 ;
asamplepos3 = 0 ;
asamplepos4 = 0 ;
kwavekey1 = 1 ; original key for source waveform
kwavekey2 = 1 ;
kwavekey3 = 1 ;
kwavekey4 = 1 ;
imax_grains = 100 ; max grains per k period

asig partikkel kgrainfreq, kdistribution, idisttab, async, kenv2amt, ienv2tab, \
    ienv_attack, ienv_decay, ksustain_amount, ka_d_ratio, kduration, kamp, igainmasks, \
    kwavfreq, ksweepshape, iwavfreqstarttab, iwavfreqendtab, awavfm, \
    ifmamptab, kfmenv, icosine, kTrainCps, knumpartial, \
    kchroma, ichannelmasks, krandommask, kwaveform1, kwaveform2, kwaveform3, kwaveform4, \
    iwaveamptab, asamplepos1, asamplepos2, asamplepos3, asamplepos4, \
    kwavekey1, kwavekey2, kwavekey3, kwavekey4, imax_grains

outs asig, asig
endin

</CsInstruments>
<CsScore>
i1 0 5 ; partikkel
e
</CsScore>
</CsSoundSynthesizer>

```

Voici un autre exemple de l'opcode partikkel. Il utilise le fichier *partikkel-2.csd* [examples/partikkel-2.csd].

### Exemple 681. Exemple 2 de l'opcode partikkel.

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out
-odac ;;;RT audio
; For Non-realtime output leave only the line below:
; -o partikkel.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 20
nchnls = 2

; Example by Joachim Heintz and Oeyvind Brandtsegg 2008

giCosine ftgen 0, 0, 8193, 9, 1, 1, 90 ; cosine
giDisttab ftgen 0, 0, 32768, 7, 0, 32768, 1 ; for kdistribution
giFile ftgen 0, 0, 0, 1, "fox.wav", 0, 0, 0 ; soundfile for source waveform
giWin ftgen 0, 0, 4096, 20, 9, 1 ; grain envelope
giPan ftgen 0, 0, 32768, -21, 1 ; for panning (random values between 0 and 1)

```

```
; *****
; partikkel example, processing of soundfile
; uses the file "fox.wav"
; *****
instr 1

/*score parameters*/
ispeed = p4 ; 1 = original speed
igrainrate = p5 ; grain rate
igrainsize = p6 ; grain size in ms
icent = p7 ; transposition in cent
iposrand = p8 ; time position randomness (offset) of the pointer in ms
icentrand = p9 ; transposition randomness in cents
ipan = p10 ; panning narrow (0) to wide (1)
idist = p11 ; grain distribution (0=periodic, 1=scattered)

/*get length of source wave file, needed for both transposition and time pointer*/
ifilen tableng giFile
ifildur = ifilen / sr

/*sync input (disabled)*/
async = 0

/*grain envelope*/
kenv2amt = 1 ; use only secondary envelope
ienv2tab = giWin ; grain (secondary) envelope
ienv_attack = -1 ; default attack envelope (flat)
ienv_decay = -1 ; default decay envelope (flat)
ksustain_amount = 0.5 ; no meaning in this case (use only secondary envelope, ienv2tab)
ka_d_ratio = 0.5 ; no meaning in this case (use only secondary envelope, ienv2tab)

/*amplitude*/
kamp = 0.4*0dbfs ; grain amplitude
igainmasks = -1 ; (default) no gain masking

/*transposition*/
kcentrand rand icentrand ; random transposition
iorig = 1 / ifildur ; original pitch
kwavfreq = iorig * cent(icent + kcentrand)

/*other pitch related (disabled)*/
ksweepshape = 0 ; no frequency sweep
iwavfreqstarttab = -1 ; default frequency sweep start
iwavfreqendtab = -1 ; default frequency sweep end
awavfm = 0 ; no FM input
ifmamptab = -1 ; default FM scaling (=1)
kfmenv = -1 ; default FM envelope (flat)

/*trainlet related (disabled)*/
icosine = giCosine ; cosine ftable
kTrainCps = igrainrate ; set trainlet cps equal to grain rate for single-cycle trainlet in each grain
knumpartials = 1 ; number of partials in trainlet
kchroma = 1 ; balance of partials in trainlet

/*panning, using channel masks*/
imid = .5; center
ileftmost = imid - ipan/2
irightmost = imid + ipan/2
giPanthis ftgen 0, 0, 32768, -24, giPan, ileftmost, irightmost ; rescales giPan according to ipan
tableiw 0, 0, giPanthis ; change index 0 ...
tableiw 32766, 1, giPanthis ; ... and 1 for ichannelmasks
ichannelmasks = giPanthis ; ftable for panning

/*random gain masking (disabled)*/
krandommask = 0
```

```

/*source waveforms*/
kwaveform1 = giFile ; source waveform
kwaveform2 = giFile ; all 4 sources are the same
kwaveform3 = giFile
kwaveform4 = giFile
iwaveamptab = -1 ; (default) equal mix of source waveforms and no amplitude for trainlets

/*time pointer*/
afilposphas phasor ispeed / ifildur
/*generate random deviation of the time pointer*/
iposrandsec = iposrand / 1000 ; ms -> sec
iposrand = iposrandsec / ifildur ; phase values (0-1)
krndpos linrand iposrand ; random offset in phase values
/*add random deviation to the time pointer*/
asamplepos1 = afileposphas + krndpos; resulting phase values (0-1)
asamplepos2 = asamplepos1
asamplepos3 = asamplepos1
asamplepos4 = asamplepos1

/*original key for each source waveform*/
kwavekey1 = 1
kwavekey2 = kwavekey1
kwavekey3 = kwavekey1
kwavekey4 = kwavekey1

/* maximum number of grains per k-period*/
imax_grains = 100

aL, aR partikkel igrainrate, idist, giDisttab, async, kenv2amt, ienv2tab, \
    ienv_attack, ienv_decay, ksustain_amount, ka_d_ratio, igrainsize, kamp, igainmasks, \
    kwavfreq, ksweepshape, iwavfreqstarttab, iwavfreqendtab, awavfm, \
    ifmampstab, kfmenv, icosine, kTrainCps, knumpartials, \
    kchroma, ichannelmasks, krandommask, kwaveform1, kwaveform2, kwaveform3, kwaveform4, \
    iwaveamptab, asamplepos1, asamplepos2, asamplepos3, asamplepos4, \
    kwavekey1, kwavekey2, kwavekey3, kwavekey4, imax_grains

outs aL, aR

endin

</CsInstruments>
<CsScore>
;il st dur speed grate gsize cent posrnd cntrnd pan dist
i1 0 2.757 1 200 15 0 0 0 0 0
s
i1 0 2.757 1 200 15 400 0 0 0 0
s
i1 0 2.757 1 15 450 400 0 0 0 0
s
i1 0 2.757 1 15 450 400 0 0 0 0.4
s
i1 0 2.757 1 200 15 0 400 0 0 1
s
i1 0 5.514 .5 200 20 0 0 600 .5 1
s
i1 0 11.028 .25 200 15 0 1000 400 1 1

</CsScore>

</CsoundSynthesizer>

```

Voici un exemple utilisant des lois de panoramique avec channelmasks dans partikkel. Il utilise le fichier *partikkel-panlaws.csd* [examples/partikkel-panlaws.csd].

**Exemple 682. Exemple avec des lois de panoramique avec des masques de canaux.**

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out
-odac          ;;RT audio
; For Non-realtime ouput leave only the line below:
; -o partikkel.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 20
nchnls = 2

; Example by Joachim Heintz and Oeyvind Brandtsegg 2008

giCosine ftgen 0, 0, 8193, 9, 1, 1, 90 ; cosine
giDisttab ftgen 0, 0, 32768, 7, 0, 32768, 1 ; for kdistribution
giFile ftgen 0, 0, 0, 1, "fox.wav", 0, 0, 0 ; soundfile for source waveform
giWin ftgen 0, 0, 4096, 20, 9, 1 ; grain envelope
giPan ftgen 0, 0, 32768, -21, 1 ; for panning (random values between 0 and 1)

; *****
; partikkel example, processing of soundfile
; uses the file "fox.wav"
; *****
instr 1

/*score parameters*/
ispeed = p4 ; 1 = original speed
igrainrate = p5 ; grain rate
igrainsize = p6 ; grain size in ms
icent = p7 ; transposition in cent
iposrand = p8 ; time position randomness (offset) of the pointer in ms
icentrand = p9 ; transposition randomness in cents
ipan = p10 ; panning narrow (0) to wide (1)
idist = p11 ; grain distribution (0=periodic, 1=scattered)

/*get length of source wave file, needed for both transposition and time pointer*/
ifilen tableng giFile
ifildur = ifilen / sr

/*sync input (disabled)*/
async = 0

/*grain envelope*/
kenv2amt = 1 ; use only secondary envelope
ienv2tab = giWin ; grain (secondary) envelope
ienv_attack = -1 ; default attack envelope (flat)
ienv_decay = -1 ; default decay envelope (flat)
ksustain_amount = 0.5 ; no meaning in this case (use only secondary envelope, ienv2tab)
ka_d_ratio = 0.5 ; no meaning in this case (use only secondary envelope, ienv2tab)

/*amplitude*/
kamp = 0.4*0dbfs ; grain amplitude
igainmasks = -1 ; (default) no gain masking

/*transposition*/
kcentrand rand icentrand ; random transposition
iorig = 1 / ifildur ; original pitch
kwavfreq = iorig * cent(icent + kcentrand)

```

```

/*other pitch related (disabled)*/
ksweepshape = 0 ; no frequency sweep
iwavfreqstarttab = -1 ; default frequency sweep start
iwavfreqendtab = -1 ; default frequency sweep end
awavfm = 0 ; no FM input
ifmampstab = -1 ; default FM scaling (=1)
kfmenv = -1 ; default FM envelope (flat)

/*trainlet related (disabled)*/
icosine = giCosine ; cosine ftable
kTrainCps = igrainrate ; set trainlet cps equal to grain rate for single-cycle trainlet in each grain
knumpartials = 1 ; number of partials in trainlet
kchroma = 1 ; balance of partials in trainlet

/*panning, using channel masks*/
imid = .5; center
ileftmost = imid - ipan/2
irightmost = imid + ipan/2
giPanthis ftgen 0, 0, 32768, -24, giPan, ileftmost, irightmost ; rescales giPan according to ipan
tableiw 0, 0, giPanthis ; change index 0 ...
tableiw 32766, 1, giPanthis ; ... and 1 for ichannelmasks
ichannelmasks = giPanthis ; ftable for panning

/*random gain masking (disabled)*/
krandommask = 0

/*source waveforms*/
kwaveform1 = giFile ; source waveform
kwaveform2 = giFile ; all 4 sources are the same
kwaveform3 = giFile
kwaveform4 = giFile
iwaveampstab = -1 ; (default) equal mix of source waveforms and no amplitude for trainlets

/*time pointer*/
afilposphas phasor ispeed / ifildur
/*generate random deviation of the time pointer*/
iposrandsec = iposrand / 1000 ; ms -> sec
iposrand = iposrandsec / ifildur ; phase values (0-1)
krndpos linrand iposrand ; random offset in phase values
/*add random deviation to the time pointer*/
asamplepos1 = afilposphas + krndpos; resulting phase values (0-1)
asamplepos2 = asamplepos1
asamplepos3 = asamplepos1
asamplepos4 = asamplepos1

/*original key for each source waveform*/
kwavekey1 = 1
kwavekey2 = kwavekey1
kwavekey3 = kwavekey1
kwavekey4 = kwavekey1

/* maximum number of grains per k-period*/
imax_grains = 100

aL, aR partikkel igrainrate, idist, giDisttab, async, kenv2amt, ienv2tab, \
ienv_attack, ienv_decay, ksustain_amount, ka_d_ratio, igrainsize, kamp, igainmasks, \
kwavfreq, ksweepshape, iwavfreqstarttab, iwavfreqendtab, awavfm, \
ifmampstab, kfmenv, icosine, kTrainCps, knumpartial, \
kchroma, ichannelmasks, krandommask, kwaveform1, kwaveform2, kwaveform3, kwaveform4, \
iwaveampstab, asamplepos1, asamplepos2, asamplepos3, asamplepos4, \
kwavekey1, kwavekey2, kwavekey3, kwavekey4, imax_grains

outs aL, aR

endin

```

```

</CsInstruments>
<CsScore>
;il st dur speed grate gsize cent posrnd cntrnd pan dist
i1 0 2.757 1 200 15 0 0 0 0 0
s
i1 0 2.757 1 200 15 400 0 0 0 0
s
i1 0 2.757 1 15 450 400 0 0 0 0
s
i1 0 2.757 1 15 450 400 0 0 0 0.4
s
i1 0 2.757 1 200 15 0 400 0 0 1
s
i1 0 5.514 .5 200 20 0 0 600 .5 1
s
i1 0 11.028 .25 200 15 0 1000 400 1 1

</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

*fof, fof2, fog, grain, grain2, grain3, granule, sndwarp, sndwarpst, syncgrain, syncloop, partikkelget partikkelset partikkelsync*

## Crédits

Auteur : Thom Johansen  
Auteur : Torgeir Strand Henriksen  
Auteur : Øyvind Brandtsegg  
Avril 2007

Exemples écrits par Joachim Heintz et Øyvind Brandtsegg.

Nouveau dans la version 5.06

# partikkelget

*partikkelget* — Retourne un index de masque pour un paramètre de masque spécifique d'une instance courante de *partikkel*.

## Description

*partikkelget* est un opcode pour obtenir un index de masque de *partikkel* pour un paramètre spécifique. Utilisé de concert avec *partikkelset*, il permet de synchroniser le masquage de *partikkel* entre différentes instances courantes de l'opcode *partikkel*. On peut aussi l'utiliser pour contrôler d'autres processus basés sur un index de masque interne, par exemple pour créer des modèles de masquage plus complexes que ceux disponibles dans le système régulier de masquage de grain.

## Syntaxe

```
kindex partikkelget kparameterindex, iopcode_id
```

## Initialisation

*iopcode\_id* -- l'id de l'opcode, liant une instance de *partikkel* à une instance de *partikkelsync*.

## Exécution

*kindex* -- index de masque en sortie. Retourne l'index de masque courant pour le paramètre spécifié par *kparameterindex* dans l'instance de *partikkel* identifiée par *iopcode\_id*.

*kparameterindex* -- paramètre de masque. Sélection du paramètre de masquage pour lequel sera fourni l'index de masque courant. Les différents paramètres sont identifiés par :

- 0 : masque du gain
- 1 : masque du début de balayage de hauteur
- 2 : masque de la fin de balayage de hauteur
- 3 : masque de l'indice de modulation mf
- 4 : masque de canal
- 5 : masque de mélange de forme d'onde

## Exemple

Voici un exemple des opcodes *partikkelget* et *partikkelset*. Il utilise le fichier *partikkelgetset.csd* [examples/partikkelgetset.csd].

**Exemple 683. Exemple de manipulation de l'index de masque interne de *partikkel*, basé sur la valeur d'autres index de masque dans la même intance de *partikkel*.**

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out
-odac          ;;RT audio
; For Non-realtime ouput leave only the line below:
-o partikkel-getset.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 65536, 10, 1
giCosine ftgen 0, 0, 8192, 9, 1, 1, 90
giSigmoRise ftgen 0, 0, 8193, 19, 0.5, 1, 270, 1
giSigmoFall ftgen 0, 0, 8193, 19, 0.5, 1, 90, 1

instr 1

; freqstart masking table, simple ascending semitones pattern
iwavfreqstarttab ftgentmp 0, 0, 32, -2, 0, 11,
    semitone(0), semitone(1), semitone(2), semitone(3),
    semitone(4), semitone(5), semitone(6), semitone(7),
    semitone(8), semitone(9), semitone(10), semitone(11)

iwavfregendtab ftgentmp 0, 0, 32, -2, 0, 11,
    semitone(0), semitone(1), semitone(2), semitone(3),
    semitone(4), semitone(5), semitone(6), semitone(7),
    semitone(8), semitone(9), semitone(10), semitone(11)

; channel masking table, simple panning back and forth
ichannelmasks ftgentmp 0, 0, 32, -2, 0, 7, 0.0, 0.25, 0.5, 0.75, 1.0, 0.75, 0.5, 0.25

; init unused arate signals
awavfm = 0
asamplepos1 = 0
async = 0

; system
id = 1

a1,a2 partikkel 6, 0, -1, async, 0, -1, giSigmoRise, giSigmoFall,
    0.9, 0.5, 100, ampdbfs(-9), -1, 1, 0,iwavfreqstarttab, iwavfregendtab, awavfm,
    -1, -1, giCosine, 1, 1, 1, ichannelmasks, 0,
    giSine, giSine, giSine, giSine,
    -1, asamplepos1, asamplepos1, asamplepos1, asamplepos1,
    440, 440, 440, 440, 100, id

outch 1, a1, 2, a2

; using partikkelget and partikkelset to make the channelmask and freqmask indices
; interact to create a more complex pattern
kfqmask partikkelget 1, id
kchmask partikkelget 4, id
kcount1 init 0
kcount2 init 0
if kchmask > (kfqmask+kchmask)%11 then
partikkelset 1, kcount1, id
partikkelset 2, kcount1, id
kcount1 = (kcount1+1)%7
elseif kfqmask > (kfqmask*kchmask)%7 then
partikkelset 4, kcount2, id
kcount2 = (kcount2+kcount1)%6
endif

```



```
endin  
  
</CsInstruments>  
<CsScore>  
i1 0 30  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*partikkel*

## Crédits

Auteur : Thom Johansen  
Auteur : Øyvind Brandtsegg  
Mai 2017

Nouveau dans la version 6.09

# partikkelset

*partikkelset* — Fixe l'index de masque pour un paramètre de masque spécifique d'une instance courante de *partikkel*.

## Description

*partikkelset* est un opcode pour fixer l'index de masque de *partikkel* pour un paramètre spécifique. Utilisé de concert avec *partikkelget*, il permet de synchroniser le masquage de *partikkel* entre différentes instances courantes de l'opcode *partikkel*. On peut aussi l'utiliser pour contrôler d'autres processus basés sur un index de masque interne, par exemple pour créer des modèles de masquage plus complexes que ceux disponibles dans le système régulier de masquage de grain.

## Syntaxe

```
partikkelset kparameterindex, kmaskindex, iopcode_id
```

## Initialisation

*iopcode\_id* -- l'id de l'opcode, liant une instance de *partikkel* à une instance de *partikkelsync*.

## Exécution

*kparameterindex* -- paramètre de masque. Sélection du paramètre de masquage pour lequel sera fixé l'index de masque courant. Les différents paramètres sont identifiés par :

- 0 : masque du gain
- 1 : masque du début de balayage de hauteur
- 2 : masque de la fin de balayage de hauteur
- 3 : masque de l'indice de modulation mf
- 4 : masque de canal
- 5 : masque de mélange de forme d'onde

*kmaskindex* -- valeur sur laquelle mettre l'index de masque. Fixe l'index de masque courant pour le paramètre spécifié avec *kparameterindex* dans l'instance de *partikkel* identifié par *iopcode\_id*.

## Exemple

Voici un exemple des opcodes *partikkelget* et *partikkelset*. Il utilise le fichier *partikkelgetset.csd* [examples/partikkelgetset.csd].

**Exemple 684. Exemple de manipulation de l'index de masque interne de *partikkel*, basé sur la valeur d'autres index de masque dans la même instance de *partikkel*.**

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out
-odac          ;;RT audio
; For Non-realtime ouput leave only the line below:
-o partikkel-getset.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 65536, 10, 1
giCosine ftgen 0, 0, 8192, 9, 1, 1, 90
giSigmoRise ftgen 0, 0, 8193, 19, 0.5, 1, 270, 1
giSigmoFall ftgen 0, 0, 8193, 19, 0.5, 1, 90, 1

instr 1

; freqstart masking table, simple ascending semitones pattern
iwavfreqstarttab ftgentmp 0, 0, 32, -2, 0, 11,
    semitone(0), semitone(1), semitone(2), semitone(3),
    semitone(4), semitone(5), semitone(6), semitone(7),
    semitone(8), semitone(9), semitone(10), semitone(11)

iwavfregendtab ftgentmp 0, 0, 32, -2, 0, 11,
    semitone(0), semitone(1), semitone(2), semitone(3),
    semitone(4), semitone(5), semitone(6), semitone(7),
    semitone(8), semitone(9), semitone(10), semitone(11)

; channel masking table, simple panning back and forth
ichannelmasks ftgentmp 0, 0, 32, -2, 0, 7, 0.0, 0.25, 0.5, 0.75, 1.0, 0.75, 0.5, 0.25

; init unused arate signals
awavfm = 0
asamplepos1 = 0
async = 0

; system
id = 1

a1,a2 partikkel 6, 0, -1, async, 0, -1, giSigmoRise, giSigmoFall,
    0.9, 0.5, 100, ampdbfs(-9), -1, 1, 0,iwavfreqstarttab, iwavfregendtab, awavfm,
    -1, -1, giCosine, 1, 1, 1, ichannelmasks, 0,
    giSine, giSine, giSine, giSine,
    -1, asamplepos1, asamplepos1, asamplepos1, asamplepos1,
    440, 440, 440, 440, 100, id

outch 1, a1, 2, a2

; using partikkelget and partikkelset to make the channelmask and freqmask indices
; interact to create a more complex pattern
kfqmask partikkelget 1, id
kchmask partikkelget 4, id
kcount1 init 0
kcount2 init 0
if kchmask > (kfqmask+kchmask)%11 then
partikkelset 1, kcount1, id
partikkelset 2, kcount1, id
kcount1 = (kcount1+1)%7
elseif kfqmask > (kfqmask*kchmask)%7 then
partikkelset 4, kcount2, id
kcount2 = (kcount2+kcount1)%6
endif

```

```
endin  
  
</CsInstruments>  
<CsScore>  
i1 0 30  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*partikkel*

## Crédits

Auteur : Thom Johansen  
Auteur : Øyvind Brandtsegg  
Mai 2017

Nouveau dans la version 6.09

# partikkelsync

**partikkelsync** — Produit l'impulsion et la phase de l'horloge du distributeur de grain de *partikkel* pour synchroniser plusieurs instances de l'opcode *partikkel* à la même source d'horloge.

## Description

*partikkelsync* est un opcode dont la tâche est de produire l'impulsion et la phase de l'horloge du distributeur de grain de *partikkel*. On peut utiliser la sortie de *partikkelsync* pour synchroniser d'autres instances de l'opcode *partikkel* à la même horloge.

## Syntaxe

```
async [,aphase] partikkelsync iopcode_id
```

## Initialisation

*iopcode\_id* -- identificateur de l'opcode, liant une instance de *partikkel* à une instance de *partikkelsync*.

## Exécution

*async* -- signal d'impulsion de déclenchement. Envoie des impulsions de déclenchement synchronisées à l'horloge du distributeur de grain d'un opcode *partikkel*. Une impulsion de déclenchement est générée pour le démarrage de chaque grain dans l'opcode *partikkel* ayant le même *opcode\_id*. Dans une utilisation normale, on enverra ce signal à l'entrée *async* d'un autre opcode *partikkel* pour synchroniser plusieurs instances de *partikkel*.

*aphase* -- phase de l'horloge. Sort un signal de phase linéaire. On peut l'utiliser par exemple pour une synchronisation légère, ou simplement comme un générateur de phase à la *phasor*.

## Exemples

Voici un exemple de l'opcode *partikkelsync*. Il utilise le fichier *partikkelsync.csd* [examples/partikkel-sync.csd].

### Exemple 685. Exemple de l'opcode *partikkelsync*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out
-odac          ;;;RT audio
; For Non-realtime output leave only the line below:
; -o partikkel_softsync.wav -W ;;; for file output any platform
</CsOptions>

<CsInstruments>

sr = 44100
ksmps = 20
nchnls = 2
```

```

; Example by Oeyvind Brandtsegg 2007, revised 2008

giSine   ftgen 0, 0, 65537, 10, 1
giCosine ftgen 0, 0, 8193, 9, 1, 1, 90
giSigmoidRise ftgen 0, 0, 8193, 19, 0.5, 1, 270, 1 ; rising sigmoid
giSigmoidFall ftgen 0, 0, 8193, 19, 0.5, 1, 90, 1 ; falling sigmoid

; *****
; example of soft synchronization of two partikkel instances
; *****
instr 1

/*score parameters*/
igrainrate = p4 ; grain rate
igrainsize = p5 ; grain size in ms
igrainFreq = p6 ; fundamental frequency of source waveform
iosc2Dev = p7 ; partikkel instance 2 grain rate deviation factor
iMaxSync = p8 ; max soft sync amount (increasing to this value during length of note)

/*overall envelope*/
iattack = 0.001
idecay = 0.2
isustain = 0.7
irelease = 0.2
amp linsegr 0, iattack, 1, idecay, isustain, 1, isustain, irelease, 0

kgrainfreq = igrainrate ; grains per second
kdistribution = 0 ; periodic grain distribution
idisttab = -1 ; (default) flat distribution used
; for grain distribution

async = 0 ; no sync input
kenv2amt = 0 ; no secondary enveloping
ienv2tab = -1 ; default secondary envelope (flat)
ienv_attack = giSigmoidRise ; default attack envelope (flat)
ienv_decay = giSigmoidFall ; default decay envelope (flat)
ksustain_amount = 0.3 ; time (in fraction of grain dur) at
; sustain level for each grain
ka_d_ratio = 0.2 ; balance between attack and decay time
kduration = igrainsize ; set grain duration in ms
kamp = 0.2*0dbfs ; amp
igainmask = -1 ; (default) no gain masking
kwavfreq = igrainFreq ; fundamental frequency of source waveform
ksweepshape = 0 ; shape of frequency sweep (0=no sweep)
iwavfreqstarttab = -1 ; default frequency sweep start
; (value in table = 1, which give
; no frequency modification)
iwavfreqendtab = -1 ; default frequency sweep end
; (value in table = 1, which give
; no frequency modification)

awavfm = 0 ; no FM input
ifmampmtab = -1 ; default FM scaling (=1)
kfmenv = -1 ; default FM envelope (flat)
icosine = giCosine ; cosine ftable
kTrainCps = kgrainfreq ; set trainlet cps equal to grain
; rate for single-cycle trainlet in
; each grain
knumpartials = 3 ; number of partials in trainlet
kchroma = 1 ; balance of partials in trainlet
ichannelmask = -1 ; (default) no channel masking,
; all grains to output 1
krandommask = 0 ; no random grain masking
kwaveform1 = giSine ; source waveforms
kwaveform2 = giSine ;
kwaveform3 = giSine ;
kwaveform4 = giSine ;
iwaveamptab = -1 ; mix of 4 source waveforms and

```

```

; trainlets (set to default)
asamplepos1 = 0 ; phase offset for reading source waveform
asamplepos2 = 0 ;
asamplepos3 = 0 ;
asamplepos4 = 0 ;
kwavekey1 = 1 ; original key for source waveform
kwavekey2 = 1 ;
kwavekey3 = 1 ;
kwavekey4 = 1 ;
imax_grains = 100 ; max grains per k period
iopcode_id = 1 ; id of opcode, linking partikkel
; to partikkelsync

a1 partikkel kgrainfreq, kdistribution, idisttab, async, kenv2amt, \
    ienv2tab, ienv_attack, ienv_decay, ksustain_amount, ka_d_ratio, \
    kduration, kamp, igainmasks, kwavfreq, ksweepshape, \
    iwavfreqstarttab, iwavfreqendtab, awavfm, ifmamptab, kfmenv, \
    icosine, kTrainCps, knumpartials, kchroma, ichannelmasks, \
    krandommask, kwaveform1, kwaveform2, kwaveform3, kwaveform4, \
    iwaveamptab, asamplepos1, asamplepos2, asamplepos3, asamplepos4, \
    kwavekey1, kwavekey2, kwavekey3, kwavekey4, imax_grains, iopcode_id

async1 partikkelsync iopcode_id ; clock pulse output of the
; partikkel instance above
ksyncGravity line 0, p3, iMaxSync ; strength of synchronization
aphase2 init 0
asyncPolarity limit (int(aphase2*2)*2)-1, -1, 1
; use the phase of partikkelsync instance 2 to find sync
; polarity for partikkel instance 2.
; If the phase of instance 2 is less than 0.5, we want to
; nudge it down when synchronizing,
; and if the phase is > 0.5 we want to nudge it upwards.
async1 = async1*ksyncGravity*asyncPolarity ; prepare sync signal
; with polarity and strength

kgrainfreq2 = igrainrate * iosc2Dev ; grains per second for second partikkel instance
iopcode_id2 = 2
a2 partikkel kgrainfreq2, kdistribution, idisttab, async1, kenv2amt, \
    ienv2tab, ienv_attack, ienv_decay, ksustain_amount, ka_d_ratio, \
    kduration, kamp, igainmasks, kwavfreq, ksweepshape, \
    iwavfreqstarttab, iwavfreqendtab, awavfm, ifmamptab, kfmenv, \
    icosine, kTrainCps, knumpartials, kchroma, ichannelmasks, \
    krandommask, kwaveform1, kwaveform2, kwaveform3, kwaveform4, \
    iwaveamptab, asamplepos1, asamplepos2, asamplepos3, \
    asamplepos4, kwavekey1, kwavekey2, kwavekey3, kwavekey4, \
    imax_grains, iopcode_id2

async2, aphase2 partikkelsync iopcode_id2
; clock pulse and phase
; output of the partikkel instance above,
; we will only use the phase

outs a1*amp, a2*amp

endin

</CsInstruments>
<CsScore>

/*score parameters
igrainrate = p4 ; grain rate
igrainsize = p5 ; grain size in ms
igrainFreq = p6 ; frequency of source wave within grain
iosc2Dev = p7 ; partikkel instance 2 grain rate deviation factor
iMaxSync = p8 ; max soft sync amount (increasing to this value during length of note)
*/
; GrRate GrSize GrFund Osc2Dev MaxSync

```

```
i1 0 10 2 20 880 1.3 0.3
s
i1 0 10 5 20 440 0.8 0.3
s
i1 0 6 55 15 660 1.8 0.45
s
i1 0 6 110 10 440 0.6 0.6
s
i1 0 6 220 3 660 2.6 0.45
s
i1 0 6 220 3 660 2.1 0.45
s
i1 0 6 440 3 660 0.8 0.22
s

e

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*partikkel*

## Crédits

Auteur : Thom Johansen  
Auteur : Torgeir Strand Henriksen  
Auteur : Øyvind Brandtsegg  
Avril 2007

Nouveau dans la version 5.06



# passign

`passign` — Affecte un ensemble de p-champs à des variables de taux *i*.

## Description

Affecte un ensemble de p-champs à des variables de taux *i*, ou à des tableaux de taux *i* ou *k*.

## Syntaxe

```
ivar1, ... passign [istart][, iend  
iarray passign [istart][, iend  
karray passign [istart][, iend
```

## Initialisation

L'argument optionnel *istart* donne l'indice du premier p-champ à affecter. La valeur par défaut est 1, ce qui correspond au numéro d'instrument.

L'argument facultatif *iend* donne l'indice du dernier p-champ à affecter. La valeur par défaut est 0, ce qui correspond à tous.

Une des variables peut être une variable chaîne de caractères, à laquelle sera affecté dans ce cas le seul paramètre de type chaîne de caractères, s'il y en a un, sinon une erreur.

## Exécution

*passign* transfère les p-champs de l'instrument à des variables de l'instrument, en commençant par celui qui est identifié par l'argument *istart*. Il ne doit pas y avoir plus de variables que de p-champs, mais il peut y en avoir moins.

*passign* peut transférer les p-champs de l'instrument dans un tableau unidimensionnel de taux-*i* ou de taux-*k*.

## Voir aussi

*assign*, *pcount*,

## Exemples

### Exemple 686. Une variante de `toot8.csd` qui utilise `passign`.

Voici un exemple de l'opcode `passign`. Il utilise le fichier *passign.csd* [examples/passign.csd].

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac      ;;realtime audio out  
;-iadc      ;;uncomment -iadc if real audio input is needed too  
; For Non-realtime ouput leave only the line below:
```

```

; -o passign.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 8

idur,iamp,iskiptime,iattack,irelease,irvbtime,irvbgain  passign  3

kamp      linen      iamp, iattack, idur, irelease
asig      soundin     "fox.wav", iskiptime
arampsig  =           kamp * asig
aeffect   reverb      asig, irvbtime
arvbretrn =           aeffect * irvbgain
;mix dry & wet signals
outs      arampsig + arvbretrn, arampsig + arvbretrn

        endin

</CsInstruments>
<CsScore>

;ins strt dur  amp  skip atk  rel          rvbt rvbgain
i8  0   4   .3   0   .03 .1          1.5 .3
i8  4   4   .3   1.6 .1  .1          3.1 .7
i8  8   4   .3   0   .5  .1          2.1 .2
i8 12   4   .4   0   .01 .1          1.1 .1
i8 16   4   .5   0.1 .01 .1          0.1 .1

e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur: John ffitch  
 Université de Bath, Codemist Ltd.  
 Bath, UK  
 Décembre 2009

Nouveau dans la version 5.12 ; point final et version avec tableau après la 5.12

# paulstretch

paulstretch — Algorithme d'étirement temporel extrême par Nasca Octavian Paul.

## Description

L'opcode *paulstretch* est une mise en oeuvre légère de l'algorithme d'étirement temporel PaulStretch par Nasca Octavian Paul. Il est idéal pour allonger la durée d'un signal dans de grandes proportions.

L'algorithme Paulstretch fonctionne de manière très semblable aux autres méthodes d'étirement temporel basées sur la TFCT, en utilisant le chevauchement et l'ajout avec une fenêtre de Hanning. L'utilisation du mélange de phase et de tailles de fenêtres très grandes (spécifiées en secondes) sont propres à paulstretch.

## Syntaxe

```
asig paulstretch istretch, iwindowsize, ift
```

## Initialisation

*istretch* -- Facteur d'étirement.

*iwindowsize* -- Taille de la fenêtre, en secondes.

*ift* -- table de fonction du signal source. Les tables à allocation différée (voir *GEN01*) sont acceptées, mais l'opcode attend une source mono.

## Exemples

Voici un exemple de l'opcode paulstretch. Il utilise le fichier *paulstretch.csd* [examples/paulstretch.csd]

### Exemple 687. Exemple de l'opcode paulstretch.

```
<CsoundSynthesizer>
<CsOptions>
-o paulstretch.wav -W
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 100
0dbfs = 1
nchnls = 2

giwav ftgen 0, 0, 0, 1, "fox.wav", 0, 0, 1

instr 1
aout paulstretch 10, 0.2, giwav
outs aout, aout
endin

</CsInstruments>
<CsScore>
i1 0 30
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : Paul Batchelor  
Mai 2016

# pcauchy

pcauchy — Générateur de nombres aléatoires de distribution de Cauchy (valeurs positives seulement).

## Description

Générateur de nombres aléatoires de distribution de Cauchy (valeurs positives seulement). C'est un générateur de bruit de classe x.

## Syntaxe

```
ares pcauchy kalpha
```

```
ires pcauchy kalpha
```

```
kres pcauchy kalpha
```

## Exécution

*pcauchy kalpha* -- contrôle l'étalement à partir de zéro (grand *kalpha* = grand étalement). Ne produit que des nombres positifs.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode *pcauchy*. Il utilise le fichier *pcauchy.csd* [examples/pcauchy.csd].

### Exemple 688. Exemple de l'opcode *pcauchy*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if real audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pcauchy.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
odbfs = 1

instr 1    ; every run time same values
```

```

kalpha pcauchy 1000
printk .2, kalpha ; look
aout oscili 0.8, 440+kalpha, 1 ; & listen
outs aout, aout
endin

instr 2 ; every run time different values

seed 0
kalpha pcauchy 1000
printk .2, kalpha ; look
aout oscili 0.8, 440+kalpha, 1 ; & listen
outs aout, aout
endin
</CsInstruments>
<CsScore>
; sine wave
f 1 0 16384 10 1

i 1 0 2
i 2 3 2
e

</CsScore>
</CsoundSynthesizer>

```

La sortie contiendra des lignes comme celles-ci :

```

i 1 time 0.00033: 10.48851
i 1 time 0.20033: 0.29508
i 1 time 0.40033: 1.75214
i 1 time 0.60033: 22.88281
i 1 time 0.80033: 16.06435
i 1 time 1.00000: 0.43110
i 1 time 1.20033: 16.51694
i 1 time 1.40033: 2.98797
i 1 time 1.60033: 1.32767
i 1 time 1.80000: 17.94039
i 1 time 2.00000: 4.85994
Seeding from current time 1526147515
i 2 time 3.00033: 0.89797
i 2 time 3.20033: 9.19447
i 2 time 3.40033: 0.73053
i 2 time 3.60000: 7.43371
i 2 time 3.80033: 5.87640
i 2 time 4.00000: 0.80456
i 2 time 4.20000: 4.50637
i 2 time 4.40033: 2.08145
i 2 time 4.60033: 13.47125
i 2 time 4.80033: 3.16399
i 2 time 5.00000: 11.05428

```

## Voir aussi

*seed, betarand, bexprnd, cauchy, exprand, gauss, linrand, poisson, trirand, unirand, weibull*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

# pchbend

pchbend — Donne la valeur actuelle du pitch-bend pour ce canal.

## Description

Donne la valeur actuelle du pitch-bend pour ce canal.

## Syntaxe

```
ibend pchbend [imin] [, imax]
```

```
kbend pchbend [imin] [, imax]
```

## Initialisation

*imin*, *imax* (optionel) -- fixe les limites minimale et maximale pour les valeurs obtenues

## Exécution

Donne la valeur actuelle du pitch-bend pour ce canal. Noter que l'on a la valeur du pitch-bend qui est indépendant du pitch MIDI, ce qui permet d'utiliser cette valeur pour n'importe quel but.

## Exemples

Voici un exemple de l'opcode pchbend. Il utilise le fichier *pchbend.csd* [exemples/pchbend.csd].

### Exemple 689. Exemple de l'opcode pchbend.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -Q1 -Ma ;;realtime audio out and midi in (on all inputs) and out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pchbend.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;use external midi keyboard

icps cpsmidi
kbnd pchbend 0, 100      ;one octave lower and higher
kenv linsegr 0,.001, 1, .1, 0      ;amplitude envelope
asig pluck .8 * kenv, icps+kbnd, 440, 0, 1
outs asig, asig
```

```
    endin
  </CsInstruments>
<CsScore>

f 0 30 ;runs 30 seconds

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*aftouch, ampmidi, cpsmidi, cpsmidib, midictrl, notnum, octmidi, octmidib, pchmidi, pchmidib, veloc*

## Crédits

Auteur : Barry L. Vercoe - Mike Berry  
MIT - Mills  
Mai 1997



# pchmidi

**pchmidi** — Retourne le numéro de note de l'évènement MIDI courant, exprimé en unités d'octave-point-classe de hauteur.

## Description

Retourne le numéro de note de l'évènement MIDI courant, exprimé en unités d'octave-point-classe de hauteur.

## Syntaxe

`ipch pchmidi`

## Exécution

Retourne le numéro de note de l'évènement MIDI courant, exprimé en unités d'octave-point-classe de hauteur, pour traitement local.



### pchmidi vs. pchmidinn

L'opcode *pchmidi* ne produit des résultats significatifs qu'avec une note activée par le MIDI (soit en , soit depuis une partition MIDI avec l'option -F). Avec *pchmidi*, la valeur du numéro de note MIDI provient de l'évènement MIDI qui est associé en interne avec l'instance de l'instrument. Au contraire, l'opcode *pchmidinn* peut être utilisé dans n'importe quelle instance d'instrument de Csound, que celle-ci soit activée par un évènement MIDI, un évènement de partition, un évènement en ligne ou depuis un autre instrument. La valeur d'entrée de *pchmidinn* peut provenir par exemple d'un p-champ dans une partition textuelle ou bien elle peut avoir été extraite au moyen de l'opcode *notnum* de l'évènement MIDI en qui a activé la note courante.

## Exemples

Voici un exemple de l'opcode *pchmidi*. Il utilise le fichier *pchmidi.csd* [examples/pchmidi.csd].

### Exemple 690. Exemple de l'opcode pchmidi.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac        -iadc      -d          -M0    ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o pchmidi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
```

```
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; This example expects MIDI note inputs on channel 1
i1 pchmidi

print i1
endin

</CsInstruments>
<CsScore>

;Dummy f-table to give time for real-time MIDI events
f 0 8000
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*aftouch, ampmidi, cpsmidi, cpsmidib, midictrl, notnum, octmidi, octmidib, pchbend, pchmidib, veloc, cpsmidinn, octmidinn, pchmidinn*

## Crédits

Auteur : Barry L. Vercoe - Mike Berry  
MIT - Mills  
Mai 1997

Exemple écrit par Kevin Conder.

# pchmidib

pchmidib — Retourne le numéro de note de l'évènement MIDI courant en le modifiant par la valeur courante de pitch-bend, exprimé en unités octave-point-classe de hauteur.

## Description

Retourne le numéro de note de l'évènement MIDI courant en le modifiant par la valeur courante de pitch-bend, exprimé en unités octave-point-classe de hauteur.

## Syntaxe

```
ipch pchmidib [irange]
```

```
kpch pchmidib [irange]
```

## Initialisation

*irange* (facultatif) -- l'étendue du pitch-bend en demi-tons.

## Exécution

Retourne le numéro de note de l'évènement MIDI courant en le modifiant par la valeur courante de pitch-bend et exprime le résultat en unités octave-point-classe de hauteur. Disponible comme une valeur d'initialisation ou comme une valeur continue de taux-k.

## Exemples

Voici un exemple de l'opcode pchmidib. Il utilise le fichier *pchmidib.csd* [examples/pchmidib.csd].

### Exemple 691. Exemple de l'opcode pchmidib.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac          -iadc      -d          -M0   ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o pchmidib.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; This example expects MIDI note inputs on channel 1
```

```
i1 pchmidib

print i1
endin

</CsInstruments>
<CsScore>

;Dummy f-table to give time for real-time MIDI events
f 0 8000
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*aftouch, ampmidi, cpsmidi, cpsmidib, midictrl, notnum, octmidi, octmidib, pchbend, pchmidi, veloc*

## Crédits

Auteur : Barry L. Vercoe - Mike Berry  
MIT - Mills  
Mai 1997

Exemple écrit par Kevin Conder.

# pchmidinn

pchmidinn — Convertit un numéro de note Midi en unités d'octave point classe de hauteur.

## Description

Convertit un numéro de note Midi en unités d'octave point classe de hauteur.

## Syntaxe

**pchmidinn** (MidiNoteNumber) (arguments de taux-i ou -k seulement)

où l'argument entre parenthèses peut être une expression.

## Exécution

*pchmidinn* est une fonction qui prend une valeur de taux-i ou de taux-k représentant un numéro de note Midi et qui retourne la valeur de hauteur équivalente dans le format octave point classe de hauteur. Cette conversion suppose que le do médian (8.00 en *pch*) est la note Midi numéro 60. Les numéros de note Midi sont par définition des nombres entiers compris entre 0 et 127 mais des valeurs fractionnaires ou des valeurs en dehors de cet intervalle seront correctement interprétées.



### pchmidinn vs. pchmidi

L'opcode *pchmidinn* peut être utilisé dans n'importe quelle instance d'instrument de Csound, que celle-ci soit activée depuis un évènement Midi, un évènement de partition, un évènement en ligne, ou depuis un autre instrument. La valeur d'entrée de *pchmidinn* peut provenir par exemple d'un p-champ dans une partition textuelle ou bien avoir été retrouvée au moyen de l'opcode *notnum* à partir de l'évènement Midi en qui a activé la note courante. Le numéro de note Midi à convertir doit être spécifié comme une expression de taux-i ou de taux-k. D'un autre côté, l'opcode *pchmidi* ne fournit des résultats significatifs qu'avec une note activée par le Midi (soit en temps réel soit à partir d'une partition Midi avec l'option -F). Avec *pchmidi*, la valeur du numéro de note Midi provient de l'évènement Midi associé à l'instance d'instrument, et aucune source ni aucune expression ne peuvent être spécifiées pour cette valeur.

*pchmidinn* et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

**Tableau 19. Valeurs de Hauteur et de Fréquence**

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme

deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par contraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.



## Note

La conversion de *pch*, *oct*, ou *midinn* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

## Exemples

Voici un exemple de l'opcode *pchmidinn*. Il utilise le fichier *cpsmidinn.csd* [examples/cpsmidinn.csd].

### Exemple 692. Exemple de l'opcode *pchmidinn*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
; Prints a table showing the equivalents of all Midi
; note numbers from 0-127 in cycles-per-second,
; octave.decimal, and octave.pitchclass units.

<CsOptions>
; Select audio/midi flags here according to platform.
; This example produces no audio, so we render in
; non-realtime and turn off sound to disk:
-n
</CsOptions>
<CsInstruments>

instr 1
; i-time loop to print conversion table
imidiNN = 0
loop1:
  icps = cpsmidinn(imidiNN)
  ioct = octmidinn(imidiNN)
  ipch = pchmidinn(imidiNN)
```

```

    print    imidiNN, icps, ioct, ipch

    imidiNN = imidiNN + 1
    if (imidiNN < 128) igoto loop1
endin

instr 2
; test k-rate converters
kMiddleC = 60
kcps = cpsmidinn(kMiddleC)
koct = octmidinn(kMiddleC)
kpch = pchmidinn(kMiddleC)

printks "%d %f %f %f\n", 1.0, kMiddleC, kcps, koct, kpch
endin

</CsInstruments>
<CsScore>
i1 0 0
i2 0 0.1
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*cpsmidinn, octmidinn, pchmidi, notnum, cpspch, cpsoct, octcps, octpch, pchoct*

## Crédits

Dérivé à partir des convertisseurs de valeur originaux de Barry Vercoe.

Nouveau dans la version 5.07

# pchoct

**pchoct** — Convertit une valeur octave-point-partie-décimale en classe de hauteur.

## Description

Convertit une valeur octave-point-partie-décimale en classe de hauteur.

## Syntaxe

**pchoct** (oct) (arguments de taux-i ou -k seulement)

où l'argument entre parenthèses peut être une expression.

## Exécution

*pchoct* et ses opcodes associés sont réellement des *convertisseurs de valeur* spécialisés dans la manipulation des données de hauteur.

Les données concernant la hauteur et la fréquence peuvent exister dans un des formats suivants :

**Tableau 20. Valeurs de Hauteur et de Fréquence**

Nom	Abréviation
octave point classe de hauteur (8ve.pc)	pch
octave point partie décimale	oct
cycles par seconde	cps
Numéro de note Midi (0-127)	midinn

Les deux premières formes sont constituées d'un nombre entier, représentant le registre d'octave, suivi d'une partie décimale dont la signification est particulière. Pour *pch*, la partie fractionnaire est lue comme deux chiffres décimaux représentant les douze classes de hauteur du tempérament égal de .00 pour do jusqu'à .11 pour si. Pour *oct*, la partie fractionnaire est interprétée comme une véritable partie fractionnaire décimale d'une octave. Les deux formes fractionnaires sont ainsi dans un rapport de 100/12. Dans les deux formes, la fraction est précédée par un nombre entier indice de l'octave, tel que 8.00 représente le do médian, 9.00 le do au-dessus, etc. Les numéros de note Midi sont compris entre 0 et 127 (inclus), avec 60 représentant le do médian, et sont habituellement des nombres entiers. Ainsi, on peut représenter le la 440 alternativement par 440 (*cps*), 69 (*midinn*), 8.09 (*pch*), ou 8.75 (*oct*). On peut encoder des divisions microtonales du demi-ton *pch* en utilisant plus de deux positions décimales.

Les noms mnémotechniques des unités de conversion de hauteur sont dérivés des morphèmes des formes concernées, le second morphème décrivant la source et le premier morphème l'objet (le résultat). Ainsi *cpspch*(8.09) convertira l'argument de hauteur 8.09 en son équivalent en *cps* (ou Hertz), ce qui donne la valeur 440. Comme l'argument est constant pendant toute la durée de la note, cette conversion aura lieu pendant l'initialisation, avant qu'aucun échantillon de la note actuelle ne soit produit.

Par constraste, la conversion *cpsoct*(8.75 + k1) donne la valeur du la 440 transposée par l'intervalle octaviant *k1*. Le calcul sera répété à chaque k-période car c'est le taux de variation de *k1*.





## Note

La conversion de *pch*, *oct*, ou *midim* vers *cps* n'est pas une opération linéaire mais elle implique un calcul d'exponentielle qui peut coûter cher en temps de traitement s'il est exécuté de manière répétitive. Csound utilise dorénavant une consultation de table interne pour faire cela efficacement, même aux taux audio. Comme l'indice dans la table est tronqué sans interpolation, la résolution en hauteur avec un de ces opcodes est limitée à 8192 divisions discrètes et égales de l'octave, et quelques degrés de l'échelle tempérée égale de 12 demi-tons sont très légèrement désaccordés (d'au plus 0,15 cent).

## Exemples

Voici un exemple de l'opcode *pchoct*. Il utilise le fichier *pchoct.csd* [examples/pchoct.csd].

### Exemple 693. Exemple de l'opcode *pchoct*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pchoct.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Convert an octave-point-decimal value into a
; pitch-class value.
ioct = 8.75
ipch = pchoct(ioct)

print ipch
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1: ipch = 8.090
```

## Voir aussi

*cpsoct, cpspch, octcps, octpch, cpsmidinn, octmidinn, pchmidinn*

## Crédits

Exemple écrit par Kevin Conder.

# pchtom

pchtom — Conversion d'un pch en numéro de note MIDI.

## Description

Opcode du greffon emugens.

Convertit un pch en numéro de note MIDI. La représentation **pch** est de la forme octave.classe\_de\_hauteur où classe\_de\_hauteur est un nombre entre 00 et 12.

pch	midi	nom de note
-----	-----	-----
8.09	69	4A
8.00	60	4C

## Syntaxe

imidi **pchtom** ipch

kmidi **pchtom** kpch

## Exécution

*kpch* / *ipch* -- Hauteur représentée en octave.classe\_de\_hauteur.

*kmidi* / *imidi* -- Numéro de note MIDI.



### Note

Utiliser *pchmidinn* pour faire l'opération inverse.

## Exemples

Voici un exemple de l'opcode pchtom. Il utilise le fichier *pchtom.csd* [examples/pchtom.csd].

### Exemple 694. Exemple de l'opcode pchtom.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2
0dbfs = 1.0

; Show some conversions, both at i- and at k-time

instr 1
  imidi = pchtom(8.09)
```

```
print imidi

kidx init 0

kpch = 8 + kidx / 100
kmidi = pchtom(kpch)
kidx += 1
printf "kpch: %f    kmidi: %f\n", kidx+1, kpch, kmidi

if kidx >= 12 then
    turnoff
endif

endin

</CsInstruments>
<CsScore>
i 1 0 1

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pchoct, pchmidinn, cpsmidinn, ftom, mton*

## Crédits

Par : Eduardo Moguillansky 2017

# pconvolve

convolve — Convolution basée sur un algorithme overlap-save à découpage uniforme.

## Description

Convolution basée sur un algorithme overlap-save à découpage uniforme. Comparé à l'opcode *convolve*, *pconvolve* a trois atouts :

- petit délai
- peut fonctionner en temps réel pour les fichier de réponse impulsionnelle les plus courts
- pas de passe d'analyse avant le traitement
- restitution souvent plus rapide que *convolve*

## Syntaxe

```
ar1 [, ar2] [, ar3] [, ar4] pconvolve ain, ifilcod [, ipartitionsizes, ichannel]
```

## Initialisation

*ifilcod* -- entier ou chaîne de caractères définissant un fichier de réponse impulsionnelle. Les fichiers multi-canaux sont supportés. Le fichier doit avoir le même taux d'échantillonnage que l'orchestre. [Note : on ne peut pas utiliser les fichiers de *cvanal* !] Il faut garder à l'esprit que les fichiers plus longs nécessitent plus de temps de calcul [et probablement une plus grande taille des fragments et plus de latence]. Avec les processeurs actuels, les fichiers dépassant quelques secondes pourront ne pas être restitués en temps réel.

*ipartitionsizes* (facultatif, par défaut égal à la taille du tampon de sortie [-b]) -- la taille en échantillons de chaque morceau de la réponse impulsionnelle. C'est le paramètre qu'il faut ajuster pour avoir les meilleures performances en fonction de la taille du fichier de réponse impulsionnelle. En général, une petite taille signifie une latence moins importante mais plus de temps de calcul. Si l'on spécifie une valeur qui n'est pas une puissance de 2 l'opcode trouvera la plus petite puissance de 2 immédiatement supérieure et l'utilisera comme taille des fragments.

*ichannel* (facultatif) -- le canal de la réponse impulsionnelle à utiliser.

## Exécution

*ain* -- signal audio en entrée.

La latence totale de l'opcode peut être calculée comme ceci [*ipartitionsizes* étant une puissance de 2]

```
latency = (ksmps < ipartitionsizes ? ipartitionsizes + ksmps : ipartitionsizes)/sr
```

## Exemples

L'instrument 1 montre un exemple de convolution en temps réel.

L'instrument 2 montre comment faire une convolution basée sur un fichier avec une méthode de "prospec-tion" pour supprimer tout délai.



## NOTE

Vous pouvez télécharger les fichiers de réponse impulsionnelle depuis [noisevault.com](http://noisevault.com) ou bien remplacer les noms de fichier avec vos propres fichiers de réponse impulsionnelle.

Voici un exemple de l'opcode `pconvolve`. Il utilise le fichier `pconvolve.csd` [exemples/pconvolve.csd].

### Exemple 695. Exemple de l'opcode `pconvolve`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
-iadc   ;;uncomment -iadc if real audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pconvolve.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kmix = .5 ; Wet/dry mix. Vary as desired.
kvol = .05*kmix ; Overall volume level of reverb. May need to adjust
           ; when wet/dry mix is changed, to avoid clipping.

; do some safety checking to make sure we the parameters a good
kmix = (kmix < 0 || kmix > 1 ? .5 : kmix)
kvol = (kvol < 0 ? 0 : .5*kvol*kmix)

; size of each convolution partion -- for best performance, this parameter needs to be tweaked
ipartitionsize = p4

; calculate latency of pconvolve opcode
idel = (ksmps < ipartitionsize ? ipartitionsize + ksmps : ipartitionsize)/sr
prints "Convolving with a latency of %f seconds%n", idel

; actual processing
al, ar ins ;get live input
awetl, awetr pconvolve kvol*(al+ar), "kickroll.wav", ipartitionsize
; Delay dry signal, to align it with the convoled sig
adryl delay (1-kmix)*al, idel
adryr delay (1-kmix)*ar, idel
outs adryl+awetl, adryr+awetr

endin

instr 2

imix = 0.5 ; Wet/dry mix. Vary as desired.
ivol = .05*imix ; Overall volume level of reverb. May need to adjust
           ; when wet/dry mix is changed, to avoid clipping.
ipartitionsize = 1024 ; size of each convolution partion
idel = (ksmps < ipartitionsize ? ipartitionsize + ksmps : ipartitionsize)/sr ; latency of pconvolve c

kcount init idel*kr
; since we are using a soundin [instead of ins] we can
```

```

; do a kind of "look ahead" by looping during one k-pass
; without output, creating zero-latency
loop:
    asig soundin p4, 0
    awetl, awetr pconvolve ivol*(asig), "rv_stereo.wav", ipartitionsiz
    adry delay (1-imix)*asig, idel ; Delay dry signal, to align it with
    kcount = kcount - 1
    if kcount > 0 kgoto loop
    outs awetl+adry, awetr+adry

endin
</CsInstruments>
<CsScore>

i 1 0 20 1024 ;play live for 20 seconds

i 2 20 5 "fox.wav"
i 2 25 5 "flute.aiff"

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*convolve, dconv.*

## Crédits

Auteur : Matt Ingalls  
2004

# pcount

`pcount` — Retourne le nombre de p-champs appartenant à un évènement de note.

## Description

`pcount` retourne le nombre de p-champs appartenant à un évènement de note.

## Syntaxe

`icount pcount`

## Initialisation

`icount --` reçoit le nombre de p-champs de l'évènement de note courant.



### Note

Noter que le nombre de p-champs rapporté n'est pas nécessairement celui qui est explicitement écrit dans la partition, mais les p-champs présentés à l'instrument au travers de mécanismes comme le *report de p-champ*.

## Exemples

Voici un exemple de l'opcode `pcount`. Il utilise le fichier `pcount.csd` [examples/pcount.csd].

### Exemple 696. Exemple de l'opcode `pcoun`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pcount.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

; This UDO returns a pfield value but returns 0 if it does not exist.
opcode mypvalue, i, i
index xin
inum pcount
if (index > inum) then
iout = 0.0
else
iout pindex index
endif
```



```

xout iout
endop

; Envelope UDO that reads parameters from a flexible number of pfields
; Syntax: kenv flexlinseg ipstart
;         ipstart is the first pfield of the envelope
;         parameters. Reads remaining pfields (up to 21 of them).
;         kenv is the output envelope.

opcode flexlinseg, k, i
  ipstart xin

  iep1 mypvalue ipstart
  iep2 mypvalue ipstart + 1
  iep3 mypvalue ipstart + 2
  iep4 mypvalue ipstart + 3
  iep5 mypvalue ipstart + 4
  iep6 mypvalue ipstart + 5
  iep7 mypvalue ipstart + 6
  iep8 mypvalue ipstart + 7
  iep9 mypvalue ipstart + 8
  iepa mypvalue ipstart + 9
  iepb mypvalue ipstart + 10
  iepc mypvalue ipstart + 11
  iepd mypvalue ipstart + 12
  iepe mypvalue ipstart + 13
  iepf mypvalue ipstart + 14
  iepg mypvalue ipstart + 15
  ieph mypvalue ipstart + 16
  iepi mypvalue ipstart + 17
  iepj mypvalue ipstart + 18
  iepk mypvalue ipstart + 19
  iepl mypvalue ipstart + 20

  kenv linseg iep1, iep2, iep3, iep4, iep5, iep6, iep7, iep8, \
           iep9, iepa, iepb, iepc, iepd, iepe, iepf, iepg, \
           ieph, iepi, iepj, iepk, iepl

  xout kenv
endop

instr 1
; This instrument only requires 3 pfields but can accept up to 24.
; (You will still get warnings about more than 3).
kenv flexlinseg 4 ; envelope params start at p4
aout oscili kenv*.5, 256, 1
outs aout, aout

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1

i1 0 2 0.0 1.0 1.0 1.0 0.0
i1 2 2 0.0 0.1 1.0 1.0 1.0 0.1 0.0
i1 4 2 0.0 0.5 0.0 ; one problem is that "missing" pfields carry
i1 6 2 0.0 0.5 0.0 ! ; now we can fix this problem with !
i1 8 10 0.0 3.0 1.0 0.3 0.1 0.3 1.0 0.3 0.1 0.3 1.0 0.3 0.1 0.8 0.9 5.0 0.0

e
</CsScore>
</CsoundSynthesizer>

```

L'exemple produit la sortie suivante :

WARNING: instr 1 uses 3 p-fields but is given 8

```
B 0.000 .. 2.000 T 2.000 TT 2.000 M: 0.49966 0.49966
WARNING: instr 1 uses 3 p-fields but is given 10
B 2.000 .. 4.000 T 4.000 TT 4.000 M: 0.50000 0.50000
WARNING: instr 1 uses 3 p-fields but is given 10
B 4.000 .. 6.000 T 6.000 TT 6.000 M: 0.49943 0.49943
WARNING: instr 1 uses 3 p-fields but is given 6
B 6.000 .. 8.000 T 8.000 TT 8.000 M: 0.00000 0.00000
WARNING: instr 1 uses 3 p-fields but is given 20
B 8.000 .. 18.000 T 18.000 TT 18.000 M: 0.49994 0.49994
```

Il y a des avertissements parce que certains p-champs ne sont pas utilisés explicitement par l'instrument

## Voir aussi

*pindex*

## Crédits

Exemple par : Anthony Kozar

Décembre 2006

# pdclip

pdclip — Réalise un écrêtage linéaire sur un signal audio ou un phaseur.

## Description

L'opcode *pdclip* permet d'écarter un pourcentage de l'intervalle d'entrée et de l'amplifier à pleine échelle. C'est la même chose que de multiplier le signal et de limiter l'ambitus du résultat, mais *pdclip* permet de décider quelle partie de l'ambitus du signal doit être distordue plutôt que de donner le facteur d'échelle, et il a un paramètre de décalage pour un écrêtage asymétrique de l'intervalle d'amplitude du signal. *pdclip* est aussi utile pour redistribuer les valeurs de phaseurs pour la synthèse par distorsion de phase.

## Syntaxe

```
aout pdclip ain, kWidth, kCenter [, ibipolar [, ifullscale]]
```

## Initialisation

*ibipolar* -- un paramètre facultatif spécifiant le mode unipolaire (0) ou bipolaire (1). Par défaut, mode unipolaire.

*ifullscale* -- paramètre facultatif spécifiant l'intervalle des valeurs d'entrée et de sortie. Le maximum sera *ifullscale*. Le minimum dépend du mode d'opération : zéro pour le mode unipolaire ou *-ifullscale* pour le mode bipolaire. Vaut 1.0 par défaut. Il faut donner à ce paramètre la valeur maximale attendue en entrée.

## Exécution

*ain* -- le signal d'entrée à écrêter.

*aout* -- le signal de sortie.

*kWidth* -- le pourcentage de l'ambitus du signal qui est écrêté (compris entre 0 et 1).

*kCenter* -- un décalage pour déplacer la fenêtre du signal non écrêté vers le haut ou vers le bas dans l'intervalle (essentiellement une composante continue). Les valeurs doivent être dans l'intervalle [-1, 1], la valeur zéro représentant l'absence de décalage (que le mode utilisé soit unipolaire ou bipolaire).

L'opcode *pdclip* effectue un écrêtage linéaire sur le signal *ain*. *kWidth* spécifie le pourcentage de l'amplitude du signal qui est écrêtée. Le reste de l'intervalle d'entrée est appliqué linéairement de zéro à *ifullscale* dans le mode unipolaire et de *-ifullscale* à *ifullscale* dans le mode bipolaire. Lorsque *kCenter* vaut zéro, les mêmes quantités des parties haute et basse du signal sont écrêtées. Une valeur négative décale la région non écrêtée vers le bas de l'intervalle d'entrée et une valeur positive la décale vers le haut. *ibipolar* doit valoir 1 pour le mode bipolaire et 0 pour le mode unipolaire. Le mode par défaut est unipolaire (*ibipolar* = 0). *ifullscale* fixe l'amplitude maximale des signaux d'entrée et de sortie (1.0 par défaut).

Cela revient à effectuer une distorsion non-linéaire de l'entrée avec la fonction de transfert suivante (normalisée à *ifullscale*=1.0 en mode bipolaire) :

```
1| _____ axe des x en entrée, axe des y en sortie
| /
| / la largeur de la région écrêtée est 2*kWidth
```

```

-1    | 1    la largeur de la région non écrêtée est 2*(1 - kWidth)
----- kCenter décale le région non écrêtée vers
      /|      la gauche ou la droite (jusqu'à kWidth)
      /|
      /|
----- |-1

```

On peut utiliser le mode bipolaire pour une distorsion linéaire directe d'un signal audio. Alternativement, le mode unipolaire est utile pour modifier la sortie d'un phaseur avant de l'utiliser pour indexer une table de fonction, ce qui en fait effectivement une technique de distorsion de phase.

## Voir aussi

*pdhalf, pdhalfy, limit, clip, distort1*

## Exemples

Voici un exemple de l'opcode *pdclip*. Il utilise le fichier *pdclip.csd* [examples/pdclip.csd].

### Exemple 697. Exemple de l'opcode *pdclip*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; test instrument for pdclip opcode
instr 3

    idur  = p3
    iamp  = p4
    ifreq = p5
    ifn   = p6

    kenv  linseg 0, .05, 1.0, idur - .1, 1.0, .05, 0
    aosc  oscil 1.0, ifreq, ifn

    kmod  expseg 0.00001, idur, 1.0
    aout  pdclip aosc, kmod, 0.0, 1.0

    out   kenv*aout*iamp
endin

</CsInstruments>
<CsScore>
f1 0 16385 10 1
f2 0 16385 10 1 .5 .3333 .25 .5

; pdclipped sine wave
i3 0 3 15000 440 1
i3 + 3 15000 330 1
i3 + 3 15000 220 1

```

```
s  
  
; pdclipped composite wave  
i3 0 3 15000 440 2  
i3 + 3 15000 330 2  
i3 + 3 15000 220 2  
e  
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : Anthony Kozar  
Janvier 2008

Nouveau dans la version 5.08 de Csound.

# pdhalf

pdhalf — Distorsion d'un phaseur pour lire les deux moitiés d'une table à des vitesses différentes.

## Description

L'opcode *pdhalf* est conçu pour simuler la méthode "classique" de synthèse par distorsion de phase des synthétiseurs CZ de Casio, du milieu des années 1980. Cette technique lit la première et la seconde moitié d'une table de fonction à différentes vitesses de façon à déformer la forme d'onde. Par exemple, *pdhalf* peut transformer progressivement une onde sinus en une approximation de dent de scie.

## Syntaxe

```
aout pdhalf ain, kShapeAmount [, ibipolar [, ifullscale]]
```

## Initialisation

*ibipolar* -- un paramètre facultatif spécifiant le mode unipolaire (0) ou bipolaire (1). Par défaut, mode unipolaire.

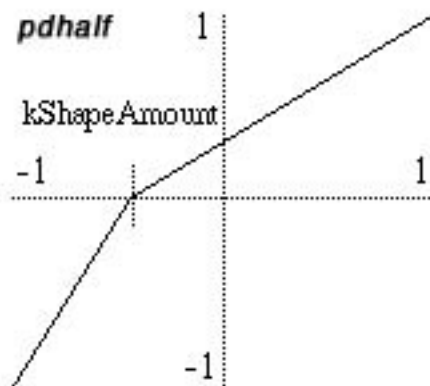
*ifullscale* -- paramètre facultatif spécifiant l'intervalle des valeurs d'entrée et de sortie. Le maximum sera *ifullscale*. Le minimum dépend du mode d'opération : zéro pour le mode unipolaire ou *-ifullscale* pour le mode bipolaire. Vaut 1.0 par défaut. Il faut donner à ce paramètre la valeur maximale attendue en entrée.

## Exécution

*ain* -- le signal d'entrée à distordre.

*aout* -- le signal de sortie.

*kShapeAmount* -- la quantité de distorsion appliquée à l'entrée. Doit être comprise entre -1 et +1. Zéro signifie pas de distorsion.



Fonction de transfert créée par *pdhalf* avec un *kShapeAmount* négatif.

L'opcode *pdhalf* calcule une fonction de transfert composée de deux segments de droite (voir le graphique). Ces segments se touchent en un "point pivot" qui se trouve toujours sur le même axe horizontal. (En mode unipolaire, l'axe est  $y = 0.5$ , et en mode bipolaire c'est l'axe des  $x$ ). Le paramètre *kShapeAmount* indique

l'endroit de l'axe horizontal où se trouve ce point. Lorsque *kShapeAmount* vaut zéro, le point pivot est au milieu de l'intervalle d'entrée, si bien que la fonction de transfert est une droite, ce qui ne provoque aucun changement dans le signal d'entrée. Si *kShapeAmount* varie de 0 à -1, le point pivot se déplace vers la gauche du graphique, produisant un motif de distorsion de phase similaire à la "dent de scie" du CZ de Casio. S'il varie de 0 à 1, le point pivot se déplace vers la droite, produisant un motif inversé.

Si l'entrée de *pdhalf* est un phaseur et que la sortie est utilisée pour indexer une table, les valeurs de *kShapeAmount* inférieures à zéro provoquent une lecture plus rapide de la première moitié de la table que de la seconde moitié. L'inverse est vrai pour les valeurs de *kShapeAmount* supérieures à zéro. Les vitesses de lecture sont calculées de façon à ce que la fréquence du phaseur reste inchangée. Ainsi, cette méthode de distorsion de phase ne peut produire que des partiels harmoniques. Elle ne peut pas produire de bandes latérales inharmoniques comme le fait la modulation de fréquence.

*pdhalf* peut fonctionner en modes unipolaire ou bipolaire. Le mode unipolaire est approprié pour les signaux comme les phaseurs qui varient entre zéro et une valeur maximale (fixée par *ifullscale*). Le mode bipolaire est approprié pour les signaux qui varient de part et d'autre de zéro d'environ la même quantité comme la plupart des signaux audio. L'application directe de *pdhalf* à un tel signal audio produit une sorte de distorsion non linéaire grossière mais ajustable.

Voici un exemple typique de l'utilisation de *pdhalf*

```
aphase    phasor    ifreq
apd        pdhalf    aphase, kamount
aout       tablei    apd, 1, 1
```

## Exemples

Voici un exemple de l'opcode *pdhalf*. Il utilise le fichier *pdhalf.csd* [examples/pdhalf.csd].

### Exemple 698. Exemple de l'opcode *pdhalf*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pdhalf.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 4

    idur          = p3
    iamp          = p4
    ifreq         = p5
    itable        = p6

    aenv          linseg    0, .001, 1.0, idur - .051, 1.0, .05, 0
    aosc          phasor    ifreq
```

```

kamount    linseg    0.0, 0.02, -0.99, 0.05, -0.9, idur-0.06, 0.0
apd        pdhalf    aosc, kamount
aout       tablei    apd, itable, 1

                                outs    aenv*aout*iamp, aenv*aout*iamp

endin

</CsInstruments>
<CsScore>
f1 0 16385 10 1
f2 0 16385 10 1 .5 .3333 .25 .5
f3 0 16385 9 1 1 270          ; inverted cosine

; descending "just blues" scale

; pdhalf with cosine table
; (imitates the CZ-101 "sawtooth waveform")
t 0 100
i4 0 3 .6      512      3
i. + .      .      448
i. + .      .      384
i. + .      .      358.4
i. + .      .      341.33
i. + .      .      298.67
i. + 5      .      256
s
; pdhalf with a sine table
t 0 120
i4 0 3 .6      512      1
i. + .      .      448
i. + .      .      384
i. + .      .      358.4
i. + .      .      341.33
i. + .      .      298.67
i. + 5      .      256
s
; pdhalf with a sawtooth-like table
t 0 150
i4 0 3 .6      512      2
i. + .      .      448
i. + .      .      384
i. + .      .      358.4
i. + .      .      341.33
i. + .      .      298.67
i. + 5      .      256
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*pdhalfy*, *pdclip*

On peut trouver plus d'information au sujet de la synthèse par distorsion de phase sur Wikipedia à [http://en.wikipedia.org/wiki/Phase\\_distortion\\_synthesis](http://en.wikipedia.org/wiki/Phase_distortion_synthesis) [[http://en.wikipedia.org/wiki/Phase\\_distortion\\_synthesis](http://en.wikipedia.org/wiki/Phase_distortion_synthesis)]

## Crédits

Auteur : Anthony Kozar  
Janvier 2008

Nouveau dans la version 5.08 de Csound.



# pdhalfy

pdhalfy — Distorsion d'un phaseur pour lire deux parties inégales d'une table avec la même vitesse.

## Description

L'opcode *pdhalfy* est une variation sur la méthode de distorsion de phase de l'opcode *pdhalf*. Il est utile pour distordre un phaseur afin de lire deux parties inégales d'une table dans le même nombre d'échantillons.

## Syntaxe

```
aout pdhalfy ain, kShapeAmount [, ibipolar [, ifullscale]]
```

## Initialisation

*ibipolar* -- un paramètre facultatif spécifiant le mode unipolaire (0) ou bipolaire (1). Par défaut, mode unipolaire.

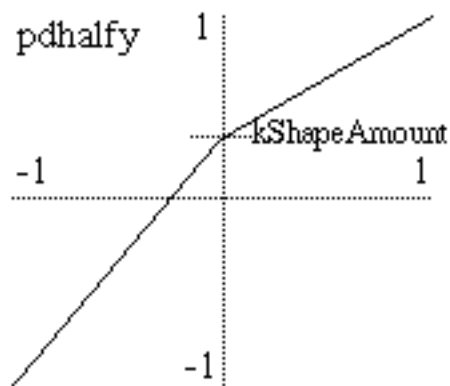
*ifullscale* -- paramètre facultatif spécifiant l'intervalle des valeurs d'entrée et de sortie. Le maximum sera *ifullscale*. Le minimum dépend du mode d'opération : zéro pour le mode unipolaire ou *-ifullscale* pour le mode bipolaire. Vaut 1.0 par défaut. Il faut donner à ce paramètre la valeur maximale attendue en entrée.

## Exécution

*ain* -- le signal d'entrée à distordre.

*aout* -- le signal de sortie.

*kShapeAmount* -- la quantité de distorsion appliquée à l'entrée. Doit être comprise entre -1 et +1. Zéro signifie pas de distorsion.



Fonction de transfert créée par *pdhalfy* avec un *kShapeAmount* négatif.

L'opcode *pdhalfy* calcule une fonction de transfert composée de deux segments de droite (voir le graphique). Ces segments se touchent en un "point pivot" qui se trouve toujours sur le même axe vertical. (En mode unipolaire, l'axe est  $x = 0.5$ , et en mode bipolaire c'est l'axe des  $y$ ). Ainsi, *pdhalfy* est une variation

de l'opcode *pdhalfy* qui place le point pivot du motif de distorsion de phase sur un axe vertical au lieu d'un axe horizontal.

Le paramètre *kShapeAmount* indique l'endroit de l'axe vertical où se trouve ce point. Lorsque *kShapeAmount* vaut zéro, le point pivot est au milieu de l'intervalle de sortie, si bien que la fonction de transfert est une droite, ce qui ne provoque aucun changement dans le signal d'entrée. Si *kShapeAmount* varie de 0 à -1, le point pivot se déplace vers le bas du graphique. S'il varie de 0 à 1, le point pivot se déplace vers le haut, produisant un motif inversé.

Si l'entrée de *pdhalfy* est un phaseur et que la sortie est utilisée pour indexer une table, celle-ci sera divisée en deux parties de différentes tailles, chaque partie étant lue pendant la moitié de la période de l'oscillateur. Les valeurs de *kShapeAmount* inférieures à zéro provoqueront une lecture de moins de la moitié de la table pendant la moitié de la première période d'oscillation. Le reste de la table sera lu pendant la seconde moitié de la période. L'inverse est vrai pour les valeurs de *kShapeAmount* supérieures à zéro. Noter que la fréquence du phaseur reste inchangée. Ainsi, cette méthode de distorsion de phase ne peut produire que des partiels harmoniques. Elle ne peut pas produire de bandes latérales inharmoniques comme le fait la modulation de fréquence. *pdhalfy* tend à avoir une qualité de distorsion plus douce que celle de *pdhalf*.

*pdhalfy* peut fonctionner en modes unipolaire ou bipolaire. Le mode unipolaire est approprié pour les signaux comme les phaseurs qui varient entre zéro et une valeur maximale (fixée par *ifullscale*). Le mode bipolaire est approprié pour les signaux qui varient de part et d'autre de zéro d'environ la même quantité comme la plupart des signaux audio. L'application directe de *pdhalfy* à un tel signal audio produit une sorte de distorsion non linéaire grossière mais ajustable.

Voici un exemple typique de l'utilisation de *pdhalfy*

```
aphase    phasor    ifreq
apd        pdhalfy   aphase, kamount
aout      tablei    apd, 1, 1
```

## Exemples

Voici un exemple de l'opcode *pdhalfy*. Il utilise le fichier *pdhalfy.csd* [examples/pdhalfy.csd].

### Exemple 699. Exemple de l'opcode *pdhalfy*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pdhalfy.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 5

    idur      = p3
    iamp      = p4
```

```

ifreq      = p5
iamtinit   = p6      ; initial amount of phase distortion
iatt       = p7      ; attack time
isuslvl    = p8      ; sustain amplitude
idistdec   = p9      ; time for distortion amount to reach zero
itable     = p10

idec       = idistdec - iatt
irel       = .05
isus       = idur - (idistdec + irel)

aenv       linseg     0, iatt, 1.0, idec, isuslvl, isus, isuslvl, irel, 0, 0, 0
kamount    linseg     -iamtinit, idistdec, 0.0, idur-idistdec, 0.0
aosc       phasor     ifreq
apd        pdhalfy    aosc, kamount
aout       tablei     apd, itable, 1

          outs        aenv*aout*iamp, aenv*aout*iamp

endin

</CsInstruments>
<CsScore>
f1 0 16385 10 1          ; sine
f3 0 16385 9 1 1 270     ; inverted cosine

; descending "just blues" scale

; pdhalfy with cosine table
t 0 100
i5 0 .333 .6 512 1.0 .02 0.5 .12 3
i. + . . 448 <
i. + . . 384 <
i. + . . 358.4 <
i. + . . 341.33 <
i. + . . 298.67 <
i. + 2 . 256 0.5
s

; pdhalfy with sine table
t 0 100
i5 0 .333 .6 512 1.0 .001 0.1 .07 1
i. + . . 448 <
i. + . . 384 <
i. + . . 358.4 <
i. + . . 341.33 <
i. + . . 298.67 <
i. + 2 . 256 0.5
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*pdhalf*, *pdclip*

On peut trouver plus d'information au sujet de la synthèse par distorsion de phase sur Wikipedia à [http://en.wikipedia.org/wiki/Phase\\_distortion\\_synthesis](http://en.wikipedia.org/wiki/Phase_distortion_synthesis) [[http://en.wikipedia.org/wiki/Phase\\_distortion\\_synthesis](http://en.wikipedia.org/wiki/Phase_distortion_synthesis)]

## Crédits

Auteur : Anthony Kozar  
Janvier 2008

Nouveau dans la version 5.08 de Csound.

# peak

**peak** — Maintient la sortie égale à la plus haute valeur absolue reçue.

## Description

Ces opcodes maintiennent dans la variable de sortie de taux-k le niveau de crête absolu reçu jusqu'à présent.

## Syntaxe

```
kres peak asig
```

```
kres peak ksig
```

## Exécution

*kres* -- la sortie est égale à la plus haute valeur absolue reçue jusqu'à présent. C'est également une entrée de l'opcode, car ce dernier lit *kres* pour décider s'il y faut écrire une valeur plus grande.

*ksig* -- signal de taux-k en entrée.

*asig* -- signal de taux-a en entrée.

## Exemples

Voici un exemple de l'opcode **peak**. Il utilise les fichiers *peak.csd* [examples/peak.csd] et *beats.wav* [examples/beats.wav].

### Exemple 700. Exemple de l'opcode **peak**.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o peak.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1 - play an audio file.
instr 1
; Capture the highest amplitude in the "beats.wav" file.
asig soundin "beats.wav"
kp peak asig
```

```
; Print out the peak value once per second.
printk 1, kp

out asig
endin

</CsInstruments>
<CsScore>

; Play Instrument #1, the audio file, for three seconds.
i 1 0 3
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie comprendra des lignes comme celles-ci :

```
i   1 time      0.00002:  4835.00000
i   1 time      1.00002: 29312.00000
i   1 time      2.00002: 32767.00000
```

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Exemple écrit par Kevin Conder.

# pgmassign

`pgmassign` — Affecte un numéro d'instrument à un numéro de programme MIDI spécifié.

## Description

Affecte un numéro d'instrument à un (ou à tous) le(s) programme(s) MIDI spécifié(s).

Par défaut, le numéro de l'instrument est le même que celui du programme. Si l'instrument choisi est inférieur ou égal à zéro, ou n'existe pas, le changement de programme est ignoré. Cet opcode est normalement utilisé dans l'en-tête de l'orchestre. Cependant, comme *massign*, il fonctionne aussi dans les instruments.

## Syntaxe

```
pgmassign ipgm, inst[, ichn]  
pgmassign ipgm, "insname"[, ichn]
```

## Initialisation

*ipgm* -- numéro de programme MIDI (1 à 128). Une valeur de zéro sélectionne tous les programmes.

*inst* -- numéro d'instrument. S'il est inférieur ou égal à zéro, les changements de programme MIDI à *ipgm* sont ignorés. Actuellement, l'affectation à un instrument qui n'existe pas a le même effet. Ceci pourra changer dans une version future afin d'imprimer un message d'erreur.

« *insname* » -- une chaîne de caractères (entre guillemets) représentant un nom d'instrument.

« *ichn* » (facultatif, par défaut zéro) -- numéro de canal. S'il vaut zéro, les changements de programme sont effectués sur tout les canaux.

Vous pouvez empêcher l'activation de n'importe quel instrument en utilisant l'en-tête ci-dessous :

```
massign 0, 0  
pgmassign 0, 0
```

## Exemples

Voici un exemple de l'opcode `pgmassign`. Il utilise le fichier *pgmassign.csd* [examples/pgmassign.csd].

### Exemple 701. Exemple de l'opcode `pgmassign`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
; Audio out  Audio in  No messages  MIDI in  
-odac        -iadc      -d          -M0   ;;RT audio I/O with MIDI in  
; For Non-realtime ouput leave only the line below:  
; -o pgmassign.wav -W ;; for file output any platform  
</CsOptions>
```

```

<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Program 55 (synth vox) uses Instrument #10.
pgmassign 55, 10

; Instrument #10.
instr 10
    ; Just an example, no working code in here!
endin

</CsInstruments>
<CsScore>

; Play Instrument #10 for one second.
i 10 0 1
e

</CsScore>
</CsSoundSynthesizer>

```

Voici un exemple de l'opcode `pgmassign` qui ignorera les évènements de changement de programme. Il utilise le fichier `pgmassign_ignore.csd` [examples/pgmassign\_ignore.csd].

### Exemple 702. Exemple de l'opcode `pgmassign` qui ignorera les évènements de changement de programme.

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac          -iadc      -d           -M0   ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o pgmassign_ignore.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Ignore all program change events.
pgmassign 0, -1

; Instrument #1.
instr 1
    ; Just an example, no working code in here!
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1

```



e

```
</CsScore>
</CsoundSynthesizer>
```

Voici un exemple avancé de l'opcode `pgmassign`. Il utilise le fichier `pgmassign_advanced.csd` [examples/pgmassign\_advanced.csd].

Ne pas oublier qu'il faut inclure l'option `-F` lorsque l'on utilise un fichier MIDI externe comme « `pgmassign_advanced.mid` ».

### Exemple 703. Un exemple avancé de l'opcode `pgmassign`.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -F pgmassign_advanced.mid ;;realtime audio out with midifile in
;-iadc ;;uncomment -iadc if real audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pgmassign_advanced.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

massign 1, 1 ; channels 1 to 4 use instr 1 by default
massign 2, 1
massign 3, 1
massign 4, 1

; pgmassign_advanced.mid can be found in /manual/examples
; pgmassign.mid has 4 notes with these parameters:
;
; Start time Channel Program
;
; note 1 0.5 1 10
; note 2 1.5 2 11
; note 3 2.5 3 12
; note 4 3.5 4 13

pgmassign 0, 0 ; disable program changes
pgmassign 11, 3 ; program 11 uses instr 3
pgmassign 12, 2 ; program 12 uses instr 2

; waveforms for instruments
itmp ftgen 1, 0, 1024, 10, 1
itmp ftgen 2, 0, 1024, 10, 1, 0.5, 0.3333, 0.25, 0.2, 0.1667, 0.1429, 0.125
itmp ftgen 3, 0, 1024, 10, 1, 0, 0.3333, 0, 0.2, 0, 0.1429, 0, 0.10101

instr 1 /* sine */

kcps cpsmidib 2 ; note frequency
asnd oscili .6, kcps, 1
outs asnd, asnd

endin

instr 2 /* band-limited sawtooth */

kcps cpsmidib 2 ; note frequency
```

```
asnd oscili .6, kcps, 2
outs asnd, asnd

endin

instr 3 /* band-limited square */

kcps cpsmidib 2 ; note frequency
asnd oscili .6, kcps, 3
outs asnd, asnd

endin

</CsInstruments>
<CsScore>

t 0 120
f 0 8.5 2 -2 0
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*midichn* et *massign*

## Crédits

Auteur : Istvan Varga  
Mai 2002

Nouveau dans la version 4.20

# phaser1

phaser1 — Filtres passe-tout du premier ordre arrangés en série.

## Description

Une implémentation de *iord* filtres passe-tout du premier ordre en série.

## Syntaxe

```
ares phaser1 asig, kfreq, kord, kfeedback [, iskip]
```

## Initialisation

*iskip* (facultatif, 0 par défaut) -- utilisé pour contrôler l'état initial de la mémoire interne. Comme le filtrage comprend une boucle rétroactive de la sortie précédente, l'état initial de la mémoire utilisée est significatif. Une valeur nulle efface l'état ; une valeur non-nulle permet de conserver l'information précédente. La valeur par défaut est 0.

## Exécution

*kfreq* -- fréquence (en Hz) du(des) filtre(s). C'est la fréquence à laquelle chaque filtre de la série déphase sont entrée de 90 degrés.

*kord* -- le nombre d'étages passe-tout en série. Ce sont des filtres du premier ordre et il peut y en avoir de 1 à 4999.



### Note

Bien que *kord* soit présenté au taux-k, il n'est en fait lu qu'à l'initialisation. Ainsi, si l'on utilise un argument de taux-k, il faut lui affecter une valeur avec *init*.

*kfeedback* -- quantité du signal de sortie qui est réinjectée dans l'entrée de la chaîne passe-tout. Plus la rétroaction est importante et plus il y aura d'encoches proéminentes dans le spectre de la sortie. *kfeedback* doit être compris entre -1 et +1 pour la stabilité.

*phaser1* implémente *iord* sections passe-tout du premier ordre, connectées en série, partageant toutes le même coefficient. Chaque section passe-tout peut être représentée par l'équation aux différences suivante :

$$y(n) = C * x(n) + x(n-1) - C * y(n-1)$$

où  $x(n)$  est l'entrée,  $x(n-1)$  est l'entrée précédente,  $y(n)$  est la sortie,  $y(n-1)$  est la sortie précédente et  $C$  est un coefficient qui est calculé à partir de la valeur de *kfreq* en utilisant une transformée en  $z$  bilinéaire.

En faisant varier *kfreq* lentement et en mélangeant la sortie globale de la chaîne passe-tout à l'entrée, on obtient l'effet "phase shifter" classique, avec des encoches se déplaçant en fréquence dans les deux directions. On obtient les meilleurs résultats avec *iord* compris entre 4 et 16. Lorsque l'entrée est mélangée avec la sortie, 1 encoche est générée pour chaque couple d'étages passe-tout ; ainsi avec *iord* = 6, il y aura 3 encoches dans la sortie. Avec des valeurs plus importantes de *iord*, en modulant *kfreq* on obtiendra une forme de modulation non-linéaire de la hauteur.

## Exemples

Voici un exemple de l'opcode phaser1. Il utilise le fichier *phaser1.csd* [examples/phaser1.csd].

### Exemple 704. Exemple de l'opcode phaser1.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o phaser1.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; demonstration of phase shifting abilities of phaser1.
instr 1
; Input mixed with output of phaser1 to generate notches.
; Shows the effects of different iorder values on the sound
idur   = p3
iamp   = p4 * .05
iorder = p5          ; number of 1st-order stages in phaser1 network.
                        ; Divide iorder by 2 to get the number of notches.
ifreq  = p6          ; frequency of modulation of phaser1
ifeed  = p7          ; amount of feedback for phaser1

kamp   linseg 0, .2, iamp, idur - .2, iamp, .2, 0

iharms = (sr*.4) / 100

asig   gbuzz 1, 100, iharms, 1, .95, 2 ; "Sawtooth" waveform modulation oscillator for phaser1 ugen.
kfreq  oscili 5500, ifreq, 1
kmod   = kfreq + 5600

aphs   phaser1 asig, kmod, iorder, ifeed

out     (asig + apha) * iamp
endin

</CsInstruments>
<CsScore>

; inverted half-sine, used for modulating phaser1 frequency
f1 0 16384 9 .5 -1 0
; cosine wave for gbuzz
f2 0 8192 9 1 1 .25

; phaser1
i1 0 5 7000 4 .2 .9
i1 6 5 7000 6 .2 .9
i1 12 5 7000 8 .2 .9
i1 18 5 7000 16 .2 .9
i1 24 5 7000 32 .2 .9
i1 30 5 7000 64 .2 .9
```

e

```
</CsScore>  
</CsoundSynthesizer>
```

## Historique Technique

On trouve dans Hartmann [1] une description générale des différences entre flanging et phasing. On peut trouver une implémentation ancienne de filtres passe-tout du premier ordre connectés en série dans Beigel [2], dans laquelle la transformation en  $z$  bilinéaire est utilisée pour déterminer la fréquence du déphasage de chaque étage. Cronin [3] présente une implémentation similaire pour un réseau déphaseur à quatre étages. Chamberlin [4] et Smith [5] discutent tous deux de l'utilisation de sections passe-tout du deuxième ordre pour un meilleur contrôle de la profondeur, de la largeur et de la fréquence des encoches.

## Références

1. Hartmann, W.M. "Flanging and Phasers." Journal of the Audio Engineering Society, Vol. 26, No. 6, pp. 439-443, juin 1978.
2. Beigel, Michael I. "A Digital 'Phase Shifter' for Musical Applications, Using the Bell Labs (Allens-Fischer) Digital Filter Module." Journal of the Audio Engineering Society, Vol. 27, No. 9, pp. 673-676, septembre 1979.
3. Cronin, Dennis. "Examining Audio DSP Algorithms." Dr. Dobb's Journal, juillet 1994, p. 78-83.
4. Chamberlin, Hal. Musical Applications of Microprocessors. Second edition. Indianapolis, Indiana: Hayden Books, 1985.
5. Smith, Julius O. "An Allpass Approach to Digital Phasing and Flanging." Proceedings of the 1984 ICMC, p. 103-108.

## Voir aussi

*phaser2*

D'autres informations au sujet des phaseurs sur Wikipedia : [http://en.wikipedia.org/wiki/Phaser\\_\(effect\)](http://en.wikipedia.org/wiki/Phaser_(effect))  
[[http://en.wikipedia.org/wiki/Phaser\\_\(effect\)](http://en.wikipedia.org/wiki/Phaser_(effect))]

## Crédits

Auteur : Sean Costello  
Seattle, Washington  
1999

Novembre 2002. Ajout d'une note sur le paramètre *kord*, grâce à Rasmus Ekman.

Nouveau dans la version 4.0 de Csound.

# phaser2

phaser2 — Filtres passe-tout du second ordre arrangés en série.

## Description

Une implémentation de *iord* filtres passe-tout du second ordre en série.

## Syntaxe

ares **phaser2** asig, kfreq, kq, kord, kmode, ksep, kfeedback

## Exécution

*kfreq* -- fréquence (en Hz) du(des) filtre(s). C'est la fréquence centrale de l'encoche du premier filtre passe-tout de la série. Cette fréquence est utilisée comme fréquence de base à partir de laquelle les fréquences des autres encoches sont dérivées.

*kq* -- Q de chaque encoche. Des valeurs élevées de Q donnent des encoches étroites. Un Q compris entre 0.5 et 1 donne l'effet de "phasing" le plus fort, mais des valeurs de Q plus grandes peuvent être utilisées pour des effets spéciaux.

*kord* -- le nombre d'étages passe-tout en série. Ce sont des filtres du second ordre et il peut y en avoir de 1 à 2499. Avec des ordres plus élevés, le temps de calcul augmente.

*kfeedback* -- quantité du signal de sortie qui est réinjectée dans l'entrée de la chaîne passe-tout. Plus la rétroaction est importante et plus il y aura d'encoches proéminentes dans le spectre de la sortie. *kfeedback* doit être compris entre -1 et +1 pour la stabilité.

*kmode* -- utilisé pour le calcul des fréquence des encoches.



### Note

Bien que *kord* et *kmode* soient présentés au taux-k, ils ne sont en fait lus qu'à l'initialisation. Ainsi, si l'on utilise des arguments de taux-k, il faut leur affecter une valeur avec *init*.

*ksep* -- facteur de mise à l'échelle utilisé en conjonction avec *imode* pour déterminer les fréquences des encoches ajoutées dans le spectre de sortie.

*phaser2* implémente *iord* sections passe-tout du second ordre, connectées en série. L'utilisation de sections passe-tout du second ordre permet un placement précis de la fréquence, de la largeur et de la profondeur des encoches dans le spectre de fréquence. *iord* est utilisé pour déterminer directement le nombre d'encoches dans le spectre ; par exemple pour *iord* = 6, il y aura 6 encoches dans le spectre de sortie.

Il y a deux modes possibles de détermination des fréquences d'encoche. Lorsque *imode* = 1, les fréquences d'encoche sont déterminées par la fonction suivante :

fréquence de l'encoche N = kbf + (ksep \* kbf \* N-1)

Par exemple, avec *imode* = 1 et *ksep* = 1, les encoches seront en relation harmonique avec la fréquence d'encoche déterminée par *kfreq* (ainsi, s'il y a 8 encoches, la première étant à 100 Hz, les suivantes seront à 200, 300, 400, 500, 600, 700 et 800 Hz). C'est utile pour générer un effet de "filtre en peigne", dont le

nombre d'encoches est déterminé par *iord*. Différentes valeurs de *ksep* donnent des fréquences d'encoche inharmoiques et d'autres effets spéciaux. *ksep* peut être balayé pour créer un mouvement d'expension ou de contraction des fréquences d'encoche. Les soufflets d'un accordéon en mouvement donnent une bonne analogie visuelle de l'effet du balayage de *ksep* - les encoches sont séparées, puis compressées ensemble, lorsque *ksep* change.

Lorsque *imode* = 2, les encoches successives sont des puissances du paramètre d'entrée *ksep* multipliées par la fréquence d'encoche initiale donnée par *kfreq*. On peut ainsi régler les fréquences d'encoche en octaves ou sur d'autres intervalles musicaux. Par exemple, les lignes suivantes généreront 8 encoches dans le spectre de sortie, les encoches étant réparties sur les octaves supérieures de *kfreq*:

```
aphs phaser2 ain, kfreq, 0.5, 8, 2, 2, 0
aout =      ain + aphis
```

Lorsque *imode* = 2, la valeur de *ksep* doit être supérieure à 0. *ksep* peut être balayé pour créer une compression et une expension des fréquences d'encoche (avec des effets plus dramatiques que lorsque *imode* = 1).

## Exemples

Voici un exemple de l'opcode *phaser2*. Il utilise le fichier *phaser2.csd* [examples/phaser2.csd].

### Exemple 705. Exemple de l'opcode *phaser2*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o phaser2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 2
; demonstration of phase shifting abilities of phaser2.
; Input mixed with output of phaser2 to generate notches.
; Demonstrates the interaction of imode and ksep.
idur   = p3
iamp   = p4 * .04
iorder = p5      ; number of 2nd-order stages in phaser2 network
ifreq  = p6      ; not used
ifeed  = p7      ; amount of feedback for phaser2
imode  = p8      ; mode for frequency scaling
isep   = p9      ; used with imode to determine notch frequencies
kamp   linseg 0, .2, iamp, idur - .2, iamp, .2, 0
iharms = (sr*.4) / 100

; "Sawtooth" waveform exponentially decaying function, to control notch frequencies
asig   gbuzz 1, 100, iharms, 1, .95, 2
kline  expseg 1, idur, .005
aphs   phaser2 asig, kline * 2000, .5, iorder, imode, isep, ifeed

out (asig + aphis) * iamp
```

```
endin

</CsInstruments>
<CsScore>

; cosine wave for gbuzz
f2 0 8192 9 1 1 .25

; phaser2, imode=1
i2 00 10 7000 8 .2 .9 1 .33
i2 11 10 7000 8 .2 .9 1 2

; phaser2, imode=2
i2 22 10 7000 8 .2 .9 2 .33
i2 33 10 7000 8 .2 .9 2 2
e

</CsScore>
</CsoundSynthesizer>
```

## Historique Technique

On trouve dans Hartmann [1] une description générale des différences entre flanging et phasing. On peut trouver une implémentation ancienne de filtres passe-tout du premier ordre connectés en série dans Beigel [2], dans laquelle la transformation en  $z$  bilinéaire est utilisée pour déterminer la fréquence du déphasage de chaque étage. Cronin [3] présente une implémentation similaire pour un réseau déphaseur à quatre étages. Chamberlin [4] et Smith [5] discutent tous deux de l'utilisation de sections passe-tout du deuxième ordre pour un meilleur contrôle de la profondeur, de la largeur et de la fréquence des encoches.

## Références

1. Hartmann, W.M. "Flanging and Phasers." Journal of the Audio Engineering Society, Vol. 26, No. 6, pp. 439-443, Juin 1978.
2. Beigel, Michael I. "A Digital 'Phase Shifter' for Musical Applications, Using the Bell Labs (Alles-Fischer) Digital Filter Module." Journal of the Audio Engineering Society, Vol. 27, No. 9, pp. 673-676, Septembre 1979.
3. Cronin, Dennis. "Examining Audio DSP Algorithms." Dr. Dobb's Journal, Juillet 1994, p. 78-83.
4. Chamberlin, Hal. Musical Applications of Microprocessors. Second edition. Indianapolis, Indiana: Hayden Books, 1985.
5. Smith, Julius O. "An Allpass Approach to Digital Phasing and Flanging." Proceedings of the 1984 ICMC, p. 103-108.

## Voir aussi

*phaser1*

D'autres informations au sujet des phaseurs sur Wikipedia : [http://en.wikipedia.org/wiki/Phaser\\_\(effect\)](http://en.wikipedia.org/wiki/Phaser_(effect))  
[[http://en.wikipedia.org/wiki/Phaser\\_\(effect\)](http://en.wikipedia.org/wiki/Phaser_(effect))]

## Crédits

Auteur : Sean Costello



Seattle, Washington  
1999

Novembre 2002. Ajout d'une note sur les paramètres *kord* et *kmode* grâce à Rasmus Ekman.

Nouveau dans la version 4.0 de Csound.

# phasor

phasor — Produit une valeur de phase mobile normalisée.

## Description

Produit une valeur de phase mobile normalisée.

## Syntaxe

```
ares phasor xcps [ , iphs]  
kres phasor kcps [ , iphs]
```

## Initialisation

*iphs* (facultatif) -- phase initiale, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation de la phase sera ignorée. La valeur par défaut est zéro.

## Exécution

Une phase interne est augmentée successivement selon la fréquence de *kcps* ou de *xcps* pour produire une valeur de phase mobile, normalisée pour se trouver dans l'intervalle  $0 \leq \text{phs} < 1$ .

Lorsqu'elle est utilisée comme indice dans une *table*, cette phase (multipliée par la longueur de la table de fonction) permettra de l'utiliser comme un oscillateur.

Noter que *phasor* est une sorte d'intégrateur, accumulant les incréments de phase qui représentent les réglages de fréquence.

## Exemples

Voici un exemple de l'opcode phasor. Il utilise le fichier *phasor.csd* [exemples/phasor.csd].

### Exemple 706. Exemple de l'opcode phasor.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac      ;;realtime audio out  
;-iadc      ;;uncomment -iadc if real audio input is needed too  
; For Non-realtime ouput leave only the line below:  
; -o phasor.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 32  
nchnls = 2  
0dbfs = 1
```

```

instr 1

ifn = 1    ;read table 1 with our index
ixmode = 1
kndx phasor p4
kfrq table kndx, ifn, ixmode
asig poscil .6, kfrq, 2 ;re-synthesize with sine
    outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 1025 -7 200 1024 2000 ;a line from 200 to 2,000
f 2 0 16384 10 1;sine wave

i 1 0 1 1 ;once per second
i 1 2 2 .5 ;once per 2 seconds
i 1 5 1 2 ;twice per second
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

Les opcodes d'Accès aux Tables comme : *table*, *tablei*, *table3* et *tab*.

Aussi : *phasorbnk*.

# phasorbnk

phasorbnk — Produit un nombre arbitraire de valeurs de phase mobiles normalisées.

## Description

Produit un nombre arbitraire de valeurs de phase mobiles normalisées, accessibles par un indice.

## Syntaxe

```
ares phasorbnk xcps, kndx, icnt [ , iphs]
kres phasorbnk kcps, kndx, icnt [ , iphs]
```

## Initialisation

*icnt* -- nombre maximum de phaseurs à utiliser.

*iphs* -- phase initiale, exprimée comme une fraction d'une période (0 à 1). Si elle vaut -1, l'initialisation sera ignorée. Si *iphs*>1 chaque phaseur sera initialisé avec une valeur aléatoire.

## Exécution

*kndx* -- valeur d'indice pour accéder aux phaseurs individuellement

Pour chaque phaseur indépendant, une phase interne est augmentée successivement selon la fréquence de *kcps* ou de *xcps* pour produire une valeur de phase mobile, normalisée pour se trouver dans l'intervalle  $0 \leq \text{phs} < 1$ . On accède à chaque phaseur individuel par l'indice *kndx*.

On peut utiliser ce banc de phaseurs dans une boucle de taux-k pour générer plusieurs voix indépendantes, ou en conjonction avec l'opcode *adsynt* pour changer les paramètres dans les tables utilisées par *adsynt*.

## Exemples

Voici un exemple de l'opcode phasorbnk. Il utilise le fichier *phasorbnk.csd* [examples/phasorbnk.csd].

### Exemple 707. Exemple de l'opcode phasorbnk.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o phasorbnk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
```

```
kr = 4410
ksmps = 10
nchnls = 1

; Generate a sinewave table.
giwave ftgen 1, 0, 1024, 10, 1

; Instrument #1
instr 1
  ; Generate 10 voices.
  icnt = 10
  ; Empty the output buffer.
  asum = 0
  ; Reset the loop index.
  kindex = 0

; This loop is executed every k-cycle.
loop:
  ; Generate non-harmonic partials.
  kcps = (kindex+1)*100+30
  ; Get the phase for each voice.
  aphas phasorbnk kcps, kindex, icnt
  ; Read the wave from the table.
  asig table aphas, giwave, 1
  ; Accumulate the audio output.
  asum = asum + asig

  ; Increment the index.
  kindex = kindex + 1

  ; Perform the loop until the index (kindex) reaches
  ; the counter value (icnt).
  if (kindex < icnt) kgoto loop

  out asum*3000
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Génère plusieurs voix avec des partiels indépendants. Cet exemple est meilleur avec *adsynt*. Voir aussi l'exemple de la notice *adsynt* pour une utilisation de *phasorbnk* au taux-k.

## Crédits

Auteur : Peter Neubäcker  
Munich, Allemagne  
Août 1999

Nouveau dans la version 3.58 de Csound

# phs

phs — Retourne les arguments d'un tableau de nombres complexes.

## Description

Cet opcode retourne les arguments d'un tableau de nombres complexes dans un tableau de réels dont la taille est la moitié de celle du tableau d'entrée plus un. Ce point supplémentaire est là pour que la taille du tableau soit équivalente à celle de la sortie de l'opcode *mags* (qui est souvent utilisé conjointement avec celui-ci).

## Syntaxe

```
kout[] phs kin[]
```

## Exécution

*kout[]* -- tableau contenant les arguments. Créé s'il n'existe pas.

*kin[]* -- tableau contenant les valeurs complexes réelles-imaginaires d'entrée.

## Exemples

Voici un exemple de l'opcode phs. Il utilise le fichier *phs.csd* [examples/phs.csd].

### Exemple 708. Exemple de l'opcode phs.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-d -o dac
</CsOptions>
<CsInstruments>
/* ksmpls needs to be an integer div of
   hopsize */
ksmps = 64
0dbfs=1
nchnls=2

opcode PVA,k[]k[]k,aii
  asig,ysize,ihop xin
  iolaps init ysize/ihop
  kcmt init 0
  krow init 1
  kIn[] init ysize
  kOlph[] init ysize/2 + 1
  ifac = (sr/(ihop*2*$M_PI))
  iscal = (2*$M_PI*ihop/ysize)
  kfl = 0
  kIn shiftin asig
  if kcmt == ihop then
    kWin[] window kIn,krow*ihop
    kSpec[] rfft kWin
    kMags[] mags kSpec
```

```

kPha[] phs kSpec
kDelta[] = kPha - kOlph
kOlph = kPha
kk = 0
kDelta unwrap kDelta
while kk < isize/2 do
    kPha[kk] = (kDelta[kk] + kk*iscal)*ifac
    kk += 1
od
krow = (krow+1)%iolaps
kcnt = 0
kfl = 1
endif
xout kMags,kPha,kfl
kcnt += ksmps
endop

opcode PVS,a,k[]k[]kii
kMags[],kFr[],kfl,isize,ihop xin
iolaps init isize/ihop
ifac = ihop*2*$M_PI/sr;
iscal = sr/isize
krow init 0
kOla[] init isize
kOut[][] init iolaps,isize
kPhs[] init isize/2+1
if kfl == 1 then
    kk = 0
    while kk < isize/2 do
        kFr[kk] = (kFr[kk] - kk*iscal)*ifac
        kk += 1
    od
    kPhs = kFr + kPhs
    kSpec[] pol2rect kMags,kPhs
    kRow[] rifft kSpec
    kWin[] window kRow, krow*ihop
    kOut setrow kWin, krow
    kOla = 0
    kk = 0
    until kk == iolaps do
        kRow getrow kOut, kk
        kOla = kOla + kRow
        kk += 1
    od
    krow = (krow+1)%iolaps
endif
xout shiftout(kOla)/iolaps
endop

instr 1

ihopsize = 256 ; hopsize
ifftsize = 2048 ; FFT size
kFregsOut[] init ifftsize/2+1 ; synthesis freqs
kMagsOut[] init ifftsize/2+1 ; synthesis mags

a1 diskin2 "fox.wav",1,0,1

kMags[],kFregs[],kflg PVA a1,ifftsize,ihopsize

if kflg == 1 then
    ki = 0
    kMagsOut = 0
    kFregsOut = 0
    iscal = 1.5
    until ki == ifftsize/2 do
        if ki*iscal < ifftsize/2 then

```

```
        kFreqsOut[ki*iscal] = kFreqs[ki]*iscal
        kMagsOut[ki*iscal] = kMags[ki]
    endif
    ki += 1
od
endif

a2 PVS kMagsOut,kFreqsOut,kflg,ifftsize,ihopsize

a1 delay a1, (ifftsize+ihopsize)/sr
outs a1, a2
endin

</CsInstruments>
<CsScore>
i1 0 10
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux.

## Crédits

Auteur : Victor Lazzarini  
NUI Maynooth  
2014

Nouveau dans la version 6.04



# pindex

pindex — Retourne la valeur d'un p-champ spécifié.

## Description

*pindex* retourne la valeur d'un p-champ spécifié.

## Syntaxe

```
ivalue pindex ipfieldIndex
```

## Initialisation

*ipfieldIndex* -- numéro du p-champ à interroger.

*ivalue* - valeur du p-champ.

## Exemples

Voici un exemple de l'opcode pindex. Il utilise le fichier *pindex.csd* [examples/pindex.csd].

### Exemple 709. Exemple de l'opcode pindex.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages  MIDI in
-odac          -iadc    ; -d          -M0   ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
;-o pindex.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
;Example by Anthony Kozar Dec 2006

instr 1
  inum    pcount
  index   init 1
  loop1:
    ivalue pindex index
    printf_i "p%d = %f\n", 1, index, ivalue
    index  = index + 1
    if (index <= inum) igoto loop1
  print inum
endin

</CsInstruments>
<CsScore>
i1 0 3 40 50          ; has 5 pfields
i1 1 2 80             ; has 5 due to carry
i1 2 1 40 50 60 70    ; has 7
e
</CsScore>
```

```
</CsoundSynthesizer>
```

L'exemple produira la sortie suivante :

```
new alloc for instr 1:
WARNING: instr 1 uses 3 p-fields but is given 5
p1 = 1.000000
p2 = 0.000000
p3 = 3.000000
p4 = 40.000000
p5 = 50.000000
instr 1: inum = 5.000
B 0.000 .. 1.000 T 1.000 TT 1.000 M: 0.0
new alloc for instr 1:
WARNING: instr 1 uses 3 p-fields but is given 5
p1 = 1.000000
p2 = 1.000000
p3 = 2.000000
p4 = 80.000000
p5 = 50.000000
instr 1: inum = 5.000
B 1.000 .. 2.000 T 2.000 TT 2.000 M: 0.0
new alloc for instr 1:
WARNING: instr 1 uses 3 p-fields but is given 7
p1 = 1.000000
p2 = 2.000000
p3 = 1.000000
p4 = 40.000000
p5 = 50.000000
p6 = 60.000000
p7 = 70.000000
instr 1: inum = 7.000
```

On peut ignorer les avertissements, car les p-champs sont utilisés indirectement par *pindex* plutôt que de manière explicite par p4, p5, etc.

Voici un autre exemple de l'opcode *pindex*. Il utilise le fichier *pindex-2.csd* [examples/pindex-2.csd].

### Exemple 710. Second exemple de l'opcode *pindex*.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac ;;realtime audio
;-iadc ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pindex-2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

; This UDO returns a pfield value but returns 0 if it does not exist.
opcode mypvalue, i, i
index xin
inum pcount
if (index > inum) then
iout = 0.0
else
iout pindex index
```

```

endif

xout iout
endop

; Envelope UDO that reads parameters from a flexible number of pfields
; Syntax: kenv flexlinseg ipstart
;         ipstart is the first pfield of the envelope
;         parameters. Reads remaining pfields (up to 21 of them).
;         kenv is the output envelope.

opcode flexlinseg, k, i
  ipstart xin

  iep1 mypvalue ipstart
  iep2 mypvalue ipstart + 1
  iep3 mypvalue ipstart + 2
  iep4 mypvalue ipstart + 3
  iep5 mypvalue ipstart + 4
  iep6 mypvalue ipstart + 5
  iep7 mypvalue ipstart + 6
  iep8 mypvalue ipstart + 7
  iep9 mypvalue ipstart + 8
  iepa mypvalue ipstart + 9
  iepb mypvalue ipstart + 10
  iepc mypvalue ipstart + 11
  iepd mypvalue ipstart + 12
  iepe mypvalue ipstart + 13
  iepf mypvalue ipstart + 14
  iepg mypvalue ipstart + 15
  ieph mypvalue ipstart + 16
  iepi mypvalue ipstart + 17
  iepj mypvalue ipstart + 18
  iepk mypvalue ipstart + 19
  iepl mypvalue ipstart + 20

  kenv linseg iep1, iep2, iep3, iep4, iep5, iep6, iep7, iep8, \
            iep9, iepa, iepb, iepc, iepd, iepe, iepf, iepg, \
            ieph, iepi, iepj, iepk, iepl

  xout kenv
endop

instr 1
; This instrument only requires 3 pfields but can accept up to 24.
; (You will still get warnings about more than 3).
kenv flexlinseg 4 ; envelope params start at p4
aout oscili kenv*.5, 256, 1
outs aout, aout

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1

i1 0 2 0.0 1.0 1.0 1.0 0.0
i1 2 2 0.0 0.1 1.0 1.0 1.0 0.1 0.0
i1 4 2 0.0 0.5 0.0 ; one problem is that "missing" pfields carry
i1 6 2 0.0 0.5 0.0 ! ; now we can fix this problem with !
i1 8 10 0.0 3.0 1.0 0.3 0.1 0.3 1.0 0.3 0.1 0.3 1.0 0.3 0.1 0.8 0.9 5.0 0.0

e
</CsScore>
</CsoundSynthesizer>

```

L'exemple produit la sortie suivante :

```
WARNING: instr 1 uses 3 p-fields but is given 8
B 0.000 .. 2.000 T 2.000 TT 2.000 M: 0.49966 0.49966
WARNING: instr 1 uses 3 p-fields but is given 10
B 2.000 .. 4.000 T 4.000 TT 4.000 M: 0.50000 0.50000
WARNING: instr 1 uses 3 p-fields but is given 10
B 4.000 .. 6.000 T 6.000 TT 6.000 M: 0.49943 0.49943
WARNING: instr 1 uses 3 p-fields but is given 6
B 6.000 .. 8.000 T 8.000 TT 8.000 M: 0.00000 0.00000
WARNING: instr 1 uses 3 p-fields but is given 20
B 8.000 .. 18.000 T 18.000 TT 18.000 M: 0.49994 0.49994
```

## Voir aussi

*pcount*

## Crédits

Exemple par : Anthony Kozar

Décembre 2006

# pinker

pinker — Génère du bruit rose.

## Description

Génère du bruit rose (réponse à -3dB/oct) avec l'algorithme *New Shade of Pink* de Stefan Stenzel.

## Syntaxe

ares **pinker**

## Exécution

*pinker* génère du bruit rose (bruit ayant la même énergie dans chaque octave) avec l'algorithme de Stefan Stenzel. Pour une présentation détaillée de l'algorithme voir <http://stenzel.waldorfmusic.de/post/pink/>.

## Exemples

Voici un exemple de l'opcode *pinker*. Il utilise le fichier *pinker.csd* [examples/pinker.csd].

### Exemple 711. Exemple de l'opcode *pinker*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>

<CsInstruments>
ksmps = 12
nchnls = 1
0dbfs = 1

instr 1
  a1 pinker
    out a1
endin
</CsInstruments>

<CsScore>
i1 0 10
e
</CsScore>

</CsoundSynthesizer>
```

## Crédits

Auteurs : Stefan Stenzel and John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2014

Nouveau dans la version 6.04 de Csound

# pinkish

pinkish — Génère une approximation d'un bruit rose.

## Description

Génère une approximation d'un bruit rose (réponse à -3dB/oct) par une de ces deux méthodes :

- un générateur de bruit à taux multiples d'après Moore, codé par Martin Gardner
- un banc de filtres dessinés par Paul Kellet

## Syntaxe

```
ares pinkish xin [, imethod] [, inumbands] [, iseed] [, iskip]
```

## Initialisation

*imethod* (facultatif, par défaut=0) -- sélectionne la méthode de filtrage :

- 0 = méthode de Gardner (par défaut).
- 1 = banc de filtres de Kellet.
- 2 = Un banc de filtres quelque peu plus rapides par Kellet, avec une réponse moins précise.

*inumbands* (facultatif) -- ne fonctionne qu'avec la méthode de Gardner. Nombre de bandes de bruit à générer. Le maximum vaut 32 et le minimum vaut 4. Les valeurs plus élevées donnent un spectre plus lisse, mais au-delà de 20 bandes il y aura des fluctuations lentes presque comme une composante continue. La valeur par défaut est 20.

*iseed* (facultatif, par défaut=0) -- ne fonctionne qu'avec la méthode de Gardner. s'il est non nul, sert de graine au générateur de nombres aléatoires. S'il est nul, le générateur sera initialisé à partir de la valeur de l'horloge. Vaut 0 par défaut.

*iskip* (facultatif, par défaut=0) -- s'il est non nul, l'état interne n'est pas (ré)initialisé (utile pour les notes liées). Vaut 0 par défaut.

## Exécution

*xin* -- pour la méthode de Gardner : amplitude de taux-k ou -a. Pour les filtres de Kellet : normalement un bruit de taux-a de distribution uniforme obtenu à partir de *rand* (31-bit) ou de *unirand*, mais ça peut être n'importe quel signal de taux-a. La valeur de crête de la sortie varie largement ( $\pm 15\%$ ) même sur de longues périodes, et sera habituellement d'un niveau bien inférieur à celui de l'amplitude de l'entrée. Les valeurs de crête peuvent aussi dépasser occasionnellement l'amplitude de l'entrée ou celle du bruit.

*pinkish* tente de générer un bruit rose (c-à-d un bruit avec la même énergie dans chaque octave), par une des deux méthodes suivantes.

La première méthode, par Moore & Gardner, ajoute plusieurs signaux de bruit blanc (jusqu'à 32), générés à des taux en octave (*sr*, *sr/2*, *sr/4*, etc). Les valeurs pseudo aléatoires sont obtenues à partir d'un générateur interne sur 32 bit. Ce générateur est local à chaque instance de l'opcode et initialisable (comme pour *rand*).

La seconde méthode est un filtrage passe-bas avec une réponse d'environ -3dB/oct. Si l'entrée est un bruit blanc uniforme, la sortie sera un bruit rose. Avec cette méthode, on peut utiliser n'importe quel signal comme entrée. Le filtre de haute qualité est plus lent, mais il a moins d'ondulations et un intervalle de fréquences opératoires légèrement plus large que les versions moins gourmandes en calcul. Avec les filtres de Kellet, il n'y a pas de graine pour le générateur de nombres aléatoires.

La réponse en fréquence de la sortie dans la méthode de Gardner comporte quelques anomalies dans les intervalles basse-moyenne et moyenne-haute fréquence. On peut générer plus d'énergie en basse fréquence en augmentant le nombre de bandes. Cette méthode est aussi un peu plus rapide. Le filtre raffiné de Kellet a un spectre très lisse, mais un intervalle efficace plus limité. Le niveau augmente légèrement à l'extrémité haute du spectre.

## Exemples

Voici un exemple de l'opcode pinkish. Il utilise le fichier *pinkish.csd* [examples/pinkish.csd].

### Exemple 712. Exemple de l'opcode pinkish.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pinkish.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  awhite unirand 2.0

  ; Normalize to +/-1.0
  awhite = awhite - 1.0

  apink pinkish awhite, 1, 0, 0, 1

  out apink * 30000
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Bruit filtré (Kellet) pour une note liée (*iskip* est non nul).

## Crédits

Auteurs : Phil Burk et John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la version 4.07 de Csound

Adapté pour Csound par Rasmus Ekman

La méthode par bandes de bruit est due à F. R. Moore (ou R. F. Voss), et fut présentée par Martin Gardner dans un article de Scientific American souvent cité. La présente version fut codée par Phil Burk après une discussion sur la liste de diffusion de music-dsp, avec des optimisations significatives suggérées par James McCartney.

Le banc de filtres a été dessiné par Paul Kellet, et posté sur la liste de diffusion de music-dsp.

La discussion complète sur le bruit rose a été rassemblée sur une page HTML par Robin Whittle, qui est actuellement consultable à <http://www.firstpr.com.au/dsp/pink-noise/>.

Notes ajoutées par Rasmus Ekman en Septembre 2002.



# pitch

pitch — Suit la hauteur d'un signal.

## Description

En utilisant les mêmes techniques que *spectrum* et *specptrk*, *pitch* suit la hauteur du signal sous la forme octave point décimal ainsi que l'amplitude en dB.

## Syntaxe

```
koct, kamp pitch asig, iupdte, ilo, ihi, idbthresh [, ifrqs] [, iconf] \  
[, istrtr] [, iocts] [, iq] [, inptls] [, irolloff] [, iskip]
```

## Initialisation

*iupdte* -- longueur en secondes de la période de mise à jour des sorties.

*ilo, ihi* -- intervalle dans lequel la hauteur est détectée, exprimé en octave point décimal.

*idbthresh* -- amplitude, exprimée en décibels, nécessaire pour que la hauteur soit détectée. Une fois démarré, continue jusqu'à une diminution de 6 dB.

*ifrqs* (facultatif) -- nombre de divisions de l'octave. Vaut 12 par défaut et est limité à 120.

*iconf* (facultatif) -- nombre de conformations nécessaires pour un saut d'octave. Vaut 10 par défaut.

*istrtr* (facultatif) -- hauteur initiale pour le détecteur. La valeur par défaut est  $(ilo + ihi)/2$ .

*iocts* (facultatif) -- nombre de décimations d'octave dans le spectre. Vaut 6 par défaut.

*iq* (facultatif) -- Q des filtres d'analyse. Vaut 10 par défaut.

*inptls* (facultatif) -- nombre d'harmoniques utilisés pour la concordance. Le temps de calcul augmente avec le nombre d'harmoniques. Vaut 4 par défaut.

*irolloff* (facultatif) -- roll-off d'amplitude pour l'ensemble de filtres exprimé en fraction par octave. Les valeurs doivent être positives. Vaut 0.6 par défaut.

*iskip* (facultatif) -- s'il est non nul, l'initialisation est ignorée. Vaut 0 par défaut.

## Exécution

*koct* -- La sortie de hauteur, donnée dans le format octave point décimal.

*kamp* -- La sortie d'amplitude.

*pitch* analyse le signal d'entrée, *asig*, pour donner en sortie une paire hauteur/amplitude pour la fréquence la plus forte dans le signal. La valeur est mise à jour toutes les *iupdte* secondes.

Le nombre d'harmoniques et la fraction de roll-off pouvant affecter la détection de hauteur, il peut être nécessaire d'expérimenter. Les valeurs suggérées vont de 4 à 5 harmoniques avec un roll-off de 0.6 jusqu'à 10 à 12 harmoniques avec un roll-off de 0.75 pour les timbres complexes ayant un fondamental faible.

## Exemples

Voici un exemple de l'opcode pitch. Il utilise le fichier *pitch.csd* [examples/pitch.csd].

### Exemple 713. Exemple de l'opcode pitch.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if real audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pitch.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;clean audio

asig soundin p4
    outs asig, asig
endin

instr 2 ;use pitch

iupdt = 0.001 ;high definition
ilo = 6
ihi = 10
idbthresh = 10
ifrqs = 12
iconf = 10
istrt = 8

asig soundin p4
koct, kamp pitch asig, iupdt, ilo, ihi, idbthresh, ifrqs, iconf, istrt
kamp = kamp*.00005 ;lower volume
kcps = cpsoct(koct)
asig poscil kamp, kcps, 1 ;re-synthesize with sawtooth
    outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 16384 10 1 0.5 0.3 0.25 0.2 0.167 0.14 0.125 .111 ;sawtooth

i 1 0 3 "fox.wav"
i 2 3 3 "fox.wav"
i 1 6 3 "mary.wav"
i 2 9 3 "mary.wav"
i 1 12 3 "beats.wav"
i 2 15 3 "beats.wav"
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch  
Université de Bath, Codemist Ltd.  
Bath, UK  
Avril 1999

Nouveau dans la version 3.54 de Csound.

# pitchamdf

pitchamdf — Suit la hauteur d'un signal en se basant sur la méthode AMDF.

## Description

Suit la hauteur d'un signal en se basant sur la méthode AMDF (fonction de différence de grandeur moyenne). Retourne les signaux de hauteur et d'amplitude détectées. La méthode est assez rapide et devrait fonctionner en temps réel. Cette technique est habituellement plus efficace avec des signaux monophoniques.

## Syntaxe

```
kcps, krms pitchamdf asig, imincps, imaxcps [, icps] [, imedi] \
    [, idowns] [, iexcps] [, irmsmedi]
```

## Initialisation

*imincps* -- estimation de la fréquence minimale (en Hz) présente dans le signal.

*imaxcps* -- estimation de la fréquence maximale présente dans le signal.

*icps* (facultatif, 0 par défaut) -- estimation de la fréquence initiale du signal. Si elle vaut 0, *icps* = (*imincps*+*imaxcps*) / 2. La valeur par défaut est 0.

*imedi* (facultatif, 1 par défaut) -- taille du filtre médian appliqué à la sortie *kcps*. La taille du filtre sera *imedi*\*2+1. Si elle vaut 0, aucun filtre médian n'est appliqué. La valeur par défaut est 1.

*idowns* (facultatif, 1 par défaut) -- facteur de sous-échantillonnage pour *asig*. Doit être un entier. Un facteur *idowns* > 1 donne une exécution plus rapide, mais au risque d'une détection de hauteur moins bonne. L'intervalle utile est 1 - 4. La valeur par défaut est 1.

*iexcps* (facultatif, 0 par défaut) -- fréquence, en Hz, d'exécution de l'analyse de hauteur. Si elle vaut 0, *iexcps* est fixé à *imincps*. C'est habituellement raisonnable, mais l'expérimentation avec d'autres valeurs peut conduire à de meilleurs résultats. Vaut 0 par défaut.

*irmsmedi* (facultatif, 0 par défaut) -- taille du filtre médian appliqué à la sortie *krms*. La taille du filtre sera *irmsmedi*\*2+1. Si elle vaut 0, aucun filtre médian n'est appliqué. La valeur par défaut est 0.

## Exécution

*kcps* -- sortie de la hauteur détectée

*krms* -- sortie de l'amplitude détectée

Habituellement, *pitchamdf* fonctionne mieux avec des signaux monophoniques et il est assez fiable si des valeurs initiales appropriées sont choisies. En donnant à *imincps* et à *imaxcps* des valeurs aussi proches que possible que celles de la hauteur du signal, on obtient une meilleure détection et de meilleurs résultats.

Parce que le processus ne peut détecter la hauteur qu'après un délai initial, l'affectation à *icps* d'une valeur proche de la hauteur initiale réelle du signal protège des données erronées du début.

Le filtre médian évite les sauts de *kcps*. Expérimentez pour déterminer la valeur optimale de *imedi* pour un signal donné.

Les autres valeurs initiales peuvent habituellement prendre leurs valeurs par défaut. Passer *asig* dans un filtre passe-bas avant *pitchamdf* peut améliorer les résultats, en particulier avec des formes d'onde complexes.

## Exemples

Voici un exemple de l'opcode *pitchamdf*. Il utilise le fichier *pitchamdf.csd* [examples/pitchamdf.csd].

### Exemple 714. Exemple de l'opcode *pitchamdf*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if real audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pitchamdf.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;clean audio

asig soundin p4
outs asig, asig
endin

instr 2 ;use pitch

asig soundin p4
asig tone asig, 1000 ;lowpass-filter
kcps, krms pitchamdf asig, 100, 500, 200
asig poscil krms, kcps, 1 ;re-synthesize with sawtooth
outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 16384 10 1 0.5 0.3 0.25 0.2 0.167 0.14 0.125 .111 ;sawtooth

i 1 0 3 "fox.wav"
i 2 3 3 "fox.wav"
i 1 6 3 "mary.wav"
i 2 9 3 "mary.wav"
i 1 12 3 "beats.wav"
i 2 15 3 "beats.wav"
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Peter Neubäcker  
Munich, Allemagne

Août 1999

Nouveau dans la version 3.59 de Csound.

# planet

`planet` — Simulation d'un planète en orbite dans un système d'étoile binaire.

## Description

*planet* simule l'orbite d'une planète dans un système d'étoile binaire. Les sorties sont les coordonnées x, y et z de la planète en orbite. Il est possible que la planète atteigne sa vitesse de libération si elle croise une étoile de très près. Cela rend le système quelque peu instable.

## Syntaxe

```
ax, ay, az planet kmass1, kmass2, ksep, ix, iy, iz, ivx, ivy, ivz, idelta \  
[, ifriction] [, iskip]
```

## Initialisation

*ix, iy, iz* -- les coordonnées initiales x, y et z de la planète

*ivx, ivy, ivz* -- les composantes initiales du vecteur vitesse de la planète.

*idelta* -- la taille du pas utilisé dans l'approximation de l'équation différentielle.

*ifriction* (facultatif, 0 par défaut) -- une valeur de frottement que l'on peut utiliser pour empêcher le système de diverger

*iskip* (facultatif, 0 par défaut) -- s'il est non nul, l'initialisation du filtre est ignorée. (Nouveau dans les versions 4.23f13 et 5.0 de Csound)

## Exécution

*ax, ay, az* -- les coordonnées x, y et z de la planète en sortie

*kmass1* -- la masse de la première étoile

*kmass2* -- la masse de la seconde étoile

## Exemples

Voici un exemple de l'opcode *planet*. Il utilise le fichier *planet.csd* [examples/planet.csd].

### Exemple 715. Exemple de l'opcode *planet*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
; Audio out      Audio in      No messages  
-odac            -iadc         -d      ;;RT audio I/O  
; For Non-realtime ouput leave only the line below:  
; -o planet.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>
```

```

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 2

; Instrument #1 - a planet orbiting in 3D space.
instr 1
  ; Create a basic tone.
  kamp init 5000
  kcps init 440
  ifn = 1
  asnd oscil kamp, kcps, ifn

  ; Figure out its X, Y, Z coordinates.
  km1 init 0.5
  km2 init 0.35
  ksep init 2.2
  ix = 0
  iy = 0.1
  iz = 0
  ivx = 0.5
  ivy = 0
  ivz = 0
  ih = 0.0003
  ifric = -0.1
  ax1, ay1, az1 planet km1, km2, ksep, ix, iy, iz, \
    ivx, ivy, ivz, ih, ifric

  ; Place the basic tone within 3D space.
  kx downsamp ax1
  ky downsamp ay1
  kz downsamp az1
  idist = 1
  ift = 0
  imode = 1
  imdel = 1.018853416
  iovr = 2
  aw2, ax2, ay2, az2 spat3d asnd, kx, ky, kz, idist, \
    ift, imode, imdel, iovr

  ; Convert the 3D sound to stereo.
  aleft = aw2 + ay2
  aright = aw2 - ay2

  outs aleft, aright
endin

</CsInstruments>
<CsScore>

; Table #1 a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 10 seconds.
i 1 0 10
e

</CsScore>
</CsoundSynthesizer>

```



## Voir aussi

Plus d'information sur cet opcode : <http://www.csoundjournal.com/issue9/FlutesInOrbit.html> [http://www.csoundjournal.com/issue9/FlutesInOrbit.html], écrit par Brian Redfern.

## Crédits

Auteur : Hans Mikelson  
Décembre 1998

Nouveau dans la version 3.50 de Csound

# platerrev

platerrev — Modélise la réverbération d'une plaque métallique.

## Description

Modélise la réverbération d'une plaque métallique rectangulaire avec des caractéristiques physiques ajustables lorsqu'elle est excitée par des signaux audios.

## Syntaxe

```
al[, a2, ...] platerrev itabexcite. itabouts, kbndry, iaspect, istiff, idecay, iloss, aexcitel[, aexcite]
```

## Initialisation

*itabexcite* -- numéro d'une table contenant un triplet pour chaque signal d'excitation (fréquence, rayon, phase initiale en radians). Le rayon doit être inférieur à 1. Ceux-ci contrôlent l'endroit où l'excitation se produit. Les valeurs dans la table pour la fréquence et le rayon peuvent être changée durant l'exécution avec le risque de clics si les changements sont trop importants.

*itabouts* -- numéro d'une table contenant un triplet pour chaque signal de sortie (fréquence, rayon, phase initiale en radians). Voir la description de *itabexcite*.

*kbndry* -- conditions aux limites de la plaque ; 0 = libre, 1 = fixée, 2 = pivotante. Les autres valeurs sont indéfinies. Ce paramètre peut être changé au taux-k, au risque de l'apparition de clics.

*iaspect* -- rapport d'aspect de la plaque qui doit être inférieur ou égal à 1.

*istiff* -- paramètre de raideur de la plaque (fixé autour de 1 ou à une valeur inférieure pour une réverbération de plaque).

*idecay* -- temps de décroissance à 30 db.

*iloss* -- paramètre de perte des hautes fréquences (une valeur d'environ 0.001 est recommandée).

## Exécution

Une plaque métallique est excitée par un signal stéréo et la réverbération résultante est prélevée.

*aexciten* -- signaux d'excitation à injecter dans la plaque.

## Exemples

Voici un exemple de l'opcode *platerrev*. Il utilise le fichier *plate.csd* [examples/plate.csd].

### Exemple 716. Exemple de l'opcode *platerrev*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
```

```

<CsOptions>
; Select audio/midi flags here according to platform
;-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
-o plate.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
; Note: this example is heavy on CPU
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ab      diskin2  "beats.wav", 1
al,ar   platerev 1, 2, p4, 0.73, 1.0, 5.0, 0.001, ab,ab
outs    al*.25,ar*.25

endin
</CsInstruments>
<CsScore>
f1 0 8 -2  0.3  0.3875  0.39274  0.32  0.85714 0.78548
f2 0 8 -2  0.2  0.666667 1.57097  0.24  0.75    0.78548
i1 0 4 1
i1 + 4 2
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Stefan Bilbao  
 Université d'Edimbourg, UK  
 Auteur : John ffitch  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 5.17.12 de Csound.

# plltrack

plltrack — Traque la hauteur d'un signal.

## Description

*plltrack* est un détecteur de hauteur basé sur un algorithme de boucle à verrouillage de phase, décrit par Zolzer, U, Sankarababu, S.V. et Moller, S, "PLL-based Pitch Detection and Tracking for Audio Signals. Proc. of IHH-MSP 2012".

## Syntaxe

```
acps, alock plltrackasig, kd [, kloopf, kloopq, klf, khf, kthresh]
```

## Exécution

*acps* -- hauteur estimée en Hz.

*alock* -- indicateur du verrouillage de phase, une erreur de phase indiquant la qualité de la détection, avec des valeurs comprises entre 0 et 1. Les plus grandes valeurs dénotent une bonne détection.

*kd* -- Gain de réaction de la PLL. Contrôle l'intervalle de fréquence de la PLL (entre 0 et 1). Les plus grandes valeurs augmentent l'intervalle de détection.

*kloopf* -- Fréquence de coupure du filtre passe-bas de la PLL. Contrôle l'intervalle de fréquence de la PLL (facultatif, vaut 20 Hz par défaut).

*kloopq* -- Q du filtre passe-bas de la PLL. Contrôle le temps de montée vers la fréquence centrale (facultatif, vaut 1/3 par défaut).

*klf* -- fréquence de détection la plus basse (facultatif, vaut 20 Hz par défaut)

*khf* -- fréquence de détection la plus haute (facultatif, vaut 1500 Hz par défaut)

*kthresh* -- seuil de niveau du signal de détection (facultatif, vaut 0.001 par défaut, ce qui équivaut à -60 dBfs)

*plltrack* analyse le signal d'entrée *asig*, estimant la fondamentale d'un signal monophonique. Sa sortie est actualisée à chaque échantillon.

## Exemples

Voici un exemple de l'opcode *plltrack*. Il utilise le fichier *plltrack.csd* [examples/plltrack.csd].

### Exemple 717. Exemple de l'opcode *plltrack*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform
```

```

-odac      ;;;realtime audio out
;-iadc      ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o plltrack.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kd = p4
a1 diskin2 "fox.wav", 1,0,1
apitch, aloc plltrack a1, kd
krms rms a1
krms port krms, 0.01
asig buzz krms, apitch, 10, 1
      outs asig, asig ;mix in some dry signal as well

endin
</CsInstruments>
<CsScore>
f1 0 65536 10 1 ;sine wave

i 1 0 6 0.1
i 1 7 6 0.3 ;more feedback

e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Victor Lazzarini  
 NUI, Maynooth.  
 Maynooth, Ireland  
 Septembre, 2012

Nouveau dans la version 5.18.031 de Csound.

# pluck

pluck — Produit un son de corde pincée à décroissance naturelle ou un son de tambour.

## Description

La sortie audio est un son de corde pincée à décroissance naturelle ou un son de tambour basés sur l'algorithme de Karplus-Strong.

## Syntaxe

```
ares pluck kamp, kcps, icps, ifn, imeth [, iparm1] [, iparm2]
```

## Initialisation

*icps* -- valeur de hauteur attendue en Hz, utilisée pour fixer un tampon contenant une période d'échantillons audio qui sera lissée progressivement par une méthode de décroissance choisie. *icps* anticipe normalement la valeur de *kcps*, mais il peut recevoir artificiellement une grande ou une petite valeur pour influencer la taille du tampon d'échantillons.

*ifn* -- numéro de la table d'une fonction utilisée pour initialiser le tampon de décroissance cyclique. Si *ifn* = 0, une séquence aléatoire sera utilisée à la place.

*imeth* -- méthode de décroissance naturelle. Il y en a six, dont certaines utilisent les paramètres qui suivent.

1. moyenne simple. Un procédé de lissage simple, sans paramètres.
2. moyenne variable. Comme ci-dessus, avec une durée de lissage étirée d'un facteur de *iparm1* ( $\geq 1$ ).
3. simple tambour. L'intervalle allant de la hauteur au bruit est contrôlé par un "facteur de rugosité" dans *iparm1* (0 à 1). Zéro donne l'effet de corde pincée, tandis que 1 inverse la polarité de chaque échantillon (baisse d'une octave, harmoniques impairs). La valeur 0.5 donne un son de caisse claire optimal.
4. tambour variable. Combine les facteurs de rugosité et d'étirement. *iparm1* est la rugosité (0 à 1), et *iparm2* est le facteur d'étirement ( $\geq 1$ ).
5. moyenne pondérée. Comme la méthode 1, avec *iparm1* pondérant l'échantillon courant (le status quo) et *iparm2* pondérant l'échantillon précédant. *iparm1* + *iparm2* doit être  $\leq 1$ .
6. filtre récursif du premier ordre, avec des coefficients de 0.5. N'est pas affecté par les paramètres.

*iparm1*, *iparm2* (facultatif) -- valeurs des paramètres à utiliser avec les algorithmes de lissage (ci-dessus). Les valeurs par défaut sont 0.

## Exécution

*kamp* -- l'amplitude de sortie.

*kcps* -- la fréquence de re-échantillonnage en Hz.

Un tampon audio interne, rempli lors de l'initialisation selon *ifn*, est continuellement re-échantillonné avec une fréquence de *kcps* et sa sortie est multipliée par *kamp*. Le re-échantillonnage du tampon est complété par un lissage pour simuler l'effet de décroissance naturelle du son.

Les cordes pincées (1, 2, 5, 6) sont plus réalistes si l'on commence avec une source de bruit, qui est riche en harmoniques initiaux. Les sons de tambour (méthodes 3 et 4) fonctionnent mieux avec une source plate (impulsion large), qui produit une attaque très bruiteuse et une extinction rapide.

L'algorithme original de Karplus-Strong utilisait un nombre fixe d'échantillons par cycle, ce qui provoquait une sérieuse quantification des hauteurs disponibles et de leur intonation. Cette implémentation re-échantillonne un tampon à la hauteur exacte donnée par *kcps*, qui peut être variée pour des effets de vibrato ou de glissando. Avec de faibles valeurs du taux d'échantillonnage de l'orchestre (par exemple *sr* = 10000), les fréquences élevées ne stockeront que très peu d'échantillons (*sr* / *icps*). Comme ceci peut causer un bruit notable lors du re-échantillonnage, le tampon interne a une taille minimale de 64 échantillons. Celui-ci peut être agrandi en fixant *icps* à une hauteur artificiellement basse.

## Exemples

Voici un exemple de l'opcode *pluck*. Il utilise le fichier *pluck.csd* [examples/pluck.csd].

### Exemple 718. Exemple de l'opcode *pluck*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if real audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pluck.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kcps = 220
icps = 220
ifn = 0
imeth = p4

asig pluck 0.7, 220, 220, ifn, imeth, .1, 10
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 5 1
i 1 5 5 4 ;needs 2 extra parameters (iparm1, iparm2)
i 1 10 5 6
e
</CsScore>
</CsoundSynthesizer>
```

# poisson

poisson — Générateur de nombres aléatoires de distribution de Poisson (valeurs positives seulement).

## Description

Générateur de nombres aléatoires de distribution de Poisson (valeurs positives seulement). C'est un générateur de bruit de classe x.

## Syntaxe

`ares poisson klambda`

`ires poisson klambda`

`kres poisson klambda`

## Exécution

*ares*, *kres*, *ires* - nombre d'évènements se produisant (toujours un entier).

*klambda* - le nombre attendu d'évènements par intervalle d'échantillonnage.

## Adapté de Wikipédia :

En théorie des probabilités et en statistiques, la distribution de Poisson est une distribution de probabilité discrète. Elle exprime la probabilité d'apparition d'un certain nombre d'évènements pendant une période de temps fixée si ces évènements se produisent avec un taux moyen connu et indépendamment du temps écoulé depuis le dernier évènement.

La distribution de Poisson décrivant la probabilité qu'il y ait exactement  $k$  évènements ( $k$  étant un nombre non négatif,  $k = 0, 1, 2, \dots$ ) est :

$$f(k; \lambda) = \frac{e^{-\lambda} \lambda^k}{k!},$$

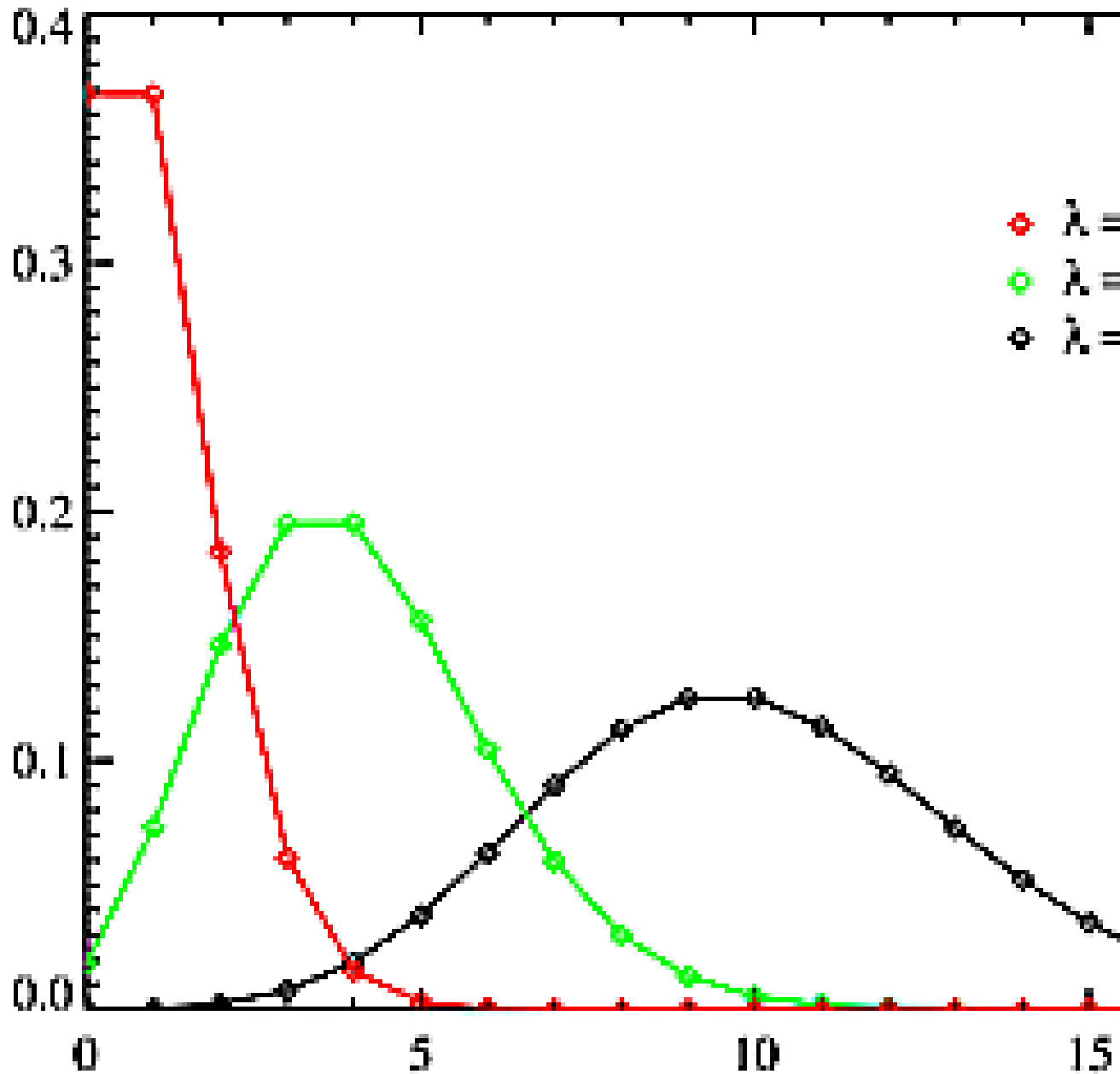
où :

- $\lambda$  est un nombre réel positif, égal au nombre attendu d'évènements se produisant durant l'intervalle donné. Par exemple, si les évènements se produisent en moyenne toutes les 4 minutes, et que l'on est intéressé par le nombre d'évènements se produisant dans un intervalle de 10 minutes, on utilisera comme modèle une distribution de Poisson avec  $\lambda = 10/4 = 2,5$ . Ce paramètre se nomme *klambda* dans les opcodes *poisson*.
- $k$  fait référence au nombre de i-, k- ou a- périodes écoulées.

La distribution de Poisson apparaît aussi avec les processus de Poisson. Elle s'applique à différents phénomènes de nature discrète (c-à-d, ceux qui peuvent se produire 0, 1, 2, 3, ... fois durant une période de temps donnée ou dans un espace donné) chaque fois que la probabilité du phénomène se produisant est constante dans le temps ou dans l'espace. Parmi les exemples qui peuvent être modélisés par une distribution de Poisson, on trouve :



- Le nombre d'automobiles passant devant un repère sur une route (suffisamment éloigné des feux de circulation) pendant un intervalle de temps donné.
- Le nombre de fautes de frappe que l'on fait lorsque l'on tape une page.
- Le nombre d'appels par minute dans un centre d'appel téléphonique.
- Le nombre d'accès par minute à un serveur web.
- Le nombre d'animaux écrasés par unité de longueur sur une route.
- Le nombre de mutations dans un brin d'ADN après une certaine quantité de radiations.
- Le nombre de noyaux instables qui a diminué pendant une période de temps donnée dans un morceau de substance radioactive. Comme la radioactivité de la substance diminue avec le temps, l'intervalle de temps total utilisé dans le modèle doit être significativement inférieur à la durée de vie moyenne de la substance.
- Le nombre de pins par unité de surface dans une forêt hétérogène.
- Le nombre d'étoiles dans une région donnée de l'espace.
- La distribution des cellules réceptrices de la vision dans la rétine de l'oeil humain.
- Le nombre de virus qui peuvent infecter une cellule dans une culture de cellules.



Un diagramme montrant la distribution de Poisson.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode poisson. Il utilise le fichier *poisson.csd* [examples/poisson.csd]. Il est écrit pour des systèmes \*NIX et générera des erreurs sur Windows.

## Exemple 719. Exemple de l'opcode poisson.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o poisson.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 441 ;ksmps set deliberately high to have few k-periods per second
nchnls = 1

; Instrument #1.
instr 1
; Generates a random number in a poisson distribution.
; klambda = 1

i1 poisson 1

print i1
endin

instr 2

kres poisson p4
printk (ksmps/sr),kres ;prints every k-period
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
i 2 1 0.2 0.5
i 2 2 0.2 4 ;average 4 events per k-period
i 2 3 0.2 20 ;average 20 events per k-period
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*seed, betarand, bexpnd, cauchy, expand, gauss, linrand, pcauchy, trirand, unirand, weibull*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

# pol2rect

pol2rect — Conversion du format polaire au format rectangulaire.

## Description

Convertit un tableau d'entrée du format module-argument au format réel-imaginaire.

## Syntaxe

```
kout[] pol2rect kin[]  
kout[] pol2rect kmags[], kphs[]
```

## Exécution

*kout[]* -- tableau contenant les valeurs complexes réelles-imaginaires de sortie. Créé s'il n'existe pas.

*kin[]* -- tableau contenant les valeurs complexes modules-arguments d'entrée.

*kmags[]* -- tableau contenant les valeurs réelles des modules en entrée.

*kphs[]* -- tableau contenant les valeurs réelles des arguments en entrée.

## Exemples

Voici un exemple de l'opcode pol2rect. Il utilise le fichier *pol2rect.csd* [exemples/pol2rect.csd].

### Exemple 720. Exemple de l'opcode pol2rect.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
-d -o dac  
</CsOptions>  
<CsInstruments>  
/* ksmpts needs to be an integer div of  
   hopsize */  
ksmps = 64  
  
instr 1  
  
   ihopsize = 256 ; hopsize  
   ifftsize = 1024 ; FFT size  
   iolaps = ifftsize/ihopsize ; overlaps  
   ibw = sr/ifftsize ; bin bandwidth  
   kcnt init 0 ; counting vars  
   krow init 0  
  
   k0la[] init ifftsize ; overlap-add buffer  
   kIn[] init ifftsize ; input buffer  
   kOut[][] init iolaps, ifftsize ; output buffers  
  
   a1 diskin2 "fox.wav",1,0,1 ; audio input
```

```
/* every hopsize samples */
if kcnt == ihopsize then
  /* window and take FFT */
  kWin[] window kIn,krow*ihopsize
  kSpec[] rfft kWin

  kSpec rect2pol kSpec

  /* reduce mags between high and low freqs */
  ilow = 0
  ihigh = 1000
  ki = int(ilow/ibw)
  until ki >= int(ihigh/ibw) do
    kSpec[ki] = kSpec[ki]*0.1
    ki += 2
  od

  kSpec pol2rect kSpec

  /* IFFT + window */
  kRow[] rfft kSpec
  kWin window kRow, krow*ihopsize
  /* place it on out buffer */
  kOut setrow kWin, krow

  /* zero the ola buffer */
  kOla = 0
  /* overlap-add */
  ki = 0
  until ki == iolaps do
    kRow getrow kOut, ki
    kOla = kOla + kRow
    ki += 1
  od

  /* update counters */
  krow = (krow+1)%iolaps
  kcnt = 0
endif

/* shift audio in/out of buffers */
kIn shiftin a1
a2 shiftout kOla
out a2/iolaps

/* increment counter */
kcnt += ksmpls

endin

</CsInstruments>
<CsScore>
i1 0 10
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux.

## Crédits

Auteur : Victor Lazzarini  
NUI Maynooth

2014

Nouveau dans la version 6.04

# polyaft

polyaft — Retourne la pression d'after-touch polyphonique du numéro de note sélectionné.

## Description

*polyaft* retourne la pression polyphonique du numéro de note choisi, optionnellement mappé dans un intervalle défini par l'utilisateur.

## Syntaxe

```
ires polyaft inote [, ilow] [, ihigh]
```

```
kres polyaft inote [, ilow] [, ihigh]
```

## Initialisation

*inote* -- numéro de note. Normalement ajusté à la valeur retournée par *notnum*

*ilow* (facultatif, par défaut : 0) -- la valeur de sortie la plus basse

*ihigh* (facultatif, par défaut : 127) -- la valeur de sortie la plus haute

## Exécution

*kres* -- Pression polyphonique (aftertouch).

## Exemples

Voici un exemple de l'opcode *polyaft*. Il utilise le fichier *polyaft.csd* [examples/polyaft.csd].

Ne pas oublier d'inclure l'option *-F* lorsque l'on utilise un fichier MIDI externe comme « *polyaft.mid* ».

### Exemple 721. Exemple de l'opcode *polyaft*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages MIDI in
-odac        -iadc      -d          -M0   ;;RT audio I/O with MIDI in
; For Non-realtime ouput leave only the line below:
; -o polyaft.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 10
nchnls = 1

massign 1, 1
itmp ftgen 1, 0, 1024, 10, 1 ; sine wave
```

```
instr 1

kcps cpsmidib 2 ; note frequency
inote notnum ; note number
kaft polyaft inote, 0, 127 ; aftertouch
; interpolate aftertouch to eliminate clicks
ktmp phasor 40
ktmp trigger 1 - ktmp, 0.5, 0
kaft tlineto kaft, 0.025, ktmp
; map to sine curve for crossfade
kaft = sin(kaft * 3.14159 / 254) * 22000

asnd oscili kaft, kcps, 1

out asnd

endin

</CsInstruments>
<CsScore>

t 0 120
f 0 9 2 -2 0
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Ajouté grâce à un courriel de Istvan Varga

Nouveau dans la version 4.12



# polynomial

polynomial — Evalue efficacement un polynôme d'ordre arbitraire.

## Description

L'opcode *polynomial* calcule un polynôme à une seule variable d'entrée de taux-a. Le polynôme est la somme de n'importe quel nombre de termes de la forme  $kn * x^n$  où  $kn$  est le nième coefficient de l'expression. Ces coefficients sont des valeurs de taux-k.

## Syntaxe

```
out polynomial ain, k0 [, k1 [, k2 [...]]]
```

## Exécution

*ain* -- le signal d'entrée jouant le rôle de la variable indépendante du polynôme ("x").

*out* -- le signal de sortie ("y").

*k0, k1, k2, ...* -- les coefficients pour chaque terme du polynôme.

Si l'on considère que le paramètre d'entrée *ain* est "x" et que la sortie *out* est "y", alors l'opcode *polynomial* calcule l'équation suivante :

$$y = k0 + k1 * x + k2 * x^2 + k3 * x^3 + \dots$$

## Voir aussi

*chebyshevpoly, mac maca*

## Exemples

Voici un exemple de l'opcode polynomial. Il utilise le fichier *polynomial.csd* [examples/polynomial.csd].

### Exemple 722. Exemple de l'opcode polynomial.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime output leave only the line below:
; -o polynomial.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
```

```

; The polynomial y=x^n where n is odd always produces a curve
; that traverses the range [-1, 1] when the input is within
; the same range. Therefore, we can use one of these curves
; to make a nonlinear phasor that repeatedly reads a table
; from beginning to end like a linear phasor (maintaining
; continuity) but that distorts the waveform in the table.

instr 4 ; This instrument demonstrates phase distortion with x^3

idur   = p3
iamp   = p4
ifreq  = p5
itable = p6

aenv linseg 0, .001, 1.0, idur - .051, 1.0, .05, 0 ; declicking envelope
aosc phasor ifreq ; create a linear phasor
apd polynomial aosc, 0, 0, 0, 1 ; distort the phasor with x^3
aout tablei apd, itable, 1 ; read a sine table with the nonlinear phasor
outs aenv*aout*iamp, aenv*aout*iamp

endin

instr 5 ; This instrument demonstrates phase distortion with x^11

idur   = p3
iamp   = p4
ifreq  = p5
itable = p6

aenv linseg 0, .001, 1.0, idur - .051, 1.0, .05, 0 ; declicking envelope
aosc phasor ifreq ; create a linear phasor
apd polynomial aosc, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 ; distort the phasor with x^11
aout tablei apd, itable, 1 ; read a sine table with the nonlinear phasor
outs aenv*aout*iamp, aenv*aout*iamp

endin

instr 6 ; This instrument crossfades between a pure sine and one distorted with x^11

idur   = p3
iamp   = p4
ifreq  = p5
itable = p6

aenv linseg 0, .001, 1.0, idur - .051, 1.0, .05, 0 ; declicking envelope
aosc phasor ifreq ; create a linear phasor
aout3 tablei aosc, itable, 1 ; read a sine table without the linear phasor
apd11 polynomial aosc, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 ; distort the phasor with x^11
aout11 tablei apd11, itable, 1 ; read a sine table with the nonlinear phasor
kamount linseg 1.0, 0.05, 0.9, 1.0, 0.0 ; crossfade between two outputs
aout = aout3*kamount + aout11*(1.0 - kamount)
outs aenv*aout*iamp, aenv*aout*iamp

endin
</CsInstruments>
<CsScore>
f1 0 16385 10 1 ; sine wave

; descending "just blues" scale

t 0 100
i4 0 .333 .7 512 1
i. + . . 448
i. + . . 384
i. + . . 360
i. + . . 341.33
i. + . . 298.67

```

```

i. + 2      . 256
s

t 0 100
i5 0 .333 .7 512      1
i. + .      . 448
i. + .      . 384
i. + .      . 360
i. + .      . 341.33
i. + .      . 298.67
i. + 2      . 256
s

t 0 100
i6 0 .333 .7 512      1
i. + .      . 448
i. + .      . 384
i. + .      . 360
i. + .      . 341.33
i. + .      . 298.67
i. + 2      . 256

e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*chebyshevpoly, mac maca*

## Crédits

Auteur : Anthony Kozar  
Janvier 2008

Nouveau dans la version 5.08 de Csound.

# port

`port` — Applique un portamento à un signal de contrôle en escalier.

## Description

Applique un portamento à un signal de contrôle en escalier.

## Syntaxe

```
kres port ksig, ihtim [, isig]
```

## Initialisation

*ihitim* -- durée à mi-parcours de la fonction, en secondes.

*isig* (facultatif, par défaut 0) -- valeur initiale (c-à-d. précédente) pour la rétroaction interne. La valeur par défaut est 0. Avec une valeur négative l'initialisation sera ignorée et la dernière valeur de l'instance précédente sera la valeur initiale de la note.

## Exécution

*kres* -- le signal de sortie au taux de contrôle.

*ksig* -- le signal d'entrée au taux de contrôle.

`port` applique un portamento à un signal de contrôle en escalier. A chaque nouveau palier, *ksig* est filtré par un filtre passe-bas pour que la transition vers cette valeur se fasse au taux déterminé par *ihitim*. *ihitim* est la durée à « mi-parcours » de la fonction (en secondes), au cours de laquelle la courbe parcourera la moitié de la distance la séparant de la nouvelle valeur, puis la moitié de la moitié, etc., n'atteignant théoriquement jamais son asymptote. Avec *portk*, la durée à mi-parcours peut être variée au taux de contrôle.

## Exemples

Voici un exemple de l'opcode `port`. Il utilise le fichier *port.csd* [examples/port.csd].

### Exemple 723. Exemple de l'opcode `port`.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o port.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
```

```
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1

aout diskin2 "fox.wav",1, 0, 1
kf,ka ptrack aout, 512 ; pitch track with winsize=1024
kcps port kf, 0.01 ; smooth freq
kamp port ka, 0.01 ; smooth amp
; drive an oscillator
asig poscil ampdB(kamp)*0dbfs, kcps, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
; simple sine wave
f 1 0 4096 10 1

i 1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

## Voir aussi

*areson, aresonk, atone, atonek, portk, reson, resonk, tone, tonek*

# portk

portk — Applique un portamento à un signal de contrôle en escalier.

## Description

Applique un portamento à un signal de contrôle en escalier.

## Syntaxe

```
kres portk ksig, khtim [, isig]
```

## Initialisation

*isig* (facultatif, par défaut 0) -- valeur initiale (c-à-d. précédente) pour la rétroaction interne. La valeur par défaut est 0.

## Exécution

*kres* -- le signal de sortie au taux de contrôle.

*ksig* -- le signal d'entrée au taux de contrôle.

*khtim* -- durée à mi-parcours de la fonction, en secondes.

*portk* est semblable à *port* à part le fait que la durée à mi-parcours peut-être variée au taux de contrôle.

## Exemples

Voici un exemple de l'opcode portk. Il utilise le fichier *portk.csd* [exemples/portk.csd].

### Exemple 724. Exemple de l'opcode portk.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      ; -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o portk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 1

;Example by Andres Cabrera 2007

FLpanel "Slider", 650, 140, 50, 50
gkvall, gislider1 FLslider "Watch me", 0, 127, 0, 5, -1, 580, 30, 25, 20
```

```
gkval2, gislider2 FLslider "Move me", 0, 127, 0, 5, -1, 580, 30, 25, 80
gkhtim, gislider3 FLslider "khtim", 0.1, 1, 0, 6, -1, 30, 100, 610, 10
FLpanelEnd
FLrun

FLsetVal_i 0.1, gislider3 ;set initial time to 0.1

instr 1
kval portk gkval2, gkhtim ; take the value of slider 2 and apply portamento
FLsetVal 1, kval, gislider1 ;set the value of slider 1 to kval
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one minute.
i 1 0 60
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*areson, aresonk, atone, atonek, port, reson, resonk, tone, tonek*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

# poscil3

poscil3 — Oscillateur haute précision avec interpolation cubique.

## Description

Oscillateur haute précision avec interpolation cubique.

## Syntaxe

```
ares poscil3 aamp, acps [, ifn, iphs]
ares poscil3 aamp, kcps [, ifn, iphs]
ares poscil3 kamp, acps [, ifn, iphs]
ares poscil3 kamp, kcps [, ifn, iphs]
ires poscil3 kamp, kcps [, ifn, iphs]
kres poscil3 kamp, kcps [, ifn, iphs]
```

## Initialisation

*ifn* (facultatif) -- numéro de la table de fonction. Vaut -1 par défaut ce qui indique une onde sinus.

*iphs* (facultatif, par défaut 0) -- phase initiale (table normalisée, index 0-1). Si une valeur négative est donnée, l'initialisation de la phase est ignorée.

## Exécution

*ares* -- signal de sortie

*kamp*, *aamp* -- amplitude du signal de sortie.

*kcps*, *acps* -- fréquence du signal de sortie en Hertz.

*poscil3* fonctionne comme *poscil*, mais il utilise l'interpolation cubique.

Noter que *poscil3* peut utiliser des tables de longueur différée (non puissance de deux).

## Exemples

Voici un exemple de l'opcode *poscil3*. Il utilise le fichier *poscil3.csd* [exemples/poscil3.csd].

### Exemple 725. Exemple de l'opcode poscil3.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;;realtime audio out
```



```

;-iadc    ;;uncomment -iadc if real audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o poscil3.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 2^10, 10, 1

instr 1

krnd randomh 40, 440, 1 ; produce random values
ain poscil3 .6, krnd, giSine
kline line 1, p3, 0 ; straight line
aL,aR pan2 ain, kline ; sent across image
outs aL, aR

endin
</CsInstruments>
<CsScore>
i1 0 10
e
</CsScore>
</CsSoundSynthesizer>

```

Voici un autre exemple de l'opcode poscil3, qui utilise une table remplie à partir d'un fichier son. Il utilise le fichier *poscil3-file.csd* [examples/poscil3-file.csd].

## Exemple 726. Un autre exemple de l'opcode poscil3.

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out Audio in No messages
-odac -iadc -d ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o poscil3-file.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Example written by Joachim Heintz 07/2008

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; non-normalized function table with a sample 1
giFile ftgen 1, 0, 0, -1, "fox.wav", 0, 0, 0

; Instrument #1 - uses poscil3 for playing samples from a function table
instr 1
kamp = p4
kspeed = p5
ifn = 1
iskip = p6
kcps = kspeed / (ftlen(ifn) / ftsr(ifn)); frequency of the oscillator
iphs = iskip / (ftlen(ifn) / ftsr(ifn)); calculates skiptime to phase values (0-1)

a1 poscil3 kamp, kcps, ifn, iphs
out a1

```

```
endin
</CsInstruments>
<CsScore>
i1 0 2.756 1 1 0
i1 3 2.756 1 -1 0
i1 6 1.378 1 .5 2.067
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*poscil*

## Crédits

Auteurs : John ffitich, Gabriel Maldonado  
Italie

Nouveau dans la version 3.52 de Csound

Les variantes avec fréquence ou amplitude de taux-a sont nouvelles dans la version 5.16

La table de fonction est facultative depuis la version 6.0 de Csound.

La possibilité d'ignorer l'initialisation a été ajoutée dans la version 6.06.

# poscil

poscil — Oscillateur haute précision.

## Description

Oscillateur haute précision.

## Syntaxe

```
ares poscil aamp, acps [, ifn, iphs]  
ares poscil aamp, kcps [, ifn, iphs]  
ares poscil kamp, acps [, ifn, iphs]  
ares poscil kamp, kcps [, ifn, iphs]  
ires poscil kamp, kcps [, ifn, iphs]  
kres poscil kamp, kcps [, ifn, iphs]
```

## Initialisation

*ifn* (facultatif) -- numéro de la table de fonction. Vaut -1 par défaut ce qui indique une onde sinus.

*iphs* (facultatif, par défaut 0) -- phase initiale (table normalisée, index 0-1). Si une valeur négative est donnée, l'initialisation de la phase est ignorée.

## Exécution

*ares* -- signal de sortie

*kamp*, *aamp* -- l'amplitude du signal de sortie.

*kcps*, *acps* -- la fréquence du signal de sortie en Hz.

*poscil* (oscillateur de précision) est identique à *oscili*, mais il permet un contrôle de la fréquence plus précis, en particulier lorsque l'on utilise de grandes tables avec de faibles valeurs de fréquence. Il utilise une indexation de la table en virgule flottante, au lieu de l'arithmétique entière utilisée par *oscil* et *oscili*. Il est à peine plus lent que *oscili*.

Depuis Csound 4.22, *poscil* accepte aussi des valeurs de fréquence négatives et il peut utiliser des valeurs de taux-a aussi bien pour l'amplitude que pour la fréquence. Ainsi, cet opcode permet la modulation d'amplitude (MA) et la modulation de fréquence (MF).

L'opcode *poscil3* est le même que *poscil*, mais il utilise une interpolation cubique.

Noter que *poscil* peut utiliser des tables de longueur différée (non puissance de deux).

## Exemples

Voici un exemple de l'opcode *poscil*. Il utilise le fichier *poscil.csd* [examples/poscil.csd].

## Exemple 727. Exemple de l'opcode poscil.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o poscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

seed 0
gisine ftgen 0, 0, 2^10, 10, 1

instr 1

ipeak random 0, 1 ;where is the envelope peak
asig poscil .8, 220, gisine
aenv transeg 0, p3*ipeak, 6, 1, p3-p3*ipeak, -6, 0
aL,aR pan2 asig*aenv, ipeak ;pan according to random value
outs aL, aR

endin

</CsInstruments>
<CsScore>
i1 0 5
i1 4 5
i1 8 5
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*poscil3*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1998

Novembre 2002. Ajout d'une note sur les changements dans la version 4.22 de Csound, merci à Rasmus Ekman.

Nouveau dans la version 3.52 de Csound.

La table de fonction est facultative depuis la version 6.0 de Csound.

La possibilité d'ignorer l'initialisation a été ajoutée dans la version 6.06.

# pow

`pow` — Calcule l'élévation à la puissance d'un argument par l'autre argument.

## Description

Calcule *xarg* élevé à la puissance *kpow* (ou *ipow*) et pondère le résultat par *inorm*.

## Syntaxe

```
ares pow aarg, kpow [, inorm]
ires pow iarg, ipow [, inorm]
kres pow karg, kpow [, inorm]
ires[] pow iarg[], ipow[]
kres[] pow karg[], kpow[]
ires[] pow iarg[], ipow
kres[] pow karg[], kpow
```

## Initialisation

*inorm* (facultatif, par défaut=1) -- Le nombre qui divisera le résultat (1 par défaut). Particulièrement utile si l'on calcule des puissances de signaux de taux -a ou de taux -k, ce qui produit très souvent des échantillons hors intervalle.

## Exécution

*aarg*, *iarg*, *karg* -- la base.

*ipow*, *kpow* -- l'exposant.



### Note

Utiliser ^ avec précaution dans les instructions arithmétiques, car les règles de précedence peuvent ne pas être correctes. Nouveau dans la version 3.493 de Csound.

## Exemples

Voici un exemple de l'opcode `pow`. Il utilise le fichier *pow.csd* [examples/pow.csd].

### Exemple 728. Exemple de l'opcode `pow`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
```

```

-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o pow.wav      ; output to audio file
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; Lo-Fi sound

kpow = 10      ;exponent
kbase line 1, p3, 1.4 ;vary the base
kQuantize = kQuantize*0.5 ;half the number of steps for each side of a bipolar signal
printk2 kQuantize
asig disk2 "fox.wav", 1, 0, 1 ;loop the fox
asig = round(asig * kQuantize) / kQuantize ;quantize and scale audio signal
outs asig, asig

endin
</CsInstruments>
<CsScore>

i1 0 19.2

e
</CsScore>
</CsoundSynthesizer>

```

La sortie contiendra des lignes comme celles-ci :

```

i1      0.50000
i1      0.50007
i1      0.50014
.....
i1     14.45986
i1     14.46130\r

```

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

# powershape

`powershape` — Distorsion non linéaire d'un signal par élévation à une puissance variable.

## Description

L'opcode *powershape* élève un signal d'entrée à une puissance avec pré- et post-ajustage du signal afin que la sortie soit dans un intervalle prévisible. Il traite également les entrées négatives de manière symétrique aux entrées positives, calculant une fonction de transfert dynamique utile pour la distorsion non-linéaire.

## Syntaxe

```
aout powershape ain, kShapeAmount [, ifullscale]
```

## Initialisation

*ifullscale* -- paramètre facultatif spécifiant l'intervalle des valeurs d'entrée entre *-ifullscale* et *ifullscale*. Vaut 1.0 par défaut. Il faut donner à ce paramètre la valeur maximale attendue en entrée.

## Exécution

*ain* -- le signal d'entrée à modeler.

*aout* -- le signal de sortie.

*kShapeAmount* -- l'importance de l'effet appliqué à l'entrée ; égal à la puissance à laquelle le signal d'entrée est élevé.

L'opcode *powershape* ressemble beaucoup aux générateurs unitaires *pow* lorsqu'il s'agit de calculer la "puissance" mathématique. Cependant, il introduit quelques particularités qui le rendent plus utile à la distorsion non-linéaire signaux de taux-audio. Le paramètre *kShapeAmount* est l'exposant de la puissance à laquelle le signal d'entrée est élevé.

Pour éviter les discontinuités, l'opcode *powershape* traite toutes les valeurs en entrée comme des nombres positifs (en prenant leur valeur absolue), mais il conserve leur signe original dans le signal de sortie. Ceci permet un modelage lisse de tout signal alors que l'exposant varie sur n'importe quel intervalle. (*powershape* traite également de manière intelligente les discontinuités qui peuvent se produire lorsque l'exposant et l'entrée sont tous deux nuls (heureusement). Noter cependant que les exposants négatifs causeront généralement un dépassement par le signal de l'amplitude maximale fixée par le paramètre *ifullscale* et qu'ils devraient ainsi être évités).

L'autre adaptation concerne le paramètre *ifullscale*. Le signal d'entrée est divisé par *ifullscale* avant d'être élevé à la puissance *kShapeAmount* et il est ensuite multiplié par *ifullscale* avant d'être retourné. Cela normalise le signal d'entrée dans l'intervalle  $[-1,1]$ , ce qui garantit que la sortie (avant la mise à l'échelle finale) sera aussi dans cet intervalle pour les valeurs de modelage positives, fournissant une fonction de transfert évoluant sans à coup tandis que la quantité de modelage varie. Les valeurs de *kShapeAmount* entre 0 et 1 rendent le signal plus "convexe" tandis que les valeurs supérieures à 1 le rendent plus "concave". Une valeur exacte de 1.0 ne produit aucun changement dans le signal d'entrée.

## Voir aussi

*pow*, *powoftwo*

## Exemples

Voici un exemple de l'opcode powershape. Il utilise le fichier *powershape.csd* [examples/powershape.csd].

### Exemple 729. Exemple de l'opcode powershape.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
instr 1
  imaxamp      =          10000
  kshapeamt    line      p5, p3, p6
  aosc         oscili     1.0, cpspch(p4), 1
  aout         powershape aosc, kshapeamt
  adeclick     linseg     0.0, 0.01, 1.0, p3 - 0.06, 1.0, 0.05, 0.0

      out          aout * adeclick * imaxamp
endin

</CsInstruments>
<CsScore>
f1 0 32768 10 1

i1 0 1      7.00  0.000001 0.8
i1 + 0.5    7.02  0.01    1.0
i1 + .      7.05  0.5     1.0
i1 + .      7.07  4.0     1.0
i1 + .      7.09  1.0     10.0
i1 + 2      7.06  1.0     25.0

e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Anthony Kozar  
Janvier 2008

Nouveau dans la version 5.08 de Csound.



# powoftwo

powoftwo — Calcule une puissance de deux.

## Description

Calcule une puissance de deux.

## Syntaxe

`powoftwo(x)` (argument au taux d'initialisation ou de contrôle seulement)

## Exécution

La fonction *powoftwo()* retourne  $2^x$  et accepte comme argument des nombres positifs et négatifs. L'intervalle des valeurs autorisées dans *powoftwo()* va de -5 à +5 permettant une précision plus fine qu'un cent dans un intervalle de dix octaves. Pour un intervalle de valeurs plus grand, utiliser l'opcode plus lent *pow*.

Ces fonctions sont rapides, car elles lisent des valeurs stockées dans des tables. Elles sont très utiles lorsque l'on travaille avec des rapports de hauteurs. Elles travaillent au taux-i et au taux-k.

## Exemples

Voici un exemple de l'opcode powoftwo. Il utilise le fichier *powoftwo.csd* [exemples/powoftwo.csd].

### Exemple 730. Exemple de l'opcode powoftwo.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o poweroftwo.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; bit reduction for Lo-Fi sound

iBits = p5          ;bit depth
iQuantize = powoftwo(iBits) ;find number of discrete steps for this bit depth
iQuantize = iQuantize*0.5 ;half the number of steps for each side of a bipolar signal
print iQuantize
asig = soundin "fox.wav"
asig = round(asig * iQuantize) / iQuantize ;quantize audio signal (bit reduce)
outs asig, asig
```

```
    endin
  </CsInstruments>
  <CsScore>
    ;          bits
    i1 0      3  16
    i1 ^+3    .  12
    i1 ^+3    .   8
    i1 ^+3    .   4
    i1 ^+3    .   2
    i1 ^+3    .   1

    e
  </CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
instr 1:  iQuantize = 32768.000
instr 1:  iQuantize = 2048.000
instr 1:  iQuantize = 128.000
instr 1:  iQuantize = 8.000
instr 1:  iQuantize = 2.000
instr 1:  iQuantize = 1.000
```

## Voir aussi

*logbtwo, pow*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Juin 1998

Auteur : John ffitch  
Université de Bath, Codemist, Ltd.  
Bath, UK  
Juillet 1999

Nouveau dans la version 3.57 de Csound

# prealloc

prealloc — Crée de l'espace pour des instruments mais ne les exécute pas.

## Description

Crée de l'espace pour des instruments mais ne les exécute pas.

## Syntaxe

```
prealloc insnum, icount
```

```
prealloc "insname", icount
```

## Initialisation

*insnum* -- numéro de l'instrument

*icount* -- nombre d'allocations de l'instrument

« *insname* » -- une chaîne de caractères (entre guillemets) représentant un instrument nommé.

## Exécution

Toutes les instances de *prealloc* doivent être définies dans la section d'en-tête, pas dans le corps de l'instrument.

## Exemples

Voici un exemple de l'opcode *prealloc*. Il utilise le fichier *prealloc.csd* [examples/prealloc.csd].

### Exemple 731. Exemple de l'opcode *prealloc*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o prealloc.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Pre-allocate memory for five instances of Instrument #1.
prealloc 1, 5
```

```
; Instrument #1
instr 1
  ; Generate a waveform, get the cycles per second from the 4th p-field.
  a1 oscil 6500, p4, 1
  out a1
endin

</CsInstruments>
<CsScore>

; Just generate a nice, ordinary sine wave.
f 1 0 32768 10 1

; Play five instances of Instrument #1 for one second.
; Note that 4th p-field contains cycles per second.
i 1 0 1 220
i 1 0 1 440
i 1 0 1 880
i 1 0 1 1320
i 1 0 1 1760
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*cpuprc, maxalloc*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Juillet 1999

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.57 de Csound

# prepiano

prepiano — Crée un son similaire à celui d'une corde de piano préparé à la manière Cage.

## Description

La sortie audio est un son similaire à celui d'une corde de piano préparé avec des gommes et des pièces de monnaie. La méthode utilise un modèle physique développé pour la résolution des équations différentielles partielles.

## Syntaxe

```
ares prepiano ifreq, iNS, iD, iK, \  
    iT30, iB, kbcl, kbcr, imass, ihvfreq, iinit, ipos, ivel, isfreq, \  
    isspread[, irattles, irubbers]  
  
al,ar prepiano ifreq, iNS, iD, iK, \  
    iT30, iB, kbcl, kbcr, imass, ihvfreq, iinit, ipos, ivel, isfreq, \  
    isspread[, irattles, irubbers]
```

## Initialisation

*ihvfreq* -- la fréquence de vibration naturelle du marteau.

*iNS* -- le nombre de cordes impliquées. Dans un vrai piano on trouve 1, 2 ou 3 cordes dans les différentes plages de fréquence.

*iD* -- l'importance du désaccord de chaque corde, hormis la première, par rapport à la fréquence principale ; mesuré en cents.

*iK* -- paramètre de raideur, sans dimension.

*iT30* -- durée de chute de 30 db en secondes.

*ib* -- paramètre de perte en haute-fréquence (à garder petit).

*imass* -- la masse du marteau.

*ifreq* -- la fréquence de vibration naturelle du marteau.

*iinit* -- la position initiale du marteau.

*ipos* -- position de la frappe sur la corde.

*ivel* -- vitesse normalisée de la frappe.

*isfreq* -- fréquence de balayage du point de lecture.

*isspread* -- dispersion de la fréquence de balayage.

*irattles* -- numéro de la table donnant les positions de la ou des pièces de monnaie.

*irubbers* -- numéro de la table donnant les positions de la ou des gommes.

Les tables des pièces de monnaie et des gommes sont des collections de quatre valeurs précédées par un compte. Dans le cas d'une pièce de monnaie, les quatre valeurs sont la position, le rapport de densité entre la pièce de monnaie et la corde, la fréquence de la pièce de monnaie et sa longueur verticale. Pour la gomme, les quatre valeurs sont la position, le rapport de densité entre la gomme et la corde, la fréquence de la gomme et le paramètre de perte.

## Exécution

Une note est jouée sur une corde de piano avec les arguments suivants.

*kbcL* -- Condition aux limites à l'extrémité gauche de la corde (1 fixée, 2 pivotante, 3 libre).

*kbcR* -- Condition aux limites à l'extrémité droite de la corde (1 fixée, 2 pivotante, 3 libre).

Il faut noter que le changement des conditions aux limites durant l'exécution peut produire des bruits parasites et que cette possibilité n'est fournie qu'à titre expérimental.

## Exemples

Voici en exemple de l'opcode *prepiano*. Il utilise le fichier *prepiano.csd* [examples/repiano.csd].

### Exemple 732. Exemple de l'opcode *prepiano*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
-iadc      ;;uncomment -iadc if real audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o prepiano.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
;;      fund NS detune stiffness decay loss (bndry) (hammer) scan prep
aa,ab prepiano 60, 3, 10, p4, 3, 0.002, 2, 2, 1, 5000, -0.01, p5, p6, 0, 0.1, 1, 2
outs aa*.2, ab*.2

endin
</CsInstruments>
<CsScore>
f1 0 8 2 1 0.6 10 100 0.001 ;; 1 rattle
f2 0 8 2 1 0.7 50 500 1000 ;; 1 rubber
i1 0.0 1 1 0.09 20
i1 1 . -1 0.09 40           ;; 1 -> skip initialisation
i1 2 . -1 0.09 60
i1 3 . -1 0.09 80
i1 4 1.8 -1 0.09 100
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Stefan Bilbao  
Université d'Edimbourg, UK  
Auteur : John ffitch  
Université de Bath, Codemist Ltd.  
Bath, UK

Nouveau dans la version 5.05 de Csound

# print

print — Affiche les valeurs de variables de taux-i.

## Description

Ces unités affichent des valeurs d'initialisation de l'orchestre.

## Syntaxe

```
print iarg [, iarg1] [, iarg2] [...]
```

## Initialisation

*iarg*, *iarg2*, ... -- arguments de taux-i.

## Exécution

*print* -- affiche la valeur courante des arguments (ou des expressions) de taux-i *iarg* à chaque passe d'initialisation dans l'instrument.



### Note

L'opcode *print* tronque des positions décimales et peut ainsi ne pas montrer la valeur complète. La précision de Csound varie selon la version float (32 bit) ou double (64 bit), car la plupart des calculs internes utilisent un de ces formats. Si l'on désire une sortie console avec plus de résolution, on peut essayer *printf*.

## Exemples

Voici un exemple de l'opcode print. Il utilise le fichier *print.csd* [examples/print.csd].

### Exemple 733. Exemple de l'opcode print.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o print.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
```



```
irand = rnd(3)    ;generate a random number from 0 to 3
print irand      ;print it
asig poscil .7, 440*irand, 1
    outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 16384 10 1 ;sine wave

i 1 0 1
i 1 2 1
i 1 4 1
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
+instr 1:  irand = 2.930
+instr 1:  irand = 0.420
+instr 1:  irand = 2.316
```

## Voir aussi

*dispf, display, printk, printk2, printks, printf and prints*

# printf

`printf` — Sortie formatée à la façon `printf`.

## Description

**`printf`** et **`printf_i`** écrivent une sortie formatée à la manière de la fonction C *printf()*. **`printf_i`** ne s'exécute qu'au taux-i, tandis que **`printf`** s'exécute à la fois à l'initialisation et pendant l'exécution de la note.

## Syntaxe

```
printf_i Sfmt, itrig, [iarg1[, iarg2[, ... ]]]
```

```
printf Sfmt, ktrig, [xarg1[, xarg2[, ... ]]]
```

## Initialisation

*Sfmt* -- chaîne de formatage ayant la même structure que dans *printf* et dans d'autres fonctions C similaires, sauf que les modificateurs de longueur (l, ll, h, etc.) ne sont pas supportés. Les indicateurs de conversion suivants sont permis :

- d, i, o, u, x, X, e, E, f, F, g, G, c, s

*iarg1*, *iarg2*, ... -- arguments d'entrée à formater (30 au maximum). Les formats entiers tels que %d arrondissent les valeurs d'entrée à l'entier le plus proche.

*itrig* -- s'il est supérieur à zéro, l'opcode effectue l'affichage ; sinon c'est une opération nulle.

## Exécution

*ktrig* -- s'il est supérieur à zéro et différent de sa valeur lors du cycle de contrôle précédent, l'opcode effectue l'affichage demandé. La valeur précédente initiale est fixée à zéro.

*xarg1*, *xarg2*, ... -- arguments d'entrée à formater (30 au maximum). Les formats entiers tels que %d arrondissent les valeurs d'entrée à l'entier le plus proche. Noter que seuls les arguments de taux-k et de taux-i sont valides (pas d'affichage au taux-a)

## Exemples

Voici un exemple de l'opcode `printf`. Il utilise le fichier *printf.csd* [examples/printf.csd].

### Exemple 734. Exemple de l'opcode `printf`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
```

```

;-o printf.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
Sfile      strget    p4
ivld       filevalid Sfile

if ivld=0 then
    printf_i "Audiofile '%s' does not exist!\n", 1, Sfile
else
    asig     diskint2 Sfile, 1
    outs     asig, asig
endif

endin

</CsInstruments>
<CsScore>

i 1 0 3 "frox.wav";file does not exist!!!
i 1 + 3 "fox.wav";but this one certainly does...

e
</CsScore>
</CsoundSynthesizer>

```

L'exemple produit la sortie suivante :

```
Audiofile 'frox.wav' does not exist!
```

## Voir aussi

Plus d'information sur *printf* : <http://www.cplusplus.com/reference/clibrary/cstdio/printf/> [<http://www.cplusplus.com/reference/clibrary/cstdio/printf/>]

## Crédits

Auteur : Istvan Varga  
2005

# printk2

printk2 — Affiche une nouvelle valeur chaque fois qu'une variable de contrôle change.

## Description

Affiche une nouvelle valeur chaque fois qu'une variable de contrôle change.

## Syntaxe

```
printk2 kvar [, inumspaces] [, inamed]
```

## Initialisation

*inumspaces* (facultatif, 0 par défaut) -- nombre d'espaces imprimés avant la valeur de *kvar*

*inamed* (facultatif, 0 par défaut) -- s'il est non nul, affiche le nom de la *kvar* ainsi que sa valeur.

## Exécution

*kvar* -- signal à imprimer

Dérivé du *printk* de Robin Whittle, il affiche une nouvelle valeur de *kvar* chaque fois que *kvar* change. Utile pour surveiller les changements des contrôles MIDI lorsque l'on utilise des réglettes.



### Avertissement

Ne pas utiliser cet opcode avec des signaux de taux-k normaux variant continuellement, car cela pourrait bloquer l'ordinateur, le taux d'impression devenant trop rapide.

## Exemples

Voici un exemple de l'opcode printk2. Il utilise le fichier *printk2.csd* [examples/printk2.csd].

### Exemple 735. Exemple de l'opcode printk2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o printk2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
```

```
instr 1

kval    randomh 0, 1.2, 20 ;choose between 0 and 1.2
if kval >0 && kval<=.5 then ;3 possible outcomes
    kval = 1
elseif kval >.5 && kval<=1 then
    kval =2
elseif kval >1 then
    kval =3
endif

printk2 kval ;print value when it changes
asig    poscil .7, 440*kval, 1
        outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 16384 10 1

i 1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

La sortie contiendra des lignes comme celles-ci :

```
i1      0.00000
i1      3.00000
i1      1.00000
i1      3.00000
i1      1.00000
i1      2.00000
i1      3.00000
```

## Voir aussi

*printk* and *printks*

## Crédits

Auteur : Gabriel Maldonado  
 Italie  
 1998

Nouveau dans la version 3.48 de Csound

Option du nom ajoutée dans la 6.11

# printk

printk — Affiche une valeur de taux-k à intervalles définis.

## Description

Affiche une valeur de taux-k à intervalles définis.

## Syntaxe

```
printk itime, kval [, ispace] [, inamed]
```

## Initialisation

*itime* -- intervalle en secondes entre les impressions.

*ispace* (facultatif, 0 par défaut) -- nombre d'espaces à insérer avant l'impression. (0 par défaut, max : 130)

*inamed* (facultatif, 0 par défaut) -- s'il est non nul, affiche le nom de la *kvar* ainsi que sa valeur.

## Exécution

*kval* -- La valeur de taux-k à afficher.

*printk* imprime une valeur de taux-k à chaque cycle-k, à chaque seconde ou à intervalles définis. Le numéro d'instrument est d'abord imprimé, puis le temps absolu en secondes, ensuite un nombre donné d'espaces, enfin la valeur de *kval*. Le nombre variable d'espaces permet de répartir différentes valeurs sur l'écran, de manière plus visible.

Cet opcode peut être exécuté à chaque cycle-k de l'instrument auquel il appartient. Pour cela, il faut mettre *itime* à 0.

Si *itime* est différent de 0, l'opcode imprime sur le premier cycle-k lors de son appel, puis chaque fois qu'une durée *itime* s'est écoulée. Le temps commence à s'écouler à partir de l'initialisation de l'opcode, typiquement à l'initialisation de l'instrument.

## Exemples

Voici un exemple de l'opcode printk. Il utilise le fichier *printk.csd* [examples/printk.csd].

### Exemple 736. Exemple de l'opcode printk.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o printk.wav -W ;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
instr 1
; Change a value linearly from 0 to 100,
; over the period defined by p3.
kval line 0, p3, 100

; Print the value of kval, once per second.
printk 1, kval
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 5 seconds.
i 1 0 5
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme :

i	1	time	0.00002:	0.00000
i	1	time	1.00002:	20.01084
i	1	time	2.00002:	40.02999
i	1	time	3.00002:	60.04914
i	1	time	4.00002:	79.93327

## Voir aussi

*printk2* and *printks*

## Crédits

Auteur : Robin Whittle  
 Australie  
 Mai 1997

Option du nom ajoutée dans la 6.11

Exemple écrit par Kevin Conder.

Merci à Luis Jure pour avoir signalé une erreur concernant le paramètre *itime*.

# printks

printks — Imprime au taux-k avec une syntaxe à la printf().

## Description

Imprime au taux-k avec une syntaxe à la *printf()*.

## Syntaxe

```
printks "string", itime [, xval1] [, xval2] [...]
```

## Initialisation

*"string"* -- la chaîne de caractères à imprimer. Peut contenir jusqu'à 8192 caractères et doit être entre guillemets.

*itime* -- intervalle en secondes entre les impressions.

## Exécution

*xval1*, *xval2*, ... (facultatif) -- Les valeurs de taux-k à imprimer. Celles-ci sont spécifiées dans « *string* » au moyen des indicateurs de valeur du C standard (%f, %d, etc.) dans l'ordre donné.

A partir de la version 4.23 de Csound, on peut utiliser autant de variables *kval* que l'on veut. Dans les versions antérieures à la 4.23, on doit donner 4 et seulement 4 kvals (mettant 0 pour les kvals non utilisées).

*printks* affiche des nombres et du texte qui peuvent être des valeurs de taux-i ou de taux-k. *printks* est extrêmement flexible, et si on l'utilise avec des codes de positionnement du curseur, il peut servir à écrire des valeurs spécifiques à certaines positions de l'écran pendant l'exécution de Csound.

Un mode d'opération spécial permet à *printks* de convertir le paramètre d'entrée *kval1* en valeur comprise entre 0 et 255 et de l'utiliser comme le premier caractère à imprimer. Un programme Csound peut ainsi envoyer des caractères arbitraires à la console. Pour cela, il faut que le premier caractère de la chaîne soit un # éventuellement suivi de texte normal et d'indicateurs de format.

Cet opcode peut être exécuté à chaque cycle-k de l'instrument auquel il appartient. Pour cela, il faut mettre *itime* à 0.

Si *itime* est différent de 0, l'opcode imprime sur le premier cycle-k lors de son appel, puis chaque fois qu'une durée *itime* s'est écoulée. Le temps commence à s'écouler à partir de l'initialisation de l'opcode, typiquement à l'initialisation de l'instrument.

## Formatage de l'Impression

Tous les caractères de contrôle de *printf()* du langage C standard peuvent être utilisés. Par exemple, si *kval1* = 153.26789, voici quelques-unes des options de formatage habituelles :

1. %f imprime avec toute la précision : 153.26789
2. %5.2f imprime : 153.26



3. %d n'imprime que la partie entière : 153
4. %c traite *kvalI* comme le code ASCII d'un caractère.

En plus de tous les codes de *printf()*, *printks* supporte ces codes de caractère utiles :

Code printks	Code de Caractère
\\r, \\R, %r, or %R	retour chariot (\r)
\\n, \\N, %n, %N	caractère de nouvelle ligne (\n)
\\t, \\T, %t, or %T	tabulation (\t)
%!	point-virgule (;) C'est nécessaire car un « ; » est interprété comme un commentaire.
^	caractère d'échappement (0x1B)
^ ^	accent circonflexe (^)
~	ESC[ (escape+[ est la séquence d'échappement des consoles ANSI)
~~	tilde (~)

Pour plus d'information sur le formatage à la *printf()*, consulter une documentation sur le langage C.



### Note

Avant la version 4.23, seul le code de format %f était supporté.

## Exemples

Voici un exemple de l'opcode *printks*. Il utilise le fichier *printks.csd* [examples/printks.csd].

### Exemple 737. Exemple de l'opcode *printks*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o printks.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
instr 1
; Change a value linearly from 0 to 100,
; over the period defined by p3.
kup line 0, p3, 100
```

```
; Change a value linearly from 30 to 10,  
; over the period defined by p3.  
kdown line 30, p3, 10  
  
; Print the value of kup and kdown, once per second.  
printks "kup = %f, kdown = %f\\n", 1, kup, kdown  
endin  
  
</CsInstruments>  
<CsScore>  
  
; Play Instrument #1 for 5 seconds.  
i 1 0 5  
e  
  
</CsScore>  
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme :

```
kup = 0.000000, kdown = 30.000000  
kup = 20.010843, kdown = 25.962524  
kup = 40.029991, kdown = 21.925049  
kup = 60.049141, kdown = 17.887573  
kup = 79.933266, kdown = 13.872493
```

## Voir aussi

*printk2* et *printk*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Exemple écrit par Kevin Conder.

Merci à Luis Jure pour avoir signalé une erreur concernant le paramètre *itime*.

Merci à Matt Ingalls pour la mise à jour de la documentation pour la version 4.23.

# printks2

`printks2` — Imprime une nouvelle valeur à chaque changement d'une variable de contrôle, en utilisant une syntaxe à la `printf()`.

## Description

Imprime une nouvelle valeur à chaque changement d'une variable de contrôle, en utilisant une syntaxe à la `printf()`.

## Syntaxe

```
printks2 "string", kval
```

## Initialisation

*"string"* -- la chaîne de caractère indiquant le format.

## Exécution

*kval* -- signal à imprimer. Le style de l'impression est spécifié dans « *string* » avec les spécificateurs standard du C (%f, %d, etc.).

## Formatage de l'impression

On peut utiliser tous les caractères de contrôle standard de `printf()` du langage C. Par exemple, si *kval* = 153.26789 quelques options de formatage sont :

1. %f imprime en pleine précision : 153.26789
2. %5.2f imprime : 153.26
3. %d imprime seulement des entiers : 153
4. %c traite *kval* comme un code de caractère ASCII.

En plus de tous les codes de `printf()`, *printks2* supporte ces codes de caractère utiles :

Code de <code>printks2</code>	Code de caractère
<code>\\r, \\R, %r ou %R</code>	Retour chariot ( <code>\r</code> )
<code>\\n, \\N, %n, %N</code>	Caractère de nouvelle ligne ( <code>\n</code> )
<code>\\t, \\T, %t, or %T</code>	Tabulation ( <code>\t</code> )
<code>%!</code>	point-virgule (;) Nécessaire car un « ; » est interprété comme un commentaire.
<code>^</code>	Caractère d'échappement (0x1B)
<code>^ ^</code>	Accent circonflexe (^)
<code>~</code>	ESC[ (escape+[ est la séquence d'échappement pour les consoles ANSI)

Code de printks2	Code de caractère
~~	tilde (~)

Pour plus d'information sur le formatage de printf(), consulter une documentation sur le langage C.

## Exemples

Voici un exemple de l'opcode printks2. Il utilise le fichier *printks2.csd* [examples/printks2.csd].

### Exemple 738. Exemple de l'opcode printks2.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
; For Non-realtime ouput leave only the line below:
; -o printk.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
instr 1
; Change a value linearly from 0 to 100,
; over the period defined by p3.
kval line 0, p3, 100

; Print the value of kval when it changes.
printks2 "value now %f\n", int(kval)
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 5 seconds.
i 1 0 5
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*printk2* and *printk2* and *printk*

## Crédits

Auteur : John ffitch  
Bath, UK

Mars 2014

Nouveau dans la version 6.03 de Csound.

# prints

prints — Imprime au taux-i avec une syntaxe à la `printf()`.

## Description

Imprime au taux-i avec une syntaxe à la *printf()*.

## Syntaxe

```
prints "string" [, xval1] [, xval2] [...]
```

## Initialisation

*"string"* -- la chaîne de caractères à imprimer. Peut contenir jusqu'à 8192 caractères et doit être entre guillemets.

## Exécution

*xval1*, *xval2*, ... (optional) -- Les valeurs de taux-k à imprimer. Celles-ci sont spécifiées dans « *string* » au moyen des indicateurs de valeur du C standard (%f, %d, etc.) dans l'ordre donné.

*prints* est semblable à l'opcode *printks* sauf qu'il opère au taux-i plutôt qu'au taux-k. Pour plus d'information sur le formatage de la sortie, veuillez consulter la documentation de *printks*.

## Exemples

Voici un exemple de l'opcode *prints*. Il utilise le fichier *prints.csd* [examples/prints.csd].

### Exemple 739. Exemple de l'opcode *prints*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o prints.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Matt Ingalls, edited by Kevin Conder. */
; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Init-time print.
```

```
prints "%2.3f\\t%!%!!%!%!semicolons! %%\\n", 1234.56789
endin
```

```
</CsInstruments>
<CsScore>
```

```
/* Written by Matt Ingalls, edited by Kevin Conder. */
; Play instrument #1.
i 1 0 0.004
```

```
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme :

```
1234.568          ;;;;semicolons!
```

## Voir aussi

*prints*

## Crédits

Auteur : Matt Ingalls  
Janvier 2003

# printarray

printarray — Affiche le contenu d'un tableau.

## Description

Opcodes du greffon emugens.

Affiche le contenu d'un tableau.

## Syntaxe

```
printarray ixs[] [, Sfmt, Slabel ]  
printarray kxs[] [, ktrig, Sfmt, Slabel ]
```

## Initialisation

*Sfmt* -- S'il est présent, il est passé à printf pour chaque élément du tableau. Sinon un format par défaut est utilisé.

*Slabel* -- S'il est présent, il est affiché avant le contenu du tableau, pour identifier facilement les données.

## Exécution

*ktrig* -- Le tableau est affiché chaque fois que cette valeur change de 0 à un nombre positif. On peut l'utiliser avec *metro* pour afficher à intervalle donné. Une valeur de -1 indique d'afficher à chaque cycle-k (1 par défaut).

## Exmples

Voici un exemple de l'opcode printarray. Il utilise le fichier *printarray.csd* [examples/printarray.csd].

### Exemple 740. Exemple de l'opcode printarray.

```
<CsoundSynthesizer>  
<CsOptions>  
  
--nosound  
  
</CsOptions>  
<CsInstruments>  
  
instr 1  
  ; test i-time, 1D  
  ivalues[] fillarray 0, 1, 3, 5, 7, 9  
  printarray ivalues           ; default fmt, no label  
  printarray ivalues, "%.2f"   ; with given fmt  
  printarray ivalues, "", "ivalues = " ; uses default fmt  
  
  ; test i-time, long array  
  ilong[] genarray 0, 3, 0.01  
  printarray ilong  
  printarray ilong, "%.2f", "long="
```

; 2D



```

ivalues2[][] init 11, 4
ivalues2 fillarray 0, 1, 2, 3, \
                    10, 11, 12, 13, \
                    20, 21, 22, 23, \
                    30, 31, 32, 33, \
                    40, 41, 42, 43, \
                    50, 51, 52, 53, \
                    60, 61, 62, 63, \
                    70, 71, 72, 73, \
                    80, 81, 82, 83, \
                    90, 91, 92, 93, \
                    100,101,102,103
printarray ivalues2, "%.2f"
printarray ivalues2, "%.1f", "ivalues2="
turnoff
endin

instr 2
; k-time, 1D, print every cycle
kxs[] fillarray 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
printarray kxs, -1, "", "instr 2"
kxs += 1
endin

instr 3
kxs[] fillarray 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
printarray kxs, metro(20), "%.0f", "kxs 1D="
kxs += 1
if kxs[0] > 1000 then
    turnoff
endif
endin

instr 4
; k-time, 2D, print with a trigger
kxs[][] init 3, 4
kxs fillarray 0, 1, 2, 3, \
              10, 11, 12, 13, \
              20, 21, 22, 23

ktrig metro 20
printarray kxs, ktrig, "", "kxs="
kxs[0][0] = kxs[0][0] + 1
if kxs[0][0] > 1000 then
    turnoff
endif
endin

instr 5
; test %d: it should work like printf("%d", (int)myfloat)
; this should print "0 1 2 3 4"
kxs[] fillarray 0, 1.1, 2, 3.3, 4
printarray kxs, 1, "%d"
turnoff
endin

</CsInstruments>

<CsScore>
i 1 0 0.01
i 2 1 0.05
i 3 2 2
i 4 2 2
i 5 0 0.1
</CsScore>

</CsSoundSynthesizer>

```

## Voir aussi

*ftprint, metro printf*

## Crédits

Auteur : Eduardo Moguillansky 2018

Nouveau greffon dans la version 6.12

# product

product — Multiplie n'importe quel nombre de signaux de taux-a.

## Description

Multiplie n'importe quel nombre de signaux de taux-a.

## Syntaxe

```
ares product asig1, asig2 [, asig3] [...]
```

## Exécution

*asig1, asig2, asig3, ...* -- signaux de taux-a à multiplier.

## Exemples

Voici un exemple de l'opcode product. Il utilise le fichier *product.csd* [examples/product.csd].

### Exemple 741. Exemple de l'opcode product.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o product.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisine ftgen 0, 0, 2^10, 10, 1

instr 1

a1  oscili 1, 10.0, gisine ;combine 3 sinusses
a2  oscili 1, 1.0, gisine  ;at different rates
a3  oscili 1, 3.0, gisine
ares product a1, a2, a3

ares = ares*10000    ;scale result and
asig poscil .5, ares+110, gisine ;add to frequency
outs asig, asig

endin
</CsInstruments>
<CsScore>
```

```
i1 0 5  
e  
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Avril 1999

Nouveau dans la version 3.54 de Csound

# product

product — Calcule le produit d'un tableau.

## Description

Prend un tableau numérique (taux-k ou i) et calcule son produit.

## Syntaxe

```
kres/iresproduct karr[]/iarr[] (k- or i-arrays )
```

## Exemples

Voici un exemple de l'opcode product. Il utilise le fichier *producta.csd* [examples/productarray.csd].

### Exemple 742. Exemple de l'opcode product.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-n
</CsOptions>
<CsInstruments>

instr 1
  kArr1[] fillarray 1,3,2,7,4
  k1 product kArr1
  printk2 k1
  turnoff
endin

</CsInstruments>
<CsScore>
i1 0 1
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*dot sumarray*

## Crédits

Auteur : Victor Lazzarini

Nouveau dans la version 6.09

# pset

pset — Définit et initialise des tableaux numériques au chargement de l'orchestre.

## Description

Définit et initialise des tableaux numériques au chargement de l'orchestre.

## Syntaxe

```
pset icon1 [, icon2] [...]
```

## Initialisation

*icon1*, *icon2*, ... -- valeurs de preset pour un instrument MIDI

*pset* (facultatif) définit et initialise des tableaux numériques au chargement de l'orchestre. On peut l'utiliser comme instruction dans l'en-tête de l'orchestre (c'est-à-dire dans l'instrument 0) ou dans un instrument. Lorsqu'il est défini dans un instrument, il ne fait pas partie de ses opérations des périodes d'initialisation ou d'exécution, et une seule de ces instructions est autorisée par instrument. Ces valeurs sont disponibles comme valeurs d'initialisation par défaut. Quand un instrument est déclenché à partir du MIDI, il ne reçoit que p1 et p2 de l'évènement, alors que p3, p4, etc proviennent des valeurs définies dans le preset.

## Exemples

Voici un exemple de l'opcode pset. Il utilise le fichier *pset.csd* [examples/pset.csd]

### Exemple 743. Exemple de l'opcode pset.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime output leave only the line below:
; -o pset.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1 ;this shows an example with non-midi use

pset 1, 0, 1, 220, 0.5
asig poscil p5, p4, 1
outs asig, asig

endin
</CsInstruments>
```

```
<CsScore>
f 1 0 1024 10 1 ;sine wave

i 1 0 1
i 1 1 1 440
i 1 2 1 440 0.1
e
</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple d'utilisation de l'opcode pset avec le midi. Il utilise le fichier *pset-midi.csd* [exemples/pset-midi.csd]

### Exemple 744. Second exemple de l'opcode pset.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0 --midi-key-oct=4 --midi-velocity=5   ;;;realtime audio out and virtual midi
;-iadc   ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pset-midi.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1

        pset 0, 0, 3600, 0, 0, 0
iinstrument = p1
istarttime = p2
iattack = 0.005
isustain = p3
irelease = 0.06
p3 = isustain + iattack + irelease
kdamping linsegr 0.0, iattack, 1.0, isustain, 1.0, irelease, 0.0

ioctave = p4
ifrequency = cpsoct(ioctave)
iamplitude = p5*.15 ;lower volume

print p1, p2, p3, p4, p5
asig STKBandedWG ifrequency, iamplitude
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 0 60 ; runs 69 seconds
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*strset*

# ptable

ptable — Accède aux valeurs d'une table par indexation directe.

## Description

Accède aux valeurs d'une table par indexation directe.

## Syntaxe

```
ares ptable andx, ifn [, ixmode] [, ixoff] [, iwrap]
ires ptable indx, ifn [, ixmode] [, ixoff] [, iwrap]
kres ptable kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

## Initialisation

*ifn* -- numéro de la table de fonction.

*ixmode* (facultatif) -- mode d'indexation. La valeur par défaut est 0.

- 0 = indexation brute
- 1 = indexation normalisée (0 à 1)

*ixoff* (facultatif) -- décalage de l'index. Pour une table dont l'origine est centrée, on utilise *taille-de-la-table*/2 (indexation brute) ou 0.5 (indexation normalisée). La valeur par défaut est 0.

*iwrap* (facultatif) -- indicateur d'indexation cyclique. La valeur par défaut est 0.

- 0 = pas d'enroulement (les index < 0 sont considérés comme nuls ; les index > *taille-de-la-table* sont bloqués à index=*taille-de-la-table*)
- 1 = indexation cyclique.

## Exécution

*ptable* invoque une lecture de table avec des indices au taux d'initialisation, au taux de contrôle ou au taux audio. Ces indices peuvent être des entrées brutes (0, 1, 2, ..., *taille*-1) ou des valeurs normalisées (0 à 1). Les indices sont d'abord modifiés par la valeur de décalage puis leur appartenance à l'intervalle valide est testée avant la lecture dans la table (voir *iwrap*). Si l'indice varie sur toute l'échelle, ou si on utilise l'interpolation, la table doit avoir un point de garde. *ptable* indexé par un phaseur périodique (voir *phasor*) simulera un oscillateur.

## Exemples

Voici un exemple de l'opcode *ptable*. Il utilise le fichier *ptable.csd* [exemples/*ptable.csd*].

### Exemple 745. Exemple de l'opcode *ptable*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.



```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o table.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Vary our index linearly from 0 to 1.
kndx line 0, p3, 1

; Read Table #1 with our index.
ifn = 1
ixmode = 1
kfreq ptable kndx, ifn, ixmode

; Generate a sine waveform, use our table values
; to vary its frequency.
a1 oscil 20000, kfreq, 2
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a line from 200 to 2,000.
f 1 0 1025 -7 200 1024 2000
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*table, tablei, table3, ptable3, ptablei, oscil1, oscilli, osciln*

## Crédits

Auteur : John ffitch  
Janvier 2012

Nouveau dans la version 5.16 de Csound.

# ptablei

`ptablei` — Accède aux valeurs d'une table par indexation directe avec interpolation linéaire.

## Description

Accède aux valeurs d'une table par indexation directe avec interpolation linéaire.

## Syntaxe

```
ares ptablei andx, ifn [, ixmode] [, ixoff] [, iwrap]
ires ptablei indx, ifn [, ixmode] [, ixoff] [, iwrap]
kres ptablei kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

## Initialisation

*ifn* -- numéro de la table de fonction.

*ixmode* (facultatif) -- mode d'indexation. La valeur par défaut est 0.

- 0 = indexation brute
- 1 = indexation normalisée (0 à 1)

*ixoff* (facultatif) -- décalage de l'index. Pour une table dont l'origine est centrée, on utilise  $\text{taille-de-la-table}/2$  (indexation brute) ou 0.5 (indexation normalisée). La valeur par défaut est 0.

*iwrap* (facultatif) -- indicateur d'indexation cyclique. La valeur par défaut est 0.

- 0 = pas d'enroulement (les index  $< 0$  sont considérés comme nuls ; les index  $> \text{taille-de-la-table}$  sont bloqués à  $\text{index}=\text{taille-de-la-table}$ )
- 1 = indexation cyclique.

## Exécution

`ptablei` est un opcode avec interpolation dans lequel la partie fractionnaire de l'index est utilisée pour interpoler entre entrées adjacentes de la table. Le lissage obtenu par interpolation se fait au prix d'une petite durée d'exécution supplémentaire (voir aussi *oscili*, etc.), sinon les opcodes avec et sans interpolation sont interchangeables.

## Exemples

Voici un exemple de l'opcode `ptablei`. Il utilise le fichier *ptablei.csd* [examples/ptablei.csd].

### Exemple 746. Exemple de l'opcode `ptablei`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;;realtime audio out
```

```

;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tablei.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

seed 0 ;generate new values every time the instr is played

instr 1

ifn = p4
isize = p5
ithresh = 0.5

itemp ftgen ifn, 0, isize, 21, 2

iwrite_value = 0
i_index = 0

loop_start:
    iread_value ptablei i_index, ifn

    if iread_value > ithresh then
        iwrite_value = 1
    else
        iwrite_value = -1
    endif
    ptableiw iwrite_value, i_index, ifn
    loop_lt i_index, 1, isize, loop_start
    turnoff

endin

instr 2

ifn = p4
isize = ftlen(ifn)
prints "Index\tValue\n"

i_index = 0
loop_start:
    ivalue tablei i_index, ifn
    prints "%d:\t%f\n", i_index, ivalue

    loop_lt i_index, 1, isize, loop_start ;read table 1 with our index

aout oscili .5, 100, ifn ;use table to play the polypulse
outs aout, aout

endin
</CsInstruments>
<CsScore>
i 1 0 1 100 16
i 2 0 2 100
e
</CsScore>
</CsSoundSynthesizer>

```

## Voir aussi

*table, tablei, table3, ptable, ptable3, oscil1, oscilli, osciln*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/issue12/genInstruments.html> [<http://www.csoundjournal.com/issue12/genInstruments.html>], écrit par Jacob Joaquin.

## Crédits

Auteur : John ffitch  
Janvier 2012

Nouveau dans la version 5.16 de Csound.

# ptable3

ptable3 — Accède aux valeurs d'une table par indexation directe avec interpolation cubique.

## Description

Accède aux valeurs d'une table par indexation directe avec interpolation cubique.

## Syntaxe

```
ares ptable3 andx, ifn [, ixmode] [, ixoff] [, iwrap]
ires ptable3 indx, ifn [, ixmode] [, ixoff] [, iwrap]
kres ptable3 kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

## Initialisation

*ifn* -- numéro de la table de fonction.

*ixmode* (facultatif) -- mode d'indexation. La valeur par défaut est 0.

- 0 = indexation brute
- 1 = indexation normalisée (0 à 1)

*ixoff* (facultatif) -- décalage de l'index. Pour une table dont l'origine est centrée, on utilise  $\text{taille-de-la-table}/2$  (indexation brute) ou 0.5 (indexation normalisée). La valeur par défaut est 0.

*iwrap* (facultatif) -- indicateur d'indexation cyclique. La valeur par défaut est 0.

- 0 = pas d'enroulement (les  $\text{index} < 0$  sont considérés comme nuls ; les  $\text{index} > \text{taille-de-la-table}$  sont bloqués à  $\text{index}=\text{taille-de-la-table}$ )
- 1 = indexation cyclique.

## Exécution

*ptable3* est identique à *table3*, sauf que l'on est pas obligé d'utiliser une table dont la taille est une puissance de deux.

## Voir aussi

*table*, *tablei*, *table3*, *ptable*, *ptablei*, *oscil1*, *oscil1i*, *osciln*

## Crédits

Auteur : John ffitch  
Janvier 2012

Nouveau dans la version 5.16 de Csound.

# ptablew

ptablew — Change le contenu de tables de fonction existantes de n'importe quelle taille.

## Description

Cet opcode opère sur des tables de fonction existantes, en changeant leur contenu. *ptablew* permet d'écrire aux taux-k et -a, avec le numéro de table spécifié à l'initialisation. On peut utiliser *ptablew* avec un signal et des valeurs d'index de taux-i, mais les données spécifiées seront toujours écrites dans la table de fonction au taux-k, pas pendant la passe d'initialisation. Les combinaisons valides de types de variables sont données par la première lettre des noms de variable.

## Syntaxe

```
ptablew asig, andx, ifn [, ixmode] [, ixoff] [, iwgmodes]
```

```
ptablew isig, indx, ifn [, ixmode] [, ixoff] [, iwgmodes]
```

```
ptablew ksig, kndx, ifn [, ixmode] [, ixoff] [, iwgmodes]
```

## Initialisation

*asig, isig, ksig* -- La valeur à écrire dans la table.

*andx, indx, kndx* -- Index dans la table, soit un nombre positif inférieur à la longueur de la table (*ixmode* = 0) ou dans l'intervalle allant de 0 à 1 (*ixmode* != 0)

*ifn* -- Numéro de la table. Doit être >= 1. Les nombres flottants sont arrondis à une valeur entière. Si le numéro de table n'est pas celui d'une table valide, ou si la table n'a pas encore été chargée (*GEN01*), il y aura une erreur et l'instrument sera désactivé.

*ixmode* (facultatif, 0 par défaut) -- mode d'indexation.

- 0 = *xndx* et *ixoff* varient dans l'intervalle compris entre 0 et la longueur de la table.
- !=0 = *xndx* et *ixoff* varient dans l'intervalle compris entre 0 et 1.

*ixoff* (facultatif, 0 par défaut) -- décalage de l'index.

- 0 = l'index est contrôlé directement par *xndx*, l'indexation partant du début de la table.
- !=0 = l'indexation part d'un endroit dans la table. Les valeurs doivent être positives et inférieures à la longueur de la table (*ixmode* = 0) ou inférieures à 1 (*ixmode* != 0).

*iwgmodes* (facultatif, 0 par défaut) -- Mode cyclique et point de garde.

- 0 = mode limite.
- 1 = mode cyclique.
- 2 = mode avec point de garde.

## Exécution

### Mode limite (0)

Limite l'index total ( $ndx + ixoff$ ) entre 0 et le point de garde. Pour une table de longueur 5, cela signifie que l'on peut écrire à la position 4 (le point de garde). Un index total négatif écrit à la position 0.

### Mode cyclique (1)

Replie l'index total entre les positions 0 et E, où E est inférieur à la longueur de la table d'une unité. Par exemple, si le repliement se fait dans l'intervalle compris entre 0 et 3, un index total de 6 écrira à la position 2.

### Mode point de garde (2)

Le point de garde est rempli en même temps que la position 0, avec la même valeur.

Ceci facilite l'écriture dans les tables que l'on doit lire avec interpolation pour produire des formes d'ondes périodiques lisses. De plus, avant son utilisation, l'index total est incrémenté de la moitié de la distance entre deux positions adjacentes, avant d'être arrondi à la valeur entière de la position dans la table.

Normalement ( $igwmode = 0$  ou  $1$ ) pour une table de longueur 5, dont les positions de 0 à 3 sont dans la table principale et dont la position 4 est le point de garde, un index total compris entre 0 et 0.999 écrira à la position 0. ("0.999" signifiant juste en dessous de 1.0). S'il est compris entre 1.0 et 1.999, il écrira à la position 1, etc. Tous les index totaux entre 0 et 4.999 ( $igwmode = 0$ ) ou 3.999 ( $igwmode = 1$ ) suivent le même schéma.  $igwmode = 0$  permet d'écrire dans les positions 0 à 4, avec la possibilité d'avoir dans le point de garde (4) une valeur différente de celle qui est à la position 0.

Avec une table de longueur 5 et  $igwmode = 2$ , lorsque l'index total est compris entre 0 et 0.499, on écrit dans les positions 0 à 4. L'intervalle compris entre 0.5 et 1.499 écrit dans la position 1, etc. De 3.5 à 4 on écrit aussi dans les positions 0 et 4.

De cette manière, l'opération d'écriture est au plus proche des résultats de la lecture avec interpolation. Le mode point de garde ne doit être utilisé qu'avec des tables ayant un point de garde.

Guardpoint mode is accomplished by adding 0.5 to the total index, rounding to the next lowest integer, wrapping it modulo the factor of two which is one less than the table length, writing the table (locations 0 to 3 in our example) and then writing to the guard point if index = 0.

*ptablew* n'a pas de valeur de sortie. Les trois derniers paramètres sont facultatifs et valent 0 par défaut.

## Précaution avec les numéros de tables de taux-k

Au taux-k et au taux-a, si un numéro de table  $< 1$  est donné, ou si le numéro de table pointe sur une table non-existante ou sur une table ayant une longueur nulle (à charger ultérieurement depuis un fichier), une erreur est générée et l'instrument est désactivé. Il faut initialiser *kfn* et *afn* au taux approprié en utilisant *init*. Si l'on tente de mettre une valeur de taux-i dans *kfn* ou dans *afn*, il y aura une erreur.



### Avertissement

Noter que *ptablew* est toujours un opcode de taux-k. Cela signifie que même sa version de taux-i est exécutée au taux-k et elle écrit la valeur de la variable de taux-i. Pour cette raison, le code suivant ne fonctionnera pas comme attendu :

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
giFt ftgen 1, 0, 8, 2, 0
instr 1
indx = 0
    ptablew 10, indx, giFt
ival tab_i indx, giFt
    print ival
endin
</CsInstruments>
<CsScore>
i 1 0 1
</CsScore>
</CsoundSynthesizer>
```

Bien que l'on s'attende à ce que ce programme affiche un 10 sur la console, il affichera 0, car *tab\_i* lira la valeur à l'initialisation de la note, avant la première passe d'exécution, quand *ptablew* écrit sa valeur.

## Voir aussi

*tableiw, tablewkt*

## Crédits

Auteur : John ffitch d'après Robin Whittle  
Février 2012



# ptrack

ptrack — Détecte la hauteur d'un signal.

## Description

*ptrack* prend un signal en entrée, le divise en blocs de taille *ihopsize* et extrait, par une méthode de TFCT, la hauteur, qui est une estimation de sa fréquence fondamentale, et une estimation de l'amplitude totale du signal en dB par rapport à l'échelle totale (0 dB). La méthode comporte une taille de fenêtre d'analyse de  $2*ihopsize$  échantillons (avec un recouvrement d'1/2 fenêtre), qui doit être une puissance de deux, entre 128 et 8192 (taille des sauts entre 64 et 4096). Plus les fenêtres sont courtes et meilleure est la précision temporelle, mais avec une précision en fréquence moins bonne (spécialement pour des fondamentales graves). Cet opcode est basé sur un algorithme original de M. Puckette.

## Syntaxe

```
kcps, kamp ptrack asig, ihopsize[,ipeaks]
```

## Initialisation

*ihopsize* -- taille des "sauts" d'analyse, en échantillons, devant être une puissance de deux (min 64, max 4096). C'est la durée entre deux mesures.

*ipeaks, ihi* -- nombre de pics spectraux à utiliser dans l'analyse. 20 par défaut (facultatif).

## Exécution

*kcps* -- hauteur estimée en Hz.

*kamp* -- amplitude estimée en dB par rapport à l'échelle totale (0 dB) (c-à-d toujours  $\leq 0$ ).

*ptrack* analyse le signal d'entrée, *asig*, pour retourner une paire hauteur/amplitude, pour le fondamental d'un signal monophonique. La sortie est mise à jour toutes les  $sr/ihopsize$  secondes.

## Exemples

Voici en exemple de l'opcode *ptrack*. Il utilise le fichier *ptrack.csd* [exemples/ptrack.csd].

### Exemple 747. Exemple de l'opcode *ptrack*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;;realtime audio out
;-iadc    ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o ptrack.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1

ihop = p4
aout diskin2 "fox.wav",1, 0, 1
kf,ka ptrack aout, ihop ; pitch track with different hopsizes
kcps port kf, 0.01 ; smooth freq
kamp port ka, 0.01 ; smooth amp
; drive an oscillator
asig poscil ampdB(kamp)*0dbfs, kcps, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
; simple sine wave
f 1 0 4096 10 1

i 1 0 5 128
i 1 6 5 512
i 1 12 5 1024
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
NUI, Maynooth.  
Maynooth, Irlande  
Mars 2007

Nouveau dans la version 5.05 de Csound.

# puts

`puts` — Imprime une chaîne de caractères constante ou variable.

## Description

`puts` imprime une chaîne de caractères terminée par un retour à la ligne facultatif chaque fois que le signal de déclenchement est positif et change de valeur.

## Syntaxe

```
puts Sstr, ktrig[, inonl]
```

## Initialisation

`Sstr` -- chaîne à imprimer.

`inonl` (facultatif, 0 par défaut) -- s'il est différent de zéro, désactive l'impression automatique d'un retour à la ligne à la fin de la chaîne.

## Exécution

`ktrig` -- signal de déclenchement, doit être valide au temps-i. La chaîne est imprimée à l'initialisation si `ktrig` est positif, et pendant l'exécution chaque fois que `ktrig` est positif et différent de sa valeur précédente. Utiliser une valeur constante de 1 pour n'imprimer qu'une fois à l'initialisation de la note.

## Exemples

Voici un exemple de l'opcode `puts`. Il utilise le fichier `puts.csd` [examples/puts.csd].

### Exemple 748. Exemple de l'opcode `puts`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o puts.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kcount init 440
```

```
ktrig metro 10
kcount = kcount + ktrig
String sprintfk "frequency in Hertz : %d \n", kcount
puts String, kcount
asig poscil .7, kcount, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 16384 10 1

i 1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
frequency in Hertz : 440
frequency in Hertz : 441
frequency in Hertz : 442
....
frequency in Hertz : 459
frequency in Hertz : 460
```

## Crédits

Auteur : Istvan Varga  
2005

# pvadd

pvadd — Lit un fichier *pvoc* et utilise ses données pour réaliser une synthèse additive.

## Description

*pvadd* lit un fichier *pvoc* et utilise ses données pour réaliser une synthèse additive au moyen d'un ensemble interne d'oscillateurs avec interpolation. L'utilisateur fournit la table d'onde (habituellement une période d'une onde sinusoïdale), et il peut choisir quels bins de l'analyse seront utilisés lors de la resynthèse.

## Syntaxe

```
ares pvadd ktmpnt, kfmmod, ifilcod, ifn, ibins [, ibinoffset] \  
          [, ibinincr] [, iextractmode] [, ifreqlim] [, igatefn]
```

## Initialisation

*ifilcod* -- entier ou chaîne de caractères dénotant un fichier de contrôle dérivé de l'analyse d'un signal audio par *pvanal*. Un entier dénote le suffixe d'un fichier *pvoc.m* ; une chaîne de caractères (entre guillemets) donne un nom de fichier, optionnellement un nom de chemin complet. Si ce n'est pas un nom de chemin complet, le fichier est d'abord cherché dans le répertoire courant, ensuite dans celui donné par la variable d'environnement *SADIR* (si elle est définie). Les fichiers de contrôle *pvoc* contiennent des données organisées pour la resynthèse par TFR. L'utilisation de la mémoire dépend de la taille des fichiers impliqués, qui sont lus et maintenus entièrement dans la mémoire durant les calculs, mais sont partagés par les appels multiples (voir aussi *lpread*).

*ifn* -- numéro de la table d'une fonction mémorisée contenant une onde sinusoïdale.

*ibins* -- nombre de bins utilisés dans la resynthèse (chaque bin compte pour un oscillateur dans la resynthèse).

*ibinoffset* (facultatif) -- est le premier bin utilisé (c'est une option et vaut 0 par défaut).

*ibinincr* (facultatif) -- fixe un incrément par lequel *pvadd* compte *ibins* composants à partir de *ibinoffset* pour la resynthèse (voir ci-dessous pour une explication plus détaillée).

*iextractmode* (facultatif) -- détermine s'il faut effectuer l'extraction spectrale et, dans ce cas, quels composants ayant des variations de fréquence sous *ifreqlim* ou au-dessus de *ifreqlim* seront écartés. Si *iextractmode* vaut 1, *pvadd* ne synthétise que les composants pour lesquels la différence de fréquence entre les trames d'analyse est plus grande que *ifreqlim*. Si *iextractmode* vaut 2, *pvadd* ne synthétise que les composants pour lesquels la différence de fréquence entre trames est plus petite que *ifreqlim*. *iextractmode* et *ifreqlim* valent 0 par défaut, ce qui provoque une simple resynthèse. Voir les exemples ci-dessous.

*igatefn* (facultatif) -- le numéro d'une fonction stockée à appliquer aux amplitudes des bins de l'analyse avant la resynthèse. Si *igatefn* est supérieur à 0, les amplitudes de chaque bin seront modifiées par *igatefn* par un procédé simple de correspondance. D'abord, les amplitudes de tous les bins dans toutes les trames de l'ensemble du fichier d'analyse sont comparées pour déterminer la valeur de l'amplitude maximale. Cette valeur est ensuite utilisée pour créer des amplitudes normalisées comme indices dans la fonction stockée *igatefn*. L'amplitude maximale correspond au dernier point dans la fonction. Une amplitude nulle correspond au premier point dans la fonction. Les valeurs entre 0 et 1 sont mises en correspondance avec les points tout au long de la table de fonction. Ceci est illustré dans les exemples ci-dessous.

## Exécution

*ktimpnt* et *kfmod* sont utilisés de la même manière que dans *pvoc*.

## Exemples

```
ptime line 0, p3, p3
asig pvadd ktime, 1, « oboe.pvoc », 1, 100, 2
```

Ci-dessus, *ibins* vaut 100 et *ibinoffset* vaut 2. Avec ces réglages, la resynthèse contiendra 100 composants commençant avec le bin n°2 (les bins sont comptés à partir de 0). Donc, la resynthèse sera réalisée avec les bins 2 à 101 inclus. Il est généralement avisé de commencer avec le bin 1 ou 2, car le bin 0, et souvent le bin 1, contiennent des données inutiles voire inefficaces pour la création d'une resynthèse propre.

```
ptime line 0, p3, p3
asig pvadd ktime, 1, « oboe.pvoc », 1, 100, 2, 2
```

L'exemple ci-dessus est le même que le précédent avec l'addition de la valeur 2 utilisée pour l'argument facultatif *ibinincr*. Le résultat comprend toujours 100 composants dans la resynthèse, mais *pvadd* compte les bins par 2 au lieu de 1. Il utilise ainsi les bins 2, 4, 6, 8, 10, et ainsi de suite. Avec *ibins*=10, *ibinoffset*=10 et *ibinincr*=10, *pvadd* utiliserait les bins 10, 20, 30, 40, jusqu'à 100 inclus.

Ci-dessous, un exemple utilisant l'extraction spectrale. Dans cet exemple, *iextractmode* vaut 1 et *ifreqlim* vaut 9. Ainsi, *pvadd* ne synthétise que les bins pour lesquels la moyenne de la déviation en fréquence prise sur 6 trames est supérieure à 9.

```
ptime line 0, p3, p3
asig pvadd ktime, 1, « oboe.pvoc », 1, 100, 2, 2, 1, 9
```

Si *iextractmode* avait eu pour valeur 2 dans l'exemple ci-dessus, seuls les bins avec une déviation en fréquence moyenne inférieure à 9 auraient été synthétisés. Avec de bons réglages, cette technique peut être utilisée pour séparer les parties à hauteur définie du spectre des parties bruiteuses. En pratique, cela dépend beaucoup du type de son, de la qualité de l'enregistrement et de la numérisation, et aussi de la taille de la fenêtre d'analyse et de l'incrément de trame.

L'exemple suivant utilise le mappage d'amplitude. Le dernier 2 dans la liste d'arguments indique f2 dans la partition.

```
asig pvadd ktime, 1, « oboe.pvoc », 1, 100, 2, 2, 0, 0, 2
```

En supposant que la partition contienne :

```
f2 0 512 7 0 256 1 256 1
```

Les bins dont l'amplitude est supérieure ou égale à 50% du maximum resteront inchangés, tandis que ceux dont l'amplitude est inférieure à 50% du maximum seront atténués. Dans ce cas, plus l'amplitude est faible et plus l'atténuation sera forte. Mais supposons que la partition contienne :

```
f2 0 512 5 1 512 .001
```

Dans ce cas, les amplitudes les plus faibles resteront inchangées tandis que les plus fortes seront atténuées, « inversant » le son en termes de spectre d'amplitude ! Les fonctions peuvent être arbitrairement com-

pliquées. Il faut simplement se souvenir que les valeurs d'amplitude de l'analyse normalisées sont elles-mêmes les indices dans la fonction.

Finalement, on peut utiliser de concert l'extraction spectrale et le mappage d'amplitude. L'exemple ci-dessous ne synthétise que les composants ayant une déviation en fréquence de moins de 5 Hz par trame et il pondère les amplitudes selon la table f2.

```
asig pvadd ktime, 1, « oboe.pvoc », 1, 100, 1, 1, 2, 5, 2
```



## REMARQUES UTILES

En utilisant plusieurs unités *pvadd* ensemble, il est possible de faire un fondu entre différentes parties de la resynthèse, ce qui crée des effets de « filtrage » variés. L'auteur utilise *pvadd* pour synthétiser un bin à la fois afin de contrôler séparément chaque composant de la resynthèse.

Si une combinaison de *ibins*, *ibinoffset* et *ibinincr* crée une situation où *pvadd* doit utiliser un numéro de bin supérieur au nombre de bins dans l'analyse, il n'utilisera que l'ensemble des bins disponibles, sans protester. Ainsi, pour utiliser tous les bins, il suffit de donner une grande valeur à *ibins* (par exemple 2000).

Il faut s'attendre dans tous les cas à augmenter les amplitudes d'un facteur compris entre 10 et 100.

Voici un exemple complet de l'opcode *pvadd*. Il utilise le fichier *pvadd.csd* [examples/pvadd.csd]

### Exemple 749. Exemple de l'opcode *pvadd*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvadd.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 1
nchnls = 2

instr 1
; analyze "fox.wav" with PVANAL first
igatefn = p4
ktime line 0, p3, p3
asig pvadd ktime, 1, "fox.pvx", 1, 300, 2, 2, 0, 0, igatefn
outs asig*3, asig*3

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave
f 2 0 512 5 1 256 .001
f 3 0 512 7 0 256 1 256 1

i 1 0 2.8 2
```

```
i 1 + 2.8 3  
e  
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : Richard Karpen  
Seattle, WA USA  
1998

Nouveau dans la version 3.48 de Csound, arguments supplémentaires dans la version 3.56



# pvbufread

pvbufread — Lit un fichier d'analyse par vocodeur de phase et rend accessibles les données récupérées.

## Description

*pvbufread* lit depuis un fichier *pvoc* et rend les données récupérées accessibles à toutes les unités *pvinterp* et *pvcross* suivantes qui surviennent dans un instrument avant un *pvbufread* consécutif (de la même façon que *lpread* et *lpreson* travaillent ensemble). Les données sont passées en interne et l'unité n'a pas de sortie propre.

## Syntaxe

```
pvbufread ktimpnt, ifile
```

## Initialisation

*ifile* -- le numéro *pvoc* (n dans *pvoc.n*) ou le nom entre guillemets du fichier d'analyse créé avec *pvanal*. (Voir *pvoc*.)

## Exécution

*ktimpnt* -- l'écoulement du temps, en secondes, dans ce fichier. *ktimpnt* doit toujours être positif, mais il peut avancer ou reculer dans le temps, être stationnaire ou discontinu, comme pointeur dans le fichier d'analyse.

## Exemples

L'exemple ci-dessous montre une utilisation de *pvbufread* avec *pvinterp* pour interpoler entre le son d'un hautbois et celui d'une clarinette. La valeur de *kinterp* retournée par l'opcode *linseg* est utilisée pour définir le déroulement temporel de la transition entre les deux sons. Les interpolations des fréquences et des amplitudes sont contrôlées par le même facteur dans cet exemple, mais il peut être intéressant de ne pas les synchroniser de cette manière pour obtenir d'autres effets. Cet exemple commence par un son de clarinette qui se transforme en hautbois et revient ensuite à la clarinette. *kfreqscale2* vaut 1.065 car dans ce cas le hautbois est plus haut d'un demi-ton que la clarinette et cela les met approximativement à la même hauteur. *kampscale2* vaut 0.75 car la clarinette analysée était un peu plus forte que le hautbois analysé. Les réglages de ces deux paramètres donnent une transition assez douce dans ce cas, mais de tels ajustements ne sont en aucun cas nécessaires ou même préconisés.

```
ktime1 line      0, p3, 3.5 ; used as index in the "oboe.pvoc" file
ktime2 line      0, p3, 4.5 ; used as index in the "clar.pvoc" file
kinterp linseg    1, p3*0.15, 1, p3*0.35, 0, p3*0.25, 0, p3*0.15, 1, p3*0.1, 1
pvbufread ktime1, "oboe.pvoc"
apv pvinterp ktime2,1,"clar.pvoc", 1, 1.065, 1, 0.75, 1-kinterp, 1-kinterp
```

Ci-dessous un exemple d'utilisation de *pvbufread* avec *pvcross*. Dans cet exemple les amplitudes utilisées dans la resynthèse évoluent graduellement de celles du hautbois à celles de la clarinette. Naturellement, les fréquences sont celles de la clarinette durant tout le processus car *pvcross* n'utilise pas les données de fréquence du fichier lut par *pvbufread*.

```
ktime1 line      0, p3, 3.5 ; used as index in the "oboe.pvoc" file
```

```

ktime2 line      0, p3, 4.5 ; used as index in the "clar.pvoc" file
kcross expon     0.001, p3, 1
      pvbufread ktime1, "oboe.pvoc"
apv      pvcross ktime2, 1, "clar.pvoc", 1-kcross, kcross

```

Voici un exemple complet de l'opcode `pvbufread`. Il utilise le fichier `pvbufread.csd` [examples/pvbufread.csd]

### Exemple 750. Exemple de l'opcode `pvbufread`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvbufread.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1
; analyze "fox.wav" and "flute.aiff" with PVANAL first
ktime1 line 0, p3, .8 ; use a part of "flute.pvx" file
ktime2 line 0, p3, 1.2 ; use a part of "beats.pvx" file
kcross expon .03, p3, 1
      pvbufread ktime1, "flute.pvx"
asig      pvcross ktime2, 1, "beats.pvx", 1-kcross, kcross
outs asig, asig

endin
</CsInstruments>
<CsScore>
i 1 0 3
i 1 + 10

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*pvcross, pvinterp, pvread, tableseg, tablexseg*

## Crédits

Auteur : Richard Karpen  
 Seattle, WA USA  
 1997

# pvcross

**pvcross** — Applique les amplitudes d'un fichier d'analyse par vocodeur de phase aux données d'un second fichier.

## Description

*pvcross* applique les amplitudes d'un fichier d'analyse par vocodeur de phase aux données d'un second fichier et réalise ensuite la resynthèse. Les données proviennent, comme décrit ci-dessus, d'une unité *pvbufread* appelée auparavant. Les deux arguments d'amplitude de taux-k sont utilisés pour pondérer les amplitudes de chaque fichier séparément avant qu'elles ne soient additionnées et utilisées dans la resynthèse (voir ci-dessous les explications plus détaillées). Les fréquences du premier fichier ne sont pas utilisées du tout dans le processus. Cette unité permet simplement de réaliser une synthèse croisée par l'application des amplitudes du spectre d'un signal aux fréquences d'un second signal. A la différence de *pvinterp*, *pvcross* permet l'utilisation du paramètre *ispecwp* comme dans *pvoc* et dans *vpvoc*.

## Syntaxe

```
ares pvcross ktimepnt, kfmod, ifile, kampscale1, kampscale2 [, ispecwp]
```

## Initialisation

*ifile* -- le numéro *pvoc* (n dans *pvoc.n*) ou le nom entre guillemets du fichier d'analyse créé avec *pvanal*. (Voir *pvoc*.)

*ispecwp* (facultatif, 0 par défaut) -- s'il est différent de zéro, l'opcode tente de préserver l'enveloppe spectrale tandis que le contenu fréquentiel est modifié par *kfmod*. La valeur par défaut est zéro.

## Exécution

*ktimepnt* -- l'écoulement du temps, en secondes, dans ce fichier. *ktimepnt* doit toujours être positif, mais il peut avancer ou reculer dans le temps, être stationnaire ou discontinu, comme pointeur dans le fichier d'analyse.

*kfmod* -- un facteur de transposition au taux de contrôle : la valeur 1 n'implique pas de transposition, 1.5 transpose vers l'aigu d'un quinte juste et 0.5 d'une octave vers le grave.

*kampscale1*, *kampscale2* -- utilisés pour mettre à l'échelle les amplitudes stockées dans chaque trame du fichier d'analyse par vocodeur de phase. *kampscale1* met à l'échelle les amplitudes des données du fichier lu par un *pvbufread* appelé précédemment. *kampscale2* met à l'échelle les amplitudes du fichier nommé par *ifile*.

Il est possible d'ajuster ces valeurs au moyen de ces arguments avant l'application de l'interpolation. Par exemple, si *file1* est beaucoup plus fort que *file2*, on peut vouloir diminuer les amplitudes de *file1* ou augmenter celles de *file2* avant l'interpolation. De même on peut ajuster les fréquences de chacun pour les rapprocher les unes des autres (ou bien les opposer, bien sûr !) avant d'effectuer l'interpolation.

## Exemples

Ci-dessous un exemple de l'utilisation de *pvbufread* avec *pvcross*. Dans cet exemple les amplitudes utilisées dans la resynthèse changent graduellement de celles d'un hautbois à celles d'une clarinette. Les fréquences, naturellement, restent celles de la clarinette durant tout le processus car *pvcross* n'utilise pas les données de fréquence du fichier lu par *pvbufread*.

```

ktime1 line 0, p3, 3.5 ; used as index in the "oboe.pvoc" file
ktime2 line 0, p3, 4.5 ; used as index in the "clar.pvoc" file
kcross expon 0.001, p3, 1
      pvbufread ktime1, "oboe.pvoc"
apv pvcross ktime2, 1, "clar.pvoc", 1-kcross, kcross

```

Voici un exemple complet de l'opcode pvcross. Il utilise le fichier *pvcross.csd* [exemples/pvcross.csd]

### Exemple 751. Exemple de l'opcode pvcross.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvcross.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1
; analyze "beats.wav", "flute.aiff" and "mary.wav" with PVANAL first
ktime1 line 0, p3, 2 ; used as index in the "beats.pvx" file
ktime2 line 0, p3, 2.6 ; used as index in the "flute.pvx" or "mary.pvx"
      pvbufread ktime1, "beats.pvx" ;take only amplitude from "beats.pvx"
if p4 = 0 then
asig pvcross ktime2, 1, "flute.pvx", 1, 0 ;and keep freqs of "flute.aiff"
asig = asig*.8 ;scale output
else
asig pvcross ktime2, 1, "mary.pvx", 1, 0 ;and keep freqs of "mary.wav"
asig = asig*.4 ;scale output
endif
      outs asig, asig

endin
</CsInstruments>
<CsScore>
i 1 0 3 0
i 1 + 3 1

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*pvbufread*, *pvinterp*, *pvread*, *tableseg*, *tablexseg*

## Crédits

Auteur : Richard Karpen  
 Seattle, Wash  
 1997

Nouveau dans la version 3.44

# pvinterp

*pvinterp* — Interpole entre les amplitudes et les fréquences de deux fichiers d'analyse par vocodeur de phase.

## Description

*pvinterp* interpole entre les amplitudes et les fréquences, bin par bin, de deux fichiers d'analyse par vocodeur de phase (l'un provenant d'une unité *pvbufread* appelée précédemment et l'autre étant spécifié dans la liste d'arguments), permettant des transitions définies par l'utilisateur entre les sons analysés. Il permet aussi une mise à l'échelle de l'ensemble des amplitudes et des fréquences de chaque fichier séparément avant le calcul des valeurs interpolées et leur envoi aux routines de resynthèse. L'argument *kfmod* dans *pvinterp* exécute sa mise à l'échelle des valeurs de fréquence sur le résultat de la mise à l'échelle individuelle suivie de l'interpolation si bien que cela agit comme une valeur de mise à l'échelle globale des nouvelles composantes de fréquence.

## Syntaxe

```
ares pvinterp ktimpnt, kfmod, ifile, kfreqscale1, kfreqscale2, \  
      kampscale1, kampscale2, kfreqinterp, kampinterp
```

## Initialisation

*ifile* -- le numéro *pvoc* (n dans *pvoc.n*) ou le nom entre guillemets du fichier d'analyse créé avec *pvanal*. (Voir *pvoc*.)

## Exécution

*ktimpnt* -- l'écoulement du temps, en secondes, dans ce fichier. *ktimpnt* doit toujours être positif, mais il peut avancer ou reculer dans le temps, être stationnaire ou discontinu, comme pointeur dans le fichier d'analyse.

*kfmod* -- un facteur de transposition au taux de contrôle : la valeur 1 n'implique pas de transposition, 1.5 transpose vers l'aigu d'un quinte juste et 0.5 d'une octave vers le grave.

*kfreqscale1*, *kfreqscale2*, *kampscale1*, *kampscale2* -- utilisés dans *pvinterp* pour mettre à l'échelle les fréquences et les amplitudes stockées dans chaque trame du fichier d'analyse par vocodeur de phase. *kfreqscale1* et *kampscale1* mettent à l'échelle les fréquences et les amplitudes des données du fichier lu par le *pvbufread* appelé précédemment (ces données sont passées en interne à l'unité *pvinterp*). *kfreqscale2* et *kampscale2* mettent à l'échelle les fréquences et les amplitudes des données du fichier nommé par *ifile* dans la liste d'arguments de *pvinterp* et lu par l'unité *pvinterp*.

Il est possible d'ajuster ces valeurs au moyen de ces arguments avant l'application de l'interpolation. Par exemple, si *file1* est beaucoup plus fort que *file2*, on peut vouloir diminuer les amplitudes de *file1* ou augmenter celles de *file2* avant l'interpolation. De même on peut ajuster les fréquences de chacun pour les rapprocher les unes des autres (ou bien les opposer, bien sûr !) avant d'effectuer l'interpolation.

*kfreqinterp*, *kampinterp* -- utilisés dans *pvinterp*, déterminent la distance d'interpolation entre les valeurs d'un fichier de vocodeur de phase et les valeurs d'un second fichier. Lorsque *kfreqinterp* vaut 1, les valeurs de fréquence sont toutes celles du premier fichier (lu par le *pvbufread*), mises ensuite à l'échelle par l'argument *kfreqscale1*. Lorsque *kfreqinterp* est nul, les valeurs de fréquence sont toutes celles du second fichier (lu par l'unité *pvinterp* elle-même), mises ensuite à l'échelle par *kfreqscale2*. Lorsque *kfreqinterp* se trouve

entre 0 et 1, les valeurs de fréquence sont calculées, bin par bin, comme le pourcentage entre chaque paire de fréquences (autrement dit,  $kfreqinterp=0.5$  met les valeurs de fréquence à mi-chemin entre les valeurs de l'ensemble des données du premier fichier et celles de l'ensemble des données du second fichier).

*kampinterp* travaille de la même manière sur les amplitudes des deux fichiers. Comme ces arguments sont de taux-k, les pourcentages peuvent changer dans le temps ce qui permet toutes sortes de transitions entre les sons.

## Exemples

L'exemple ci-dessous montre une utilisation de *pvbufread* avec *pvinterp* pour interpoler entre le son d'un hautbois et celui d'une clarinette. La valeur de *kinterp* retournée par l'opcode *linseg* est utilisée pour définir le déroulement temporel de la transition entre les deux sons. Les interpolations des fréquences et des amplitudes sont contrôlées par le même facteur dans cet exemple, mais il peut être intéressant de ne pas les synchroniser de cette manière pour obtenir d'autres effets. Cet exemple commence par un son de clarinette qui se transforme en hautbois et revient ensuite à la clarinette. *kfreqscale2* vaut 1.065 car dans ce cas le hautbois est plus haut d'un demi-ton que la clarinette et cela les met approximativement à la même hauteur. *kampscale2* vaut 0.75 car la clarinette analysée était un peu plus forte que le hautbois analysé. Les réglages de ces deux paramètres donnent une transition assez douce dans ce cas, mais de tels ajustements ne sont en aucun cas nécessaires ou même préconisés.

```

ktime1 line      0, p3, 3.5 ; used as index in the "oboe.pvoc" file
ktime2 line      0, p3, 4.5 ; used as index in the "clar.pvoc" file
kinterp linseg   1, p3*0.15, 1, p3*0.35, 0, p3*0.25, 0, p3*0.15, 1, p3*0.1, 1
          pvbufread ktime1, "oboe.pvoc"
apv          pvinterp ktime2,1,"clar.pvoc", 1, 1.065, 1, 0.75, 1-kinterp, 1-kinterp

```

Voici un exemple complet de l'opcode *pvinterp*. Il utilise le fichier *pvinterp.csd* [examples/pvinterp.csd]

### Exemple 752. Exemple de l'opcode *pvinterp*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvinterp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1
; analyze "fox.wav" and "flute.aiff" with PVANAL first
ktime1 line 0, p3, 2.8 ; used as index in the "fox.pvx" file
ktime2 line 0, p3, 3 ; used as index in the "flute.pvx" file
kinterp line 1, p3, 0
  pvbufread ktime1, "fox.pvx"
  asig pvinterp ktime2,1,"flute.pvx",.9, 3, .6, 1, kinterp,1-kinterp
    outs asig, asig
endin

```

```
</CsInstruments>
<CsScore>
i 1 0 3
i 1 + 10

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvbufread, pvcross, pvread, tableseg, tablexseg*

## Crédits

Auteur : Richard Karpen  
Seattle, Wash  
1997



# pvoc

pvoc — Implémente une reconstruction de signal au moyen d'un vocoder de phase basé sur la TFR.

## Description

Implémente une reconstruction de signal au moyen d'un vocoder de phase basé sur la TFR.

## Syntaxe

```
ares pvoc ktimpnt, kfmod, ifilcod [, ispecwp] [, iextractmode] \  
      [, ifreqlim] [, igatefn]
```

## Initialisation

*ifilcod* -- entier ou chaîne de caractères dénotant un fichier de contrôle dérivé de l'analyse d'un signal audio. Un entier dénote le suffixe d'un fichier *pvoc.m* ; une chaîne de caractères (entre guillemets) donne un nom de fichier, optionnellement un nom de chemin complet. Si ce n'est pas un nom de chemin complet, le fichier est d'abord cherché dans le répertoire courant, ensuite dans celui donné par la variable d'environnement *SADIR* (si elle est définie). Le contrôle *pvoc* contient des valeurs d'enveloppes d'amplitude et de fréquence définies par points, organisées pour une resynthèse par TFR. L'utilisation de la mémoire dépend de la taille des fichiers impliqués, qui sont lus et maintenus entièrement dans la mémoire durant les calculs, mais sont partagés par les appels multiples (voir aussi *lpread*).

*ispecwp* (facultatif) -- s'il est différent de zéro, l'opcode tente de préserver l'enveloppe spectrale tandis que le contenu fréquentiel est varié par *kfmod*. Vaut zéro par défaut.

*extractmode* (facultatif) -- détermine s'il faut effectuer l'extraction spectrale et, dans ce cas, quels composants ayant des variations de fréquence sous *ifreqlim* ou au-dessus de *ifreqlim* seront écartés. Si *extractmode* vaut 1, *pvoc* ne synthétise que les composants pour lesquels la différence de fréquence entre les trames d'analyse est plus grande que *ifreqlim*. Si *extractmode* vaut 2, *pvoc* ne synthétise que les composants pour lesquels la différence de fréquence entre trames est plus petite que *ifreqlim*. *extractmode* et *ifreqlim* valent 0 par défaut, ce qui provoque une simple resynthèse. Les exemples de la notice de *pvadd* montrent comment utiliser l'extraction spectrale.

*igatefn* (facultatif) -- le numéro d'une fonction stockée à appliquer aux amplitudes des bins de l'analyse avant la resynthèse. Si *igatefn* est supérieur à 0, les amplitudes de chaque bin seront modifiées par *igatefn* par un procédé simple de correspondance. D'abord, les amplitudes de tous les bins dans toutes les trames de l'ensemble du fichier d'analyse sont comparées pour déterminer la valeur de l'amplitude maximale. Cette valeur est ensuite utilisée pour créer des amplitudes normalisées comme indices dans la fonction stockée *igatefn*. L'amplitude maximale correspond au dernier point dans la fonction. Une amplitude nulle correspond au premier point dans la fonction. Les valeurs entre 0 et 1 sont mises en correspondance avec les points tout au long de la table de fonction. Les exemples de la notice de *pvadd* montrent comment utiliser le mappage d'amplitude.

## Exécution

*ktimpnt* -- l'écoulement du temps en secondes dans le fichier d'analyse. *ktimpnt* doit toujours être positif, mais il peut avancer ou reculer, rester stationnaire ou être discontinu, comme pointeur dans le fichier d'analyse.

*kfmod* -- un facteur de transposition au taux-k : une valeur de 1 signifie pas de transposition, 1.5 transpose vers le haut d'une quinte parfaite et 0.5 transpose vers le bas d'une octave.

*pvoc* implémente une reconstruction de signal au moyen d'un vocoder de phase basé sur la TFR. Les données de contrôle proviennent d'un fichier d'analyse précompilé avec un taux de trame connu.

Cette implémentation de *pvoc* a été écrite à l'origine par Dan Ellis. Elle est basée en partie sur le système de Mark Dolson, mais le concept de pré-analyse est nouveau. L'extraction spectrale et le mappage d'amplitude (nouveau dans la version 3.56 de Csound) ont été ajoutés par Richard Karpen en se basant sur les fonctions dans SoundHack par Tom Erbe.

## Exemples

Voici un exemple de l'opcode *pvoc*. Il utilise le fichier *pvoc.csd* [exemples/pvoc.csd].

### Exemple 753. Exemple de l'opcode *pvoc*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvoc.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1
; analyze "fox.wav" with PVANAL first
ispec = p4
ktime line 0, p3, 1.55
kfrq line .8, p3, 2
asig pvoc ktime, kfrq, "fox.pvx", ispec
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 6 0
i 1 + 6 1 ;preserve spectral envelope
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*vpvoc*, *PVANAL*.

## Crédits

Auteurs : Dan Ellis et Richard Karpen

Seattle, Wash  
1997

# pvread

**pvread** — Lit un fichier d'analyse par vocodeur de phase et retourne la fréquence et l'amplitude d'un canal d'analyse ou bin.

## Description

*pvread* lit depuis un fichier *pvoc* et retourne la fréquence et l'amplitude d'un canal d'analyse ou bin. Les valeurs retournées peuvent être utilisée à n'importe quel autre endroit de l'instrument de Csound. Par exemple, on peut les utiliser comme arguments d'un oscillateur pour synthétiser une composante d'un signal analysé ou on peut utiliser un banc de *pvreads* pour resynthétiser le son analysé en synthèse additive en passant les valeurs de fréquence et de magnitude à un banc d'oscillateurs.

## Syntaxe

```
kfreq, kamp pvread ktimepnt, ifile, ibin
```

## Initialisation

*ifile* -- le numéro *pvoc* (n dans *pvoc.n*) ou le nom entre guillemets du fichier d'analyse créé avec *pvanal*. (Voir *pvoc*.)

*ibin* -- le numéro du canal d'analyse duquel seront retournées la fréquence en Hz et la magnitude.

## Exécution

*kfreq*, *kamp* -- sorties de l'unité *pvread*. Ces valeurs, récupérées d'un fichier d'analyse par vocodeur de phase, représentent les valeurs de fréquence et d'amplitude d'un canal d'analyse spécifié par l'argument *ibin*. Une interpolation a lieu entre les trames d'analyse avec une résolution au taux-k et elle dépend bien sûr de la vitesse et de la direction de *ktimepnt*.

*ktimepnt* -- l'écoulement du temps, en secondes, dans ce fichier. *ktimepnt* doit toujours être positif, mais il peut avancer ou reculer dans le temps, être stationnaire ou discontinu, comme pointeur dans le fichier d'analyse.

## Exemples

L'exemple ci-dessous montre l'utilisation de *pvread* pour synthétiser un seul composant à la fois à partir d'un fichier d'analyse de vocodeur de phase. Il faut noter que l'on peut utiliser les sorties *kfreq* et *kamp* pour n'importe quel type de synthèse, de filtrage, de traitement, etc.

### Exemple 754. Exemple de l'opcode *pvread*.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvread.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1
; analyze "fox.wav" with PVANAL first
ibin = p4
ktime line 0, p3, 2.8
kfreq, kamp pvread ktime, "fox.pvx", ibin ;read data from 7th analysis bin.
asig poscil kamp, kfreq, 1 ;function 1 is a stored sine
outs asig*5, asig*5 ;compensate loss of volume

endin
</CsInstruments>
<CsScore>
;sine wave
f1 0 4096 10 1

i 1 0 6 7
i 1 + 6 15
i 1 + 2 25
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*pvbufread, pvcross, pvinterp, tableseg, tablexseg*

## Crédits

Auteur : Richard Karpen  
 Seattle, Wash  
 1997

Nouveau dans la version 3.44

# pvsadsyn

pvsadsyn — Resynthèse au moyen d'un banc d'oscillateurs rapide.

## Description

Resynthèse au moyen d'un banc d'oscillateurs rapide.

## Syntaxe

```
ares pvsadsyn fsrc, inoscs, kfmod [, ibinoffset] [, ibinincr] [, iinit]
```

## Initialisation

*inoscs* -- Le nombre de bins d'analyse à synthétiser. Ne peut pas être supérieur à la taille de *fsrc* (voir *pvsinfo*), par exemple comme celui créé par *pvsanal*. Le temps de traitement est directement proportionnel à *inoscs*.

*ibinoffset* (facultatif, 0 par défaut) -- Le premier bin (le plus bas) à resynthétiser, en comptant à partir de 0 (la valeur par défaut est 0).

*ibinincr* (facultatif) -- En partant du bin *ibinoffset*, l'intervalle entre les bins resynthétisés vaut *ibinincr*.

*iinit* (facultatif) -- Ignore la réinitialisation. N'est actuellement implémenté dans aucun de ces opcodes, et il reste à décider s'il serait de quelque utilité.

## Exécution

*kfmod* -- Facteur de multiplication pour toutes les fréquences. 1.0 = pas de changement, 2 = une octave vers l'aigu.

*pvsadsyn* est expérimental. Il implémente le banc d'oscillateurs en utilisant une méthode de calcul directe, plutôt qu'une table de consultation. On tire ainsi avantage du fait empirique que pour les taux d'analyse généralement pratiqués, (et en supposant que l'analyse se fait avec *pvsanal*, où les fréquences dans un bin ne changent que légèrement entre les trames), il n'est pas nécessaire d'interpoler les fréquences entre les trames, seulement les amplitudes. Une synthèse précise est souvent liée à l'utilisation de *pvsanal* avec *iwinsize = ifftsize\*2*.

Cet opcode changera très probablement ou sera bien étendu, selon le retour et les avis des utilisateurs. Il est probable qu'une méthode entièrement basée sur une table avec interpolation sera ajoutée, via un futur argument facultatif *iarg*. La liste des paramètres de *pvsadsyn* est calquée sur celle de *pvadd*, mais exclut l'extraction spectrale.

## Exemples

Voici un exemple de l'opcode *pvsadsyn*. Il utilise le fichier *pvsadsyn.csd* [exemples/pvsadsyn.csd].

### Exemple 755. Exemple de l'opcode pvsadsyn.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o pvsadsyn.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

opcode FileToPvsBuf, iik, Siiii
;;writes an audio file at the first k-cycle to a fft-buffer (via pvsbuffer)
Sfile, ifftsize, ioverlap, iwinsize, iwinshape xin
ktimek   timeinstk
if ktimek == 1 then
ilen   filelen Sfile
kcycles = ilen * kr; number of k-cycles to write the fft-buffer
kcount   init 0
loop:
ain   soundin Sfile
fftin   pvsanal ain, ifftsize, ioverlap, iwinsize, iwinshape
ibuf, ktim   pvsbuffer fftin, ilen + (ifftsize / sr)
    loop_lt kcount, 1, kcycles, loop
    xout   ibuf, ilen, ktim
endif
endop

instr 1
istretch = p4; time stretching factor
ifftsize = 1024
ioverlap = ifftsize / 4
iwinsize = ifftsize
iwinshape = 1; von-Hann window
ibuffer, ilen, k0   FileToPvsBuf "fox.wav", ifftsize, ioverlap, iwinsize, iwinshape
p3 = istretch * ilen; set p3 to the correct value
ktmpnt   linseg 0, p3, ilen; time pointer
fread   pvsbufread ktmpnt, ibuffer; read the buffer
aout   pvsadsyn fread, 10, 1; resynthesis with the first 10 bins
    out   aout
endin

</CsInstruments>
<CsScore>
i 1 0 1 20
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn*

## Crédits

Auteur : Richard Dobson

Août 2001

Nouveau dans la version 4.13



# pvsanal

**pvsanal** — Génère un fsig à partir d'une source audio mono, en utilisant l'analyse par recouvrement-addition d'un vocodeur de phase.

## Description

Génère un fsig à partir d'une source audio mono, en utilisant l'analyse par recouvrement-addition d'un vocodeur de phase.

## Syntaxe

```
fsig pvsanal ain, ifftsize, ioverlap, iwinsize, iwintype [, iformat] [, iinit]
```

## Initialisation

*ifftsize* -- La taille de la TFR en échantillons. Ne doit pas forcément être une puissance de deux (bien que celles-ci sont particulièrement efficaces), mais doit être paire. Les nombres impairs sont arrondis en interne. *ifftsize* détermine le nombre de bins d'analyse dans *fsig*, soit  $ifftsize/2 + 1$ . Par exemple, si *ifftsize* = 1024, *fsig* contiendra 513 bins d'analyse, ordonnés linéairement de la fréquence fondamentale à la fréquence de Nyquist. La fréquence fondamentale de l'analyse (qui donne en principe la fréquence résoluble la plus basse) est déterminée par  $sr/ifftsize$ . Ainsi, pour l'exemple précédent en supposant que  $sr = 44100$ , la fréquence fondamentale de l'analyse vaut 43.07Hz. En pratique, comme le vocodeur de phase préserve la phase, la fréquence de chaque bin peut dévier de façon bilatérale, si bien que des composantes continues sont enregistrées. Avec un signal fortement tonal, les fréquences des bins adjacents peuvent s'aggréger très étroitement autour des partiels de la source, et les bins inférieurs peuvent même avoir des fréquences négatives.

En principe, la seule raison d'utiliser pour *ifftsize* une valeur qui n'est pas une puissance de deux est de s'adapter à la fréquence fondamentale connue d'une source fortement tonale. Les valeurs décomposables en plusieurs petits facteurs peuvent être presque aussi efficaces que les tailles en puissance de deux ; par exemple : 384, pour une source dont la hauteur est proche du la grave à 110 Hz.

*ioverlap* -- La distance en échantillons (« taille du saut ») entre les trames d'analyse se recouvrant. En principe, doit valoir au moins  $ifftsize/4$ , par exemple 256 dans l'exemple ci-dessus. *ioverlap* détermine le taux d'analyse sous-jacent, soit  $sr/ioverlap$ . Il n'est pas nécessaire que *ioverlap* soit un facteur simple de *ifftsize* ; par exemple, une valeur de 160 sera légale. Le choix de *ioverlap* peut être dicté par l'importance de la modification de hauteur appliquée au *fsig*, s'il y en a une. En règle générale, plus la transposition est importante et plus le taux d'analyse doit être élevé, ce qui implique une plus petite valeur de *ioverlap*. Un taux d'analyse plus élevé peut aussi être plus avantageux avec des sons à transitoires à large bande tels que des tambours (pour lesquels une petite fenêtre d'analyse diminue l'étalement mais augmente le nombre d'erreurs relatives à la fréquence).

Noter qu'il est possible, et raisonnable, d'avoir différents fsigs dans un orchestre (même dans le même instrument), évoluant à différents taux d'analyse. Les interactions entre de tels fsigs ne sont pas couramment supportées et l'opcode d'affectation de fsig ne permet pas la copie entre fsigs ayant des propriétés différentes, même si la seule différence est la valeur de *ioverlap*. Cependant, ceci ne conduit pas à une impasse, car il est théoriquement possible d'effectuer une conversion grossière du taux (en particulier par rapport aux fichiers d'analyse en mémoire) comme on le fait dans les techniques du domaine temporel.

*iwinsize* -- la taille en échantillons du filtre de la fenêtre d'analyse (fixé par *iwintype*). Doit valoir au moins *ifftsize*, et peut être utilement plus grande. Bien que d'autres proportions soit permises, il est recommandé que *iwinsize* soit toujours un multiple entier de *ifftsize*, par exemple 2048 dans l'exemple ci-dessus.

En interne, la fenêtre d'analyse (Hamming, von Hann) est multipliée par une fonction sinc afin que les amplitudes soient nulles aux frontières de trame. La plus grande taille de fenêtre d'analyse s'est révélée particulièrement importante pour la resynthèse par banc d'oscillateurs (par exemple en utilisant *pvsadsyn*), car elle a pour effet d'augmenter la résolution en fréquence de l'analyse et ainsi, la précision de la resynthèse. Comme noté ci-dessus, *iwinsize* détermine la latence globale du système d'analyse/resynthèse. Dans bien des cas, et particulièrement en absence de transposition, on constate que l'égalité *iwinsize=ifftsize* fonctionne très bien et offre la latence la plus faible.

*iwintype* -- La forme de la fenêtre d'analyse. Actuellement, seulement trois choix sont implémentés :

- 0 = fenêtre de Hamming
- 1 = fenêtre de von Hann
- 3 = fenêtre de Kaiser (forme non glissante)

Elles sont aussi supportées par le format de fichier PVOC-EX. Le type de fenêtre est stocké comme attribut interne du fsig, avec les autres paramètres (voir *pvsinfo*). D'autres types pourront être implémentés dans le futur ; si la valeur de *iwintype* est strictement négative, sa valeur absolue est utilisée comme le numéro d'une ftable existante. Le problème ici est la contrainte de taille en puissance de deux des tables de fonction, ce qui en fait une solution incomplète. La plupart des utilisateurs jugeront que la fenêtre de Hamming est suffisante pour les besoins courants et qu'elle peut être considérée comme le choix par défaut.

*iformat* -- (facultatif) Le format d'analyse. Pour le moment un seul format est implémenté par cet opcode :

- 0 = amplitude + fréquence

C'est le format classique du vocodeur de phase ; facile à traiter, et naturel pour la resynthèse par banc d'oscillateurs. Il serait très facile (on pourrait dire tentant) de ne pas traiter une trame de fsig purement comme une trame de vocodeur de phase mais comme une trame de synthèse additive générique. Il est en fait possible d'utiliser un fsig de cette manière, mais il est important de garder à l'esprit que les deux ne sont pas, à strictement parler, directement équivalents.

D'autres formats importants (supportés par PVOC-EX) sont :

- 1 = amplitude + phase
- 2 = complexe (réel + imaginaire)

*iformat* est là au cas où il pourrait être utile par la suite de supporter ces autres formats. Les formats 0 et 1 sont en relation étroite (car la phase est « cyclique » dans les deux cas - il est trivial de convertir de l'un à l'autre), alors que le format complexe pourrait garantir un second type explicite de signal (un « csig ») spécialement pour les traitements à base de convolution, et d'autres traitements dans lesquels le complément des opérateurs arithmétiques peut être utile.

*iinit* -- (facultatif) Ignore la réinitialisation. N'est actuellement implémenté dans aucun de ces opcodes, et il reste à décider s'il serait de quelque utilité.



## Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

Voici un exemple de l'opcode pvsanal. Il utilise le fichier *pvsanal.csd* [examples/pvsanal.csd].

## Exemple 756. Exemple de l'opcode pvsanal.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvsanal.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;pvsanal has no influence when there is no transformation of original sound

ifftsize = p4
ioverlap = ifftsize / 4
iwinsize = ifftsize
iwinshape = 1 ;von-Hann window
Sfile = "fox.wav"
ain      soundin Sfile
fftin    pvsanal ain, ifftsize, ioverlap, iwinsize, iwinshape ;fft-analysis of the audio-signal
fftblur  pvscale fftin, p5 ;scale
aout     pvsynth fftblur ;resynthesis
outs     aout, aout

endin

</CsInstruments>
<CsScore>
s
i 1 0 3 512 1 ;original sound - ifftsize of pvsanal does not have any influence
i 1 3 3 1024 1 ;even with different
i 1 6 3 2048 1 ;settings

s
i 1 0 3 512 1.5 ;but transformation - here a fifth higher
i 1 3 3 1024 1.5 ;but with different settings
i 1 6 3 2048 1.5 ;for ifftsize of pvsanal

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Richard Dobson  
Août 2001

Nouveau dans la version 4.13

# pvsarp

pvsarp — Arpège les composantes spectrales d'un flux de signal pv.

## Description

Cet opcode arpège les composantes spectrales, en amplifiant un bin et en atténuant tous les autres autour de ce dernier. Utilisé avec un LFO il fournit un arpégiateur semblable au programme *specarp* de Trevor Wishart.

## Syntaxe

`fsig pvsarp fsigin, kbin, kdepth, kgain`

## Exécution

*fsig* -- flot pv de sortie

*fsigin* -- flot pv d'entrée

*kbin* -- bin cible, normalisé entre 0 et 1 (0Hz - Nyquist).

*kdepth* -- importance de l'atténuation des bins voisins

*kgain* -- renforcement du gain appliqué au bin cible



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

Voici un exemple de l'opcode pvsarp. Il utilise le fichier *pvsarp.csd* [exemples/pvsarp.csd]

### Exemple 757. Exemple de l'opcode pvsarp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      ;;-d      RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsarp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
```

```
ksmps = 100
nchnls = 1
0dbfs = 1

instr 1
asig in ; get the signal in
idepth = p4

fsig pvsanal asig, 1024, 256, 1024, 1 ; analyse it
kbin oscili 0.1, 0.5, 1 ; ftable 1 in the 0-1 range
ftps pvsarp fsig, kbin+0.01, idepth, 2 ; arpeggiate it (range 220.5 - 2425.5)
atps pvsynth ftps ; synthesise it

out atps
endin

</CsInstruments>
<CsScore>
f 1 0 4096 10 1 ;sine wave

i 1 0 10 0.9
i 1 + 10 0.5
e

</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode pvsarp. Il utilise le fichier *pvsarp2.csd* [examples/pvsarp2.csd]

### Exemple 758. Exemple de l'opcode pvsarp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out Audio in
-odac ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsarp2.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

instr 1
ifftsize = 1024
ioverlap = ifftsize / 4
iwinshape = ifftsize
iwinshape = 1; von-Hann window
Sfile1 = "fox.wav"
ain1 soundin Sfile1
fftin pvsanal ain1, ifftsize, ioverlap, iwinshape, iwinshape
;make 3 independently moving accentuations in the spectrum
kbin1 linseg 0.05, p3/2, .05, p3/2, .05
farp1 pvsarp fftin, kbin1, .9, 10
kbin2 linseg 0.075, p3/2, .1, p3/2, .075
```

```
farfp2 pvsarp fftin, kbin2, .9, 10
kbin3 linseg 0.02, p3/2, .03, p3/2, .04
farfp3 pvsarp fftin, kbin3, .9, 10
      ;resynthesize and add them
aout1 pvsynth farfp1
aout2 pvsynth farfp2
aout3 pvsynth farfp3
aout  = aout1*.3 + aout2*.3 + aout3*.3
      out  aout
      endin

</CsInstruments>
<CsScore>
i 1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn*

## Crédits

Auteur : Victor Lazzarini  
Avril 2005

Nouveau greffon dans la version 5

Avril 2005.

# pvsbandp

pvsbandp — Un filtre passe-bande travaillant dans le domaine spectral.

## Description

Filtre les trames pvoc, laissant passer les bins dont la fréquence se trouve dans une bande, avec interpolation linéaire pour les bandes de transition.

## Syntaxe

```
fsig pvsbandp fsigin, xlowcut, xlowfull, \
      xhighfull, xhighcut[, ktype]
```

## Exécution

*fsig* -- flot pv de sortie

*fsigin* -- flot pv d'entrée

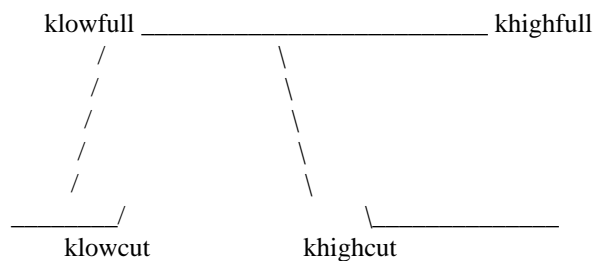
*xlowcut*, *xlowfull*, *xhighfull*, *xhighcut* -- définissent une forme de trapèze pour la bande passante. Les versions au taux-a ne s'appliquent qu'à la version glissante.

*ktype* -- spécifie la forme de la bande de transition. S'il prend la valeur par défaut de zéro, la forme est celle représentée ci-dessous, avec une transition linéaire d'amplitude. Les autres valeurs donnent une forme exponentielle :

$$(1 - \exp(-r \cdot \text{type})) / (1 - \exp(\text{type}))$$

Cela comprend une forme linéaire en dB lorsque *ktype* vaut  $\log(10)$  soit environ 2.30.

L'opcode réalise un filtre passe-bande avec une enveloppe spectrale formée comme ceci :



## Exemples

Voici un exemple de l'opcode pvsbandp. Il utilise le fichier *pvsbandp.csd* [examples/pvsbandp.csd].

### Exemple 759. Exemple de l'opcode pvsbandp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsbandp.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

instr 1
Sfile = "fox.wav"
klowcut = 100
klowfull = 200
khighfull = 1900
khighcut = 2000
ain soundin Sfile
fftin pvsanal ain, 1024, 256, 1024, 1; fft-analysis of the audio-signal
fftbp pvsbandp fftin, klowcut, klowfull, khighfull, khighcut ; band pass
abp pvsynth fftbp; resynthesis
out abp
endin

</CsInstruments>
<CsScore>
i 1 0 3
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn, pvsbandr*

## Crédits

Auteur : John ffitch  
 Décembre 2007



# pvsbandr

pvsbandr — Un filtre réjecteur de bande travaillant dans le domaine spectral.

## Description

Filtre les trames pvoc, rejetant les bins dont la fréquence se trouve dans une bande, avec interpolation linéaire pour les bandes de transition.

## Syntaxe

```
fsig pvsbandr fsigin, xlowcut, xlowfull, \  
      xhighfull, xhighcut[, ktype]
```

## Exécution

*fsig* -- flot pv de sortie

*fsigin* -- flot pv d'entrée

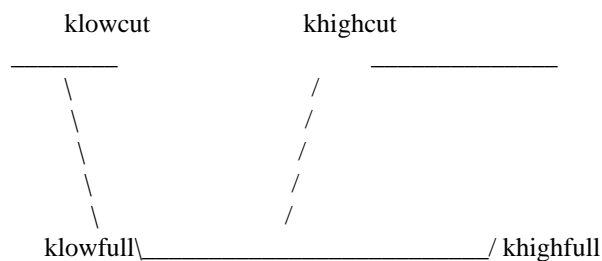
*xlowcut*, *xlowfull*, *xhighfull*, *xhighcut* -- définissent une forme de trapèze pour la bande rejetée. Les versions au taux-a ne s'appliquent qu'à la version glissante.

*ktype* -- spécifie la forme de la bande de transition. S'il prend la valeur par défaut de zéro, la forme est celle représentée ci-dessous, avec une transition linéaire d'amplitude. Les autres valeurs donnent une forme exponentielle :

$$(1 - \exp(r * \text{type})) / (1 - \exp(\text{type}))$$

Cela comprend une forme linéaire en dB lorsque *ktype* vaut  $\log(10)$  soit environ 2.30.

L'opcode réalise un filtre réjecteur de bande avec une enveloppe spectrale formée comme ceci :



## Exemples

Voici un exemple de l'opcode pvsbandr. Il utilise le fichier *pvsbandr.csd* [examples/pvsbandr.csd].

### Exemple 760. Exemple de l'opcode pvsbandr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o pvsbandr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

instr 1
Sfile = "fox.wav"
klowcut = 100
klowfull = 200
khighfull = 1900
khighcut = 2000
ain soundin Sfile
fftin pvsanal ain, 1024, 256, 1024, 1; fft-analysis of the audio-signal
fftbp pvsbandr fftin, klowcut, klowfull, khighfull, khighcut ; band reject
abp pvsynth fftbp; resynthesis
out abp
endin

</CsInstruments>
<CsScore>
i 1 0 3
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn, pvsbandp*

## Crédits

Auteur : John ffitch  
 Décembre 2007

# pvsbin

pvsbin — Obtient les valeurs d'amplitude et de fréquence d'un bin de signal PVS.

## Description

Obtient les valeurs d'amplitude et de fréquence d'un bin de signal PVS, comme variables de taux-k.

## Syntaxe

```
kamp, kfr pvsbin fsig, kbin
```

## Exécution

*kamp* -- amplitude du bin

*kfr* -- fréquence du bin

*fsig* -- flot d'entrée pv

*kbin* -- numéro du bin

## Exemples

Voici un exemple de l'opcode pvsbin. Il utilise le fichier *pvsbin.csd* [examples/pvsbin.csd]. Cet exemple utilise une entrée en temps réel, mais on peut aussi utiliser un fichier son en entrée.

### Exemple 761. Exemple de l'opcode pvsbin

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsbin.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
  ifftsize = 1024
  iwtype = 1    /* cleaner with hanning window */

  ;a1  soundin "input.wav"  ;select a soundifle
  a1 inch 1    ;Use realtime input

  fsig pvsanal  a1, ifftsize, ifftsize/4, ifftsize, iwtype
```

```
kamp, kfr pvsbin fsig, 10
adm oscil kamp, kfr, 1

    out adm
endin

</CsInstruments>
<CsScore>
; sine wave
f 1 0 4096 10 1

i 1 0 30
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn*

## Crédits

Auteur : Victor Lazzarini  
Août 2006

# pvsblur

pvsblur — Prend la moyenne des fonctions temporelles amp/fréq de chaque canal d'analyse sur une durée spécifiée.

## Description

Prend la moyenne des fonctions temporelles amp/fréq de chaque canal d'analyse sur une durée spécifiée (tronquée au nombre de trames). Il y a un effet de bord : le flot pvoc en entrée est retardé de cette durée.

## Syntaxe

```
fsig pvsblur fsigin, kblurtime, imaxdel
```

## Exécution

*fsig* -- flot pv de sortie

*fsigin* -- flot pv d'entrée

*kblurtime* -- durée en secondes pendant laquelle les valeurs moyennes des fenêtres seront prises.

*imaxdel* -- retard maximum, utilisé pour allouer la mémoire utilisée dans le calcul de la moyenne.

Cet opcode estompe un flot pvs en lissant les fonctions temporelles d'amplitude et fréquence (une sorte de filtrage passe-bas) ; l'importance de cet effet dépend de la longueur de la période sur laquelle est prise la moyenne, de plus grandes périodes donnant un effet plus prononcé.



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

Voici un exemple de l'opcode pvsblur. Il utilise le fichier *pvsblur.csd* [exemples/pvsblur.csd].

### Exemple 762. Exemple de l'opcode pvsblur.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
```

```
Odbfs = 1

;; example written by joachim heintz 2009

instr 1
ifftsize = 1024
ioverlap = ifftsize / 4
iwinsize = ifftsize
iwinshape = 1; von-Hann window
Sfile = "fox.wav"
ain soundin Sfile
fftin pvsanal ain, ifftsize, ioverlap, iwinsize, iwinshape; fft-analysis of the audio-signal
fftblur pvsblur fftin, p4, 1; blur
aout pvsynth fftblur; resynthesis
out aout
endin

</CsInstruments>
<CsScore>
i 1 0 3 0
i 1 3 3 .1
i 1 6 3 .5
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn*

## Crédits

Auteur : Victor Lazzarini

Novembre 2004

Nouveau greffon dans la version 5

Novembre 2004.

# pvsbuffer

pvsbuffer — Crée et écrit dans un tampon circulaire pour signaux-f (signaux PV en flot).

## Description

Cet opcode crée et écrit dans un tampon circulaire de longueur *ilen* (secondes), retournant un identifiant pour le tampon et un pointeur temporel qui contient la position courante d'écriture (également en secondes). On peut l'utiliser avec un ou plusieurs opcodes *pvsbufread*. L'écriture est circulaire, bouclant à la fin du tampon.

## Syntaxe

```
ihandle, ktime pvsbuffer fsig, ilen
```

## Initialisation

### Initialisation

*ihandle* -- identifiant pour ce tampon particulier, qui doit être passer à un opcode de lecture.

*ilen* -- longueur du tampon en secondes.

*fsig* -- un flot pv en entrée

*ktime* -- la position temporelle courante d'écriture dans le tampon.

*pvsbuffer* stocke *fsig* dans un tampon qui peut être lu par *pvsbufread* en utilisant l'identifiant *ihandle*. Différents tampons auront différents identifiants ce qui permet à différents opcodes *pvsbufread* de les lire indépendamment. *pvsbuffer* retourne la date courante (*ktime*) dans le tampon circulaire dans lequel il vient juste d'écrire.

## Exemples

Voir *pvsbufread* pour des exemples de l'opcode pvsbuffer.

## Voir aussi

*pvsbufread*

## Crédits

Auteur : Victor Lazzarini  
Juillet 2007

# pvsbufread

pvsbufread — Lit un tampon circulaire de signaux-f (signaux PV en flot).

## Description

Cet opcode lit à partir d'un tampon circulaire de longueur *ilen* (secondes), prenant un identificateur pour le tampon et un pointeur temporel qui conserve la position de lecture courante (aussi en secondes). Il est utilisé en conjonction avec un opcode *pvsbuffer*. La lecture est circulaire avec repliement à la fin du tampon.

## Syntaxe

```
fsig pvsbufread ktime, khandle[, ilo, ihi, iclear]
```

## Initialisation

*ilo, ihi* -- fixe les fréquences la plus basse et la plus haute à lire depuis le tampon (par défaut 0 et fréquence de Nyquist).

*iclear* -- fixé à 1 pour effacer le fsig de sortie avant chaque écriture (1 par défaut), fixé à 0 indique à l'opcode de ne pas effacer le fsig de sortie. Ceci est pertinent lorsque l'on écrit dans des sous-ensembles d'une trame de fsig en utilisant *ilo* et *ihi*.

## Exécution

*fsig* -- flot pv en sortie.

*ktime* -- position temporelle du pointeur de lecture (en secondes).

*khandle* -- identifiant du tampon à lire. Lorsque l'on utilise des identifiants de taux-k, il est important d'initialiser la variable de taux-k avec un identifiant existant. Lorsque l'on change de tampon, les différents tampons de fsig doivent être compatibles (même format de fsig).

Avec cet opcode et *pvsbuffer*, il est possible entre autres de :

- étirer/compresser dans le temps un flot fsig, en le lisant à différentes vitesses
- retarder un fsig ou certaines de ses parties.
- "brasser" deux ou plusieurs fsigs en alternant les tampons, car les identifiants de lecture sont de taux-k. Noter que lorsque l'on utilise des identifiants de taux-k, il est important d'initialiser la variable de taux-k avec un identifiant donné (afin que l'initialisation du fsig puisse avoir lieu) et on ne peut changer d'identifiant qu'entre des tampons de fsig compatibles (avec les mêmes taille de TFR et de recouvrement).



### Note

Il est important que la valeur de l'identifiant passé à *pvsbufread* soit celle d'un identifiant valide créé par *pvsbuffer*. Avec des identifiants non valides, Csound plantera.

## Exemples

Voici un exemple de l'opcode *pvsbufread*. Il effectue un "brassage" en alternant entre deux tampons.



### Exemple 763. Exemple de l'opcode pvsbufread

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
fsig1      pvsanal      asig1, 1024, 256, 1024, 1
fsig2      pvsanal      asig2, 1024, 256, 1024, 1

ibuf1, kt1  pvsbuffer    fsig1, 10      ; 10-sec buf with fsig1
ibuf2, kt2  pvsbuffer    fsig2, 7       ; 7-sec buf with fsig2

khan        init        ibuf1          ; initialise handle to buf1

if ktrig > 0 then                                ; switch buffers according to trigger
khan = ibuf2
else
khan = ibuf1
endif

fsb          pvsbufread  kt1, khan      ; read buffer
```

Voici un exemple de l'opcode pvsbufread. Il utilise le fichier *pvsbufread.csd* [examples/pvsbufread.csd].

### Exemple 764. Exemple de l'opcode pvsbufread.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsbufread.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

opcode FileToPvsBuf, iik, Siiii
;;writes an audio file at the first k-cycle to a fft-buffer (via pvsbuffer)
Sfile, ifftsize, ioverlap, iwinsize, iwinshape xin
ktimek timeinstk
if ktimek == 1 then
ilen filelen Sfile
kcycles = ilen * kr; number of k-cycles to write the fft-buffer
kcount init 0
loop:
ain soundin Sfile
fftin pvsanal ain, ifftsize, ioverlap, iwinsize, iwinshape
ibuf, ktim pvsbuffer fftin, ilen + (ifftsize / sr)
loop_lt kcount, 1, kcycles, loop
xout ibuf, ilen, ktim
endif
endop
```

```
instr 1
ifftsize = 1024
ioverlap = ifftsize / 4
iwinsize = ifftsize
iwinshape = 1; von-Hann window
ibuffer, ilen, k0 FileToPvsBuf "fox.wav", ifftsize, ioverlap, iwinsize, iwinshape
ktmpnt linseg ilen, p3, 0; reads the buffer backwards in p3 seconds
fread pvsbufread ktmpnt, ibuffer
aout pvsynth fread
out aout
endin

</CsInstruments>
<CsScore>
i 1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvsanal, pvsynth, pvsbuffer, pvsadsyn*

## Crédits

Auteur : Victor Lazzarini  
Juillet 2007

# pvsbufread2

pvsbufread2 — Lit un tampon circulaire de signaux-f (signaux PV en flot), avec des retards de bin additionnels.

## Description

Cet opcode lit à partir d'un tampon circulaire de longueur *ilen* (secondes), prenant un identificateur pour le tampon et un pointeur temporel qui conserve la position de lecture courante (aussi en secondes). Il est utilisé en conjonction avec un opcode *pvsbuffer*. La lecture est circulaire avec repliement à la fin du tampon. Des retards temporels supplémentaires proviennent d'une table de fonction, chaque point définissant un retard temporel en secondes affectant le bin correspondant.

## Syntaxe

```
fsig pvsbufread2 ktime, khandle, ift1, ift2
```

## Initialisation

*ift1* -- table de fonction d'au moins (taille de TFR)/2+1 points dans laquelle les délais (en secondes) pour les amplitudes de bin sont fixés (les positions dans la table de fonction sont équivalentes aux numéros de bin).

*ift2* -- table de fonction d'au moins (taille de TFR)/2+1 points dans laquelle les délais (en secondes) pour les fréquences de bin sont fixés (les positions dans la table de fonction sont équivalentes aux numéros de bin).

## Exécution

*fsig* -- flot pv en sortie.

*ktime* -- position temporelle du pointeur de lecture (en secondes).

*khandle* -- identifiant du tampon à lire. Lorsque l'on utilise des identifiants de taux-k, il est important d'initialiser la variable de taux-k avec un identifiant existant. Lorsque l'on change de tampon, les différents tampons de fsig doivent être compatibles (même format de fsig).

Avec cet opcode et *pvsbuffer*, il est possible entre autres de:

- étirer/compresser dans le temps un flot fsig, en le lisant à différentes vitesses
- retarder un fsig ou certaines de ses parties.
- "brasser" deux ou plusieurs fsigs en alternant les tampons, car les identifiants de lecture sont de taux-k. Noter que lorsque l'on utilise des identifiants de taux-k, il est important d'initialiser la variable de taux-k avec un identifiant donné (afin que l'initialisation du fsig puisse avoir lieu) et on ne peut changer d'identifiant qu'entre des tampons de fsig compatibles (avec les mêmes taille de TFR et de recouvrement).



### Note

Il est important que la valeur de l'identifiant passé à *pvsbufread2* soit celle d'un identifiant valide créé par *pvsbuffer*. Avec des identifiants non valides, Csound plantera.

## Exemples

Voici un exemple de l'opcode `pvsbufread2`. Il utilise le fichier `pvsbufread2.csd` [examples/pvsbufread2.csd].

### Exemple 765. Exemple de l'opcode `pvsbufread2`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>

<CsInstruments>
ksmps = 64
0dbfs = 1
nchnls = 2

instr 1
kcnt    init      0
ifftsize =      2048
ihop    =      ifftsize/4

a1      diskin2    "beats.wav", 1, 0, 1
a1      =          a1*0.5
fsig1   pvsanal    a1, ifftsize, ihop, ifftsize, 1
ih, kt   pvsbuffer  fsig1, 10

fsig2    pvsbufread2 kt, ih, 1, 1
fsig3    pvsbufread2 kt, ih, 2, 2

a2       pvsynth    fsig3
a3       pvsynth    fsig2

        outs        a2, a3

endin
</CsInstruments>

<CsScore>
f1 0 2048 -7 0 128 1.1 128 0.5 256 1.8 512 1.1 1024 0.1
f2 0 2048 -7 1 128 0.2 128 0.05 256 0.5 512 0.9 1024 0.1

i1 0 60
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvsanal, pvsynth, pvsbuffer, pvsadsyn*

## Crédits

Auteur: Victor Lazzarini  
Juillet 2011

# pvscale

pvscale — Met à l'échelle les composantes de fréquence d'un flot pv.

## Description

Met à l'échelle les composantes de fréquence d'un flot pv, ce qui provoque une transposition de hauteur. Les amplitudes peuvent être modifiées afin de préserver les formants.

## Syntaxe

```
fsg pvscale fsgin, kscal[, kkeepform, kgain, kcoefs]
```

## Exécution

*fsg* -- flot pv de sortie

*fsgin* -- flot pv d'entrée

*kscal* -- facteur de mise à l'échelle.

*kkeepform* -- tente de préserver les formants du signal d'entrée ; 0 : ne pas garder les formants ; 1 : conserve les formants en utilisant une méthode de cepstre décalé ; 2 : conserve les formants en utilisant une méthode avec une véritable enveloppe (vaut 0 par défaut).

*kgain* -- modification d'amplitude (1 par défaut).

*kcoefs* -- nombre de coefficients du cepstre utilisés pour la préservation des formants (vaut 80 par défaut).

La qualité de la transposition de hauteur sera améliorée par l'utilisation d'une fenêtre de Hanning dans l'analyse pvoc. La méthode 1 de préservation de formants est moins intensive que la méthode 2, qui, elle, pourrait ne pas convenir à une utilisation en temps réel.



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

### Exemple 766. Exemples

```
asig  in                                ; get the signal in

fsg   pvsanal    asig, 1024, 256, 1024, 1 ; analyse it
ftps  pvscale    fsg, 1.5, 1, 1          ; transpose it keeping formants
atps  pvsynth    ftps                    ; synthesise it

adp   delayr     0.1                      ; delay original signal
adel  deltapn    1024                      ; by 1024 samples
      delayw     asig
out   atps + adel                    ; add tranposed and original
```

L'exemple ci-dessus montre un harmoniseur vocal. Le délai est nécessaire pour aligner les signaux temporellement, car le traitement d'analyse-synthèse implique un délai de 1024 échantillons entre l'entrée d'analyse et la sortie de synthèse.

Voici un exemple de l'opcode `pvscale`. Il utilise le fichier `pvscale.csd` [examples/pvscale.csd].

### Exemple 767. Exemple de l'opcode `pvscale`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

instr 1
  ifftsize = 1024
  ioverlap = ifftsize / 4
  iwinsize = ifftsize
  iwinshape = 1; von-Hann window
  Sfile = "fox.wav"
  ain soundin Sfile
  fftin pvsanal ain, ifftsize, ioverlap, iwinsize, iwinshape; fft-analysis of the audio-signal
  fftblur pvscale fftin, p4, p5, p6; scale
  aout pvsynth fftblur; resynthesis
  out aout
endin

</CsInstruments>
<CsScore>
i 1 0 3 1 0 1; original sound
i 1 3 3 1.5 0 2; fifth higher without ...
i 1 6 3 1.5 1 2; ... and with different ...
i 1 9 3 1.5 2 5; ... kinds of formant preservation
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn*

## Crédits

Auteur : Victor Lazzarini  
Novembre 2004

Nouveau greffon dans la version 5

Novembre 2004.

# pvscent

pvscent — Calcule le centroïde spectral d'un signal.

## Description

Calcule le centroïde spectral d'un signal à partir de sa transformée de Fourier discrète.

## Syntaxe

```
kcent pvscent fsig
```

```
acent pvscent fsig
```

## Exécution

*kcent* -- le centroïde spectral

*acent* -- le centroïde spectral

*fsig* -- un flot pv en entrée

## Exemples

Voici un exemple de l'utilisation de l'opcode *pvscent*. Il utilise le fichier *pvscent.csd* [exemples/pvscent.csd].

### Exemple 768. Exemple de l'opcode *pvscent*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

giSine ftgen 0, 0, 4096, 10, 1

instr 1
irefrtm = p4; time for generating new values for the spectral centroid
ifftsize = 1024
ioverlap = ifftsize / 4
iwinshape = ifftsize
iwinshape = 1; von-Hann window
;Sfile = "flute-C-octave0.wav"
Sfile = "fox.wav"
ain soundin Sfile
```

```
fftin  pvsanal ain, ifftsize, ioverlap, iwinsize, iwinshape; fft-analysis of the audio-signal
ktrig  metro 1 / irefrtm
if ktrig == 1 then
kcenter pvscent fftin; spectral center
endif
aout  oscil .2, kcenter, giSine
      out  aout
endin

</CsInstruments>
<CsScore>
i 1 0 2.757 .3
i 1 3 2.757 .05
i 1 6 2.757 .005
i 1 9 2.757 .001
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn, pvspitch*

## Crédits

Auteur : John ffitch  
Mars 2005

Nouveau greffon dans la version 5.

Mars 2005.



# pvsceps

`pvsceps` — Calcule le cepstre d'une entrée pvs, avec coefficients de lifrage facultatifs.

## Description

## Syntaxe

```
keps[] pvsceps fsig[, icoefs]
```

## Initialisation

*icoefs* -- le nombre de coefficients retenus dans le cepstre en sortie. Par défaut, aucun coefficient n'est livré.

## Exécution

*keps[]* -- le cepstre en sortie, un tableau de taille  $N/2+1$ , où N est équivalent à la taille de le TFR du *fsig* en entrée.

*fsig* -- un flux pv en entrée.

## Exemples

Voici un exemple d'utilisation de l'opcode *pvsceps*. Il utilise le fichier *pvsceps.csd* [examples/pvsceps.csd].

### Exemple 769.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

See the sections *Real-time Audio* and *Command Line Flags* for more information on using command line flags.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

instr 1

a1 diskin "fox.wav",1,0,1
k1 randh 80, 2.5
a2 vco2 8, 220+k1
fsig pvsanal a1,1024,256,1024,1
fsig2 pvsanal a2,1024,256,1024,1
keps[] pvsceps fsig,30
kenv[] cepsinv keps
fenv tab2pvs r2c(kenv)
fvoc pvsfilter fsig2, fenv, 1
asig pvsynth fvoc

    out asig
endin
```

```
</CsInstruments>
<CsScore>
i1 0 60
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn, pvspitch*

## Crédits

Auteur : Victor Lazzarini  
2014

# pvscross

pvscross — Réalise une synthèse croisée entre deux sources fsig.

## Description

Réalise une synthèse croisée entre deux sources fsig.

## Syntaxe

```
fsig pvscross fsrc, fdest, kamp1, kamp2
```

## Exécution

L'opération de cet opcode est identique à celle de *pvcross* sauf que l'on utilise des *fsigs* plutôt que des fichiers d'analyse, et qu'il n'y a pas de préservation de l'enveloppe spectrale. Les amplitudes de *fsrc* et de *fdest* (en utilisant les facteurs d'échelle *kamp1* pour *fsrc* et *kamp2* pour *fdest*) sont appliquées aux fréquences de *fsrc*. *kamp1* et *kamp2* doivent rester dans l'intervalle entre 0 et 1.

Avec cet opcode, on peut réaliser une synthèse croisée sur une entrée audio temps réel, en utilisant *pvsanal* pour générer *fsrc* et *fdest*. Ils doivent avoir le même format.



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

Voici un exemple de l'opcode pvscross. Il utilise le fichier *pvscross.csd* [exemples/pvscross.csd].

### Exemple 770. Exemple de l'opcode pvscross.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

instr 1
  ipermut = p4 ; 1 = change order of soundfiles
  ifftsize = 1024
```

```
ioverlap = ifftsize / 4
iwinshape = ifftsize
iwinshape = 1 ; von-Hann window
Sfile1 = "fox.wav"
Sfile2 = "wave.wav"
ain1 = soundin:a(Sfile1)
ain2 = soundin:a(Sfile2)
fftin1 = pvsanal(ain1, ifftsize, ioverlap, iwinshape, iwinshape) ; fft-analysis of file 1
fftin2 = pvsanal(ain2, ifftsize, ioverlap, iwinshape, iwinshape) ; fft-analysis of file 2
ktrans = linseg(0, p3, 1) ; linear transition
if ipermut == 1 then
    fcross = pvscross(fftin2, fftin1, ktrans, 1 - ktrans)
else
    fcross = pvscross(fftin1, fftin2, ktrans, 1 - ktrans)
endif
aout = pvsynth(fcross)
out(aout)
endin

</CsInstruments>
<CsScore>
i 1 0 2.757 0 ; frequencies from fox.wav, amplitudes moving from wave to fox
i 1 3 2.757 1 ; frequencies from wav.wav, amplitudes moving from fox to wave
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn*

## Crédits

Auteur : Richard Dobson  
Août 2001

Novembre 2003. Grâce à Kanata Motohashi, le lien vers l'opcode *pvcross* a été fixé.

Nouveau dans la version 4.13

# pvsdemix

pvsdemix — Séparation spectrale de sources stéréo basée sur l'azimut.

## Description

Séparation spectrale de sources stéréo basée sur l'azimut, avec pour résultat l'annulation d'une spatialisation. Cet opcode implémente l'algorithme de Discrimination Azimutale et Resynthèse (ADReSS) développé par Dan Barry (Barry et Al. "Sound Source Separation Azimuth Discrimination and Resynthesis". DAFx'04, Univ. de Naples). La séparation de sources, ou démixage, est contrôlée par deux paramètres : une position d'azimut (*kpos*) et une largeur de sous-espace (*kwidth*). Le premier est utilisé pour localiser les crêtes spectrales des sources individuelles dans un mélange stéréo, tandis que le second élargit l'espace de recherche en incluant/excluant les crêtes autour de *kpos*. Ces deux paramètres peuvent être utilisés interactivement pour extraire les sources sonores d'un mélange stéréo. L'algorithme est particulièrement efficace avec les enregistrements de studio dans lesquels des instruments différents occupent des positions spatiales différentes ; c'est en fait un algorithme de d'annulation de spatialisation.



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Syntaxe

`fsig pvsdemix fleft, fright, kpos, kwidth, ipoints`

## Exécution

*fsig* -- flot pv de sortie

*fleft* -- flot pv d'entrée du canal de gauche.

*fright* -- flot pv d'entrée du canal de droite.

*kpos* -- la position centrale de la cible d'azimut qui sera démixée, de gauche à droite ( $-1 \leq kpos \leq 1$ ). C'est l'inverse d'un contrôleur de pan-pot.

*kwidth* -- la largeur du sous-espace d'azimut qui détermine le nombre de points autour de *kpos* qui seront utilisés dans le traitement de démixage. ( $1 \leq kwidth \leq ipoints$ )

*ipoints* -- nombre total de points discrets qui vont diviser chaque côté de l'image stéréo. Ceci affecte au final la résolution du traitement.

## Exemples

L'exemple ci-dessous prend une entrée stéréo et la passe dans un traitement de démixage qui révèle une source située à *ipos* +/- *iwidth* points. On peut contrôler ces paramètres en temps réel (par exemple en utilisant des widgets FLTK ou le MIDI) pour une recherche interactive de sources sonores.

### Exemple 771. Exemples

```
ifftsize = 1024
iwtype   = 1      /* cleaner with hanning window */
ipos     = -0.8   /* to the left of the stereo image */
iwidth   = 20     /* use peaks of 20 points around it */

al,ar    soundin  "sinput.wav"

flc      pvsanal  al, ifftsize, ifftsize/4, ifftsize, iwtype
frc      pvsanal  ar, ifftsize, ifftsize/4, ifftsize, iwtype
fdm      pvsdemix flc, frc, kpos, kwidth, 100
adm      pvsynth  fdm

outs     adm, adm
```

## Crédits

Auteur : Victor Lazzarini  
Janvier 2005

Nouveau greffon dans la version 5

Janvier 2005.

# pvsdiskin

pvsdiskin — Lit un canal sélectionné d'un fichier d'analyse PVOC-EX.

## Description

Crée un flot fsig en lisant un canal sélectionné d'un fichier d'analyse PVOC-EX, avec interpolation de trame.

## Syntaxe

```
fsig pvsdiskin SFname, ktscal, kgain[, ioffset, ichan]
```

## Initialisation

*Sfname* -- Nom du fichier d'analyse. Il doit avoir l'extension de fichier .pvx.

On peut générer un fichier PVOC-EX multi-canaux avec l'utilité *pvanal* étendue.

*ichan* -- (facultatif) Le canal à lire (en comptant à partir de 1). Vaut 1 par défaut.

*ioffset* -- (facultatif) Décalage à partir du début du fichier (secondes). Vaut 0 par défaut.

## Exécution

*ktscal* -- échelle temporelle, c'est-à-dire la vitesse du pointeur de lecture (1 pour la vitesse normale, valeurs négatives pour une lecture à l'envers,  $0 < ktscal < 1$  pour une lecture plus lente et  $ktscal > 1$  pour une lecture plus rapide).

*kgain* -- valeur du gain.

## Exemples

Voici un exemple de l'opcode pvsdiskin. Il utilise le fichier *pvsdiskin.csd* [examples/pvsdiskin.csd].

### Exemple 772. Exemple de l'opcode pvsdiskin.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvsdiskin.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 1
```

```
nchnls = 2

instr 1
; create a PVOC-EX (*.pvx) file with PVANAL first
ktscale line 1, p3, .05 ;change speed
fsigr pvdiskin "fox.pvx", ktscale, 1 ;read PVOC-EX file
aout pvsynth fsigr ;resynthesise it
outs aout, aout

endin
</CsInstruments>
<CsScore>

i 1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
Mai 2007  
Nouveau dans Csound 5.06



# pvsdisp

pvsdisp — Affiche le graphe amplitude/fréquence d'un signal PVS.

## Description

Cet opcode affiche un signal PVS fsig. Il utilise X11 ou une fenêtre FLTK si c'est possible, sinon (ou si l'option -g est positionnée) il affiche une approximation en caractères ASCII.

## Syntaxe

```
pvsdisp fsig[, ibins, iwtflg]
```

## Initialisation

*iprd* -- la période de *pvsdisp* en secondes.

*ibins* (facultatif, par défaut=tous les bins) -- affiche seulement *ibins* bins.

*iwtflg* (facultatif, par défaut=0) -- indicateur de pause. S'il est différent de zéro, chaque *pvsdisp* est maintenu en attendant une validation de l'utilisateur. La valeur par défaut est 0 (pas de pause).

## Exécution

*pvsdisp* -- affiche le signal PVS trame par trame.

## Exemples

Voici un exemple de l'opcode pvsdisp. Il utilise le fichier *pvsdisp.csd* [examples/pvsdisp.csd]. Cet exemple utilise une entrée temps réel, mais on peut aussi utiliser un fichier son en entrée.

### Exemple 773. Exemple de l'opcode pvsdisp

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o pvsdisp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
asig inch 1
```

```
;a1 soundin "input.wav" ;select a soundifle
fsig pvsanal asig, 1024,256, 1024, 1
pvsdisp fsig

endin

</CsInstruments>
<CsScore>

i 1 0 30
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvsanal, pvsynth, dispfft, print, pvsadsyn*

## Crédits

Auteur : Victor Lazzarini, 2006

# pvsfilter

pvsfilter — Multiplie les amplitude d'un flot pvoc par celles d'un second flot pvoc, avec mise à l'échelle dynamique.

## Description

Multiplie les amplitude d'un flot pvoc par celles d'un second flot pvoc, avec mise à l'échelle dynamique.

## Syntaxe

```
fsig pvsfilter fsigin, fsigfil, kdepth[, igain]
```

## Exécution

*fsig* -- flot pv de sortie

*fsigin* -- flot pv d'entrée

*fsigfil* -- flot pvoc filtrant

*kdepth* -- contrôle l'importance du filtrage de *fsigin* par *fsigfil*.

*igain* -- modification de l'amplitude (facultatif, 1 par défaut).

Ici les amplitudes du flot pvoc en entrée sont modifiées par le flot filtrant sans changer les fréquences. Comme d'habitude, les deux signaux doivent avoir le même format.



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

### Exemple 774. Exemples

```
kfreq  expon 500, p3, 4000          ; 3-octave sweep
kdepth  linseg 1, p3/2, 0.5, p3/2, 1 ; varying filter depth

asig  in                ; input
afil  oscili 1, kfreq, 1    ; filter t-domain signal

fim    pvsanal asig,1024,256,1024,0 ; pvoc analysis
fil    pvsanal afil,1024,256,1024,0
fou    pvsfilter fim, fil, kdepth    ; filter signal
aout   pvsynth fou              ; pvoc synthesis
```

Dans l'exemple ci-dessus la courbe du filtre dépendra de l'enveloppe spectrale de *afil* ; dans le cas d'une simple sinusoïde, il sera équivalent à un filtre passe-bande à bande étroite.

Voici un exemple de l'opcode `pvsfilter`. Il utilise le fichier `pvsfilter.csd` [exemples/pvsfilter.csd].

### Exemple 775. Exemple de l'opcode `pvsfilter`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

giSine ftgen 0, 0, 4096, 10, 1
giBell ftgen 0, 0, 4096, 9, .56, 1, 0, .57, .67, 0, .92, 1.8, 0, .93, 1.8, 0, 1.19, 2.67, 0, 1.7, 1.6

instr 1
  ipermut = p4; 1 = change order of soundfiles
  ifftsize = 1024
  ioverlap = ifftsize / 4
  iwinsize = ifftsize
  iwinshape = 1; von-Hann window
  Sfile1 = "fox.wav"
  ain1 soundin Sfile1
  kfreq randomi 200, 300, 3
  ain2 oscili .2, kfreq, giBell
  ;ain2 oscili .2, kfreq, giSine; try also this
  fftin1 pvsanal ain1, ifftsize, ioverlap, iwinsize, iwinshape; fft-analysis of file 1
  fftin2 pvsanal ain2, ifftsize, ioverlap, iwinsize, iwinshape; fft-analysis of file 2
  if ipermut == 1 then
    fcross pvsfilter fftin2, fftin1, 1
  else
    fcross pvsfilter fftin1, fftin2, 1
  endif
  aout pvsynth fcross
  out aout * 20
endin

</CsInstruments>
<CsScore>
i 1 0 2.757 0; frequencies from fox.wav, amplitudes multiplied by amplitudes of giBell
i 1 3 2.757 1; frequencies from giBell, amplitudes multiplied by amplitudes of fox.wav
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn*

## Crédits

Auteur : Victor Lazzarini  
Novembre 2004

Nouveau greffon dans la version 5

Novembre 2004.

# pvsfread

pvsfread — Lit un canal sélectionné d'un fichier d'analyse PVOC-EX.

## Description

Crée un flot fsig en lisant un canal sélectionné d'un fichier d'analyse PVOC-EX chargé en mémoire, avec interpolation de trame. Seuls les fichiers au format 0 (amplitude + fréquence) sont actuellement supportés. L'opération de cet opcode reflète celle de *pvoc*, mais en restituant un fsig au lieu d'un signal resynthétisé.

## Syntaxe

```
fsig pvsfread ktimpt, ifn [, ichan]
```

## Initialisation

*ifn* -- Nom du fichier d'analyse. Il doit avoir l'extension de fichier .pvx.

On peut générer un fichier PVOC-EX multi-canaux avec l'*pvanal utility* étendue.

*ichan* -- (facultatif) Le canal à lire (en comptant à partir de 0). Vaut 0 par défaut.

## Exécution

*ktimpt* -- pointeur temporel dans le fichier d'analyse, en secondes. Voir la description du même paramètre de *pvoc* pour son utilisation.

Noter que les fichiers d'analyse peuvent être très grands, surtout s'ils sont multi-canaux. La lecture de tels fichiers en mémoire provoquera probablement des coupures audio durant une exécution en temps réel. Comme le fichier n'est lu qu'une fois, étant ensuite disponible pour tous les opcodes intéressés, il peut être opportun de disposer d'un instrument dédié au préchargement de ces fichiers d'analyse au démarrage.

## Exemples

Voici un exemple de l'opcode pvsfread. Il utilise le fichier *pvsfread.csd* [examples/pvsfread.csd].

### Exemple 776. Exemple de l'opcode pvsfread.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvsfread.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
```

```
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1
; create a PVOC-EX (*.pvx) file with PVANAL first
idur filelen "kickroll.pvx" ;find duration of (stereo) analysis file
kpos line 0,p3,idur ;to ensure we process whole file
fsigr pvsfread kpos,"kickroll.pvx", 1 ;create fsig from right channel
aout pvsynth fsigr ;resynthesise it
outs aout, aout

endin
</CsInstruments>
<CsScore>

i 1 0 10
i 1 11 1
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Richard Dobson  
Août 2001

Nouveau dans la version 4.13

# pvsfreeze

pvsfreeze — Gèle les fonctions temporelles d'amplitude et de fréquence d'un flot pv selon un déclencheur au taux de contrôle.

## Description

cet opcode "gèle" l'évolution d'un flot pvs en figeant les valeurs d'amplitude et/ou de fréquence de chaque bin. Le gel des valeurs est contrôlé indépendamment pour les amplitudes et pour les fréquences par un déclencheur au taux de contrôle qui active le gel s'il est supérieur ou égal à 1 et le désactive s'il est inférieur à 1.

## Syntaxe

fsig **pvsfreeze** fsigin, kfreeza, kfreezf

## Exécution

*fsig* -- flot pv de sortie.

*fsigin* -- flot pv d'entrée.

*kfreeza* -- contrôle du gel des amplitudes. Le gel est actif pour les valeurs supérieures ou égales à 1 et inactif pour les valeurs inférieures à 1.

*kfcf* -- contrôle du gel des fréquences. Le gel est actif pour les valeurs supérieures ou égales à 1 et inactif pour les valeurs inférieures à 1.



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

### Exemple 777. Exemples

```
asig in                                ; input
ktrig oscil      1.5, 0.25, 1          ; trigger
fim pvsanal      asigl, 1024, 256, 1024, 0 ; pvoc analysis
fou pvsfreeze    fim, abs(ktrig), abs(ktrig) ; regular 'freeze' of spectra
aout pvsynth     fou                    ; pvoc synthesis
```

Dans l'exemple ci-dessus, le signal d'entrée sera régulièrement "gelé" un court instant lorsque le déclencheur dépasse 1, environ toutes les deux secondes.

Voici in exemple de l'opcoce pvsfreeze. Il utilise le fichier *pvsfreeze.csd* [exemples/pvsfreeze.csd].

### Exemple 778. Exemple de l'opcoce pvsfreeze.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.



```

<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
Odbfs = 1

;; example written by joachim heintz 2009

seed 0

instr 1
ifftsize = 1024
ioverlap = ifftsize / 4
iwinsize = ifftsize
iwinshape = 1; von-Hann window
Sfile1 = "fox.wav"
ain soundin Sfile1
kfreq randomh .7, 1.1, 3; probability of freezing freqs: 1/4
kamp randomh .7, 1.1, 3; idem for amplitudes
fftin pvsanal ain, ifftsize, ioverlap, iwinsize, iwinshape; fft-analysis of file
freeze pvsfreeze fftin, kamp, kfreq; freeze amps or freqs independently
aout pvsynth freeze; resynthesize
out aout
endin

</CsInstruments>
<CsScore>
r 10
i 1 0 2.757
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn*

## Crédits

Auteur : Victor Lazzarini  
Mai 2006

Nouveau greffon dans la version 5

Mai 2006.

# pvsftr

pvsftr — Lit les données d'amplitude et/ou de fréquence depuis des tables de fonction.

## Description

Lit les données d'amplitude et/ou de fréquence depuis des tables de fonction.

## Syntaxe

```
pvsftr fsrc, ifna [, ifnf]
```

## Initialisation

*ifna* -- Une table d'une taille d'au moins *inbins* qui contient les données d'amplitude. Ignorée si *ifna* = 0.

*ifnf* (facultatif) -- Une table d'une taille d'au moins *inbins* qui contient les données de fréquence. Ignorée si *ifnf* = 0.

## Exécution

*fsrc* -- une source au format PVOC-EX.

Permet d'échanger le contenu de *fsrc* avec des tables de fonction pour un traitement particulier. Sauf si le recouvrement de trame est égal à *ksmps* (ce qui ne sera généralement pas le cas), les données de trame ne sont pas mises à jour à chaque période de contrôle. Il ne faut traiter les données contenues dans *ifna*, *ifnf* que lorsque *kflag* vaut 1. Pour ne traiter que les données de fréquence, mettre *ifna* à zéro.

Comme les tables de fonction ne servent qu'à stocker des données venant de *fsrc*, il n'y a aucun avantage à les définir dans la partition et elles seront généralement créées dans l'instrument avec *ftgen*.

En exportant disons les données d'amplitude d'un *fsig* et en les important dans un autre, on peut effectuer une synthèse croisée basique (comme dans *pvsftr*), avec l'option de modifier les données au passage en utilisant les opcodes de manipulation de table.

Noter que le format des données dans le *fsig* source n'est pas écrit dans les tables. Nous avons ainsi un moyen de transférer les données d'amplitude et de fréquence entre des *fsigs* de format différent. Utilisés de cette manière, ces opcodes deviennent potentiellement pathologiques et l'on peut escompter qu'ils donneront des résultats inattendus. Dans ce cas, la resynthèse avec *pvsadsyn* sera presque certainement requise.

Pour faire une copie directe d'un *fsig* à un autre de même format, on peut utiliser la syntaxe d'affectation conventionnelle :

```
fsig1 = fsig2
```

Il n'est pas nécessaire d'utiliser des tables de fonction dans ce cas.

## Exemples

Voici un exemple de l'opcode pvsftr. Il utilise le fichier *pvsftr.csd* [examples/pvsftr.csd].

## Exemple 779. Exemple de l'opcode pvsftr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvsftr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gifil ftgen 0, 0, 0, -1, "fox.wav", 0, 0, 1

instr 1

ifftsize = 1024    ;fft size
ioverlap = 256     ;overlap
knewamp = 0        ;new value for amplitudes

;create fsig stream from function table
fsrc pvstanal 1, 1, 1, gifil, 0, 0, 0, ifftsize, ioverlap, 0
ifn ftgen 0, 0, ifftsize/2, 2, 0 ;create empty function table
kflag pvsftw fsrc,ifn ;export amps to table
;overwrite the first 10 bins each time the table has been filled new
if kflag == 1 then
kndx = 0
kmaxbin = 10
loop:
tablew knewamp, kndx, ifn
loop_le kndx, 1, kmaxbin, loop
endif
pvsftr fsrc,ifn ;read modified data back to fsrc
aout pvsynth fsrc ;and resynth
outs aout, aout

endin
</CsInstruments>
<CsScore>
i 1 0 4
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvsftw*

## Crédits

Auteur : Richard Dobson  
Août 2001

Nouveau dans la version 4.13

# pvsftw

pvsftw — Ecrit les données d'amplitude et/ou de fréquence dans des tables de fonction.

## Description

Ecrit les données d'amplitude et/ou de fréquence dans des tables de fonction.

## Syntaxe

```
kflag pvsftw fsrc, ifna [, ifnf]
```

## Initialisation

*ifna* -- Une table d'une taille d'au moins *inbins* qui contient les données d'amplitude. Ignorée si *ifna* = 0.

*ifnf* -- (facultatif) -- Une table d'une taille d'au moins *inbins* qui contient les données de fréquence. Ignorée si *ifnf* = 0.

## Exécution

*kflag* -- Un indicateur qui prend la valeur 1 lorsque de nouvelles données sont disponibles, 0 sinon.

*fsrc* -- une source au format PVOC-EX.

Permet d'échanger le contenu de *fsrc* avec des tables de fonction pour un traitement particulier. Sauf si le recouvrement de trame est égal à *ksmps* (ce qui ne sera généralement pas le cas), les données de trame ne sont pas mises à jour à chaque période de contrôle. Il ne faut traiter les données contenues dans *ifna*, *ifnf* que lorsque *kflag* vaut 1. Pour ne traiter que les données de fréquence, mettre *ifna* à zéro.

Comme les tables de fonction ne servent qu'à stocker des données venant de *fsrc*, il n'y a aucun avantage à les définir dans la partition et elles seront généralement créées dans l'instrument avec *ftgen*.

En exportant disons les données d'amplitude d'un fsig et en les important dans un autre, on peut effectuer une synthèse croisée basique (comme dans *pvcross*), avec l'option de modifier les données au passage en utilisant les opcodes de manipulation de table.

Noter que le format des données dans le fsig source n'est pas écrit dans les tables. Nous avons ainsi un moyen de transférer les données d'amplitude et de fréquence entre des fsigs de format différent. Utilisés de cette manière, ces opcodes deviennent potentiellement pathologiques et l'on peut escompter qu'ils donneront des résultats inattendus. Dans ce cas, la resynthèse avec *pvsadsyn* sera presque certainement requise.

Pour faire une copie directe d'un fsig à un autre de même format, on peut utiliser la syntaxe d'affectation conventionnelle :

```
fsig1 = fsig2
```

Il n'est pas nécessaire d'utiliser des tables de fonction dans ce cas.

## Exemples

Voici un exemple de l'opcode `pvsftw`. Il utilise le fichier `pvsftw.csd` [examples/pvsftw.csd].

### Exemple 780. Exemple de l'opcode `pvsftw`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvsftw.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

inbins = 512
ifn ftgen 0,0,inbins,10,1 ; make ftable
fsrc pvdiskin "fox.pvx", 1, 1 ; read PVOCEX file
kflag pvsftw fsrc,ifn ; export amps to table,
kamp init 0
if kflag==0 kgoto contin ; only proc when frame is ready
tablew kamp,1,ifn ; kill lowest bins, for obvious effect
tablew kamp,2,ifn
tablew kamp,3,ifn
tablew kamp,4,ifn
; read modified data back to fsrc
pvsftr fsrc,ifn
contin:
; and resynth
aout pvsynth fsrc
outs aout, aout

endin

</CsInstruments>
<CsScore>

i 1 0 4
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvsftr*

## Crédits

Auteur : Richard Dobson

Août 2001

Nouveau dans la version 4.13

# pvsfwrite

pvsfwrite — Ecrit un signal fsig dans un fichier PVOCEX.

## Description

Cet opcode écrit un signal fsig dans un fichier PVOCEX (qui peut être lu à son tour par *pvsfread* ou par d'autres programmes qui supportent les fichiers PVOCEX en entrée).

## Syntaxe

```
pvsfwrite fsig, ifile
```

## Initialisation

*fsig* -- données du fsig en entrée. *ifile* -- nom du fichier (une chaîne de caractères entre guillemets) .

## Exemples

Voici un exemple de l'opcode pvsfwrite. Il utilise le fichier *pvsfwrite.csd* [examples/pvsfwrite.csd]. Cet exemple utilise une entrée temps réel.

### Exemple 781. Exemple de l'opcode pvsfwrite

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsfwrite.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

;By Victor Lazzarini 2008

instr 1
asig oscili 10000, 440, 1
fss pvsanal asig, 1024,256,1024,0
pvsfwrite fss, "mypvs.pvx"
ase pvsynth fss
      out ase
endin

instr 2 ; must be called after instr 1 finishes
ktim timeinsts
fss pvsfread ktim, "mypvs.pvx"
asig pvsynth fss
```

```
    out asig
  endin

</CsInstruments>
<CsScore>
f1 0 16384 10 1
i1 0 1
i2 1 1
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvsanal*, *pvsynth*, *pvsadsyn*

## Crédits

Auteur : Victor Lazzarini  
Novembre 2004

Nouveau greffon dans la version 5  
Novembre 2004.



# pvsgain

pvsgain — Met à l'échelle l'amplitude d'un flot pv.

## Description

Met à l'échelle l'amplitude d'un flot pv.

## Syntaxe

`fsig pvsgain fsigin, kgain`

## Exécution

*fsig* -- flot pv de sortie.

*fsigin* -- flot pv d'entrée.

*kgain* -- mise à l'échelle de l'amplitude (1 par défaut).



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

Voici un exemple de l'opcode pvsgain. Il utilise le fichier *pvsgain.csd* [exemples/pvsgain.csd].

### Exemple 782. Exemple de l'opcode pvsgain.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvsgain.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kgain = p4
asig diskin2 "beats.wav", 1
```

```
fsig pvsanal asig, 1024, 256, 1024, 1; analyse it
ftps pvsgain fsig, kgain ; amplify it
atps pvsynth ftps ; synthesise it
outs      atps, atps

endin
</CsInstruments>
<CsScore>

i1 0 2 .5
i1 + 2 1

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn*

## Crédits

Auteur : Victor Lazzarini  
2011

Nouveau greffon dans la version 5.14

2011

# pvshift

pvshift — Décale les composantes de fréquence d'un flot pv, étirant/compressant son spectre.

## Description

Décale les composantes de fréquence d'un flot pv, étirant/compressant son spectre.

## Syntaxe

```
fsig pvshift fsign, kshift, klowest[, kkeepform, igain, kcoefs]
```

## Exécution

*f`sig`* -- flot pv de sortie

*f`sign`* -- flot pv d'entrée

*k`shift`* -- quantité de décalage (en Hz, positive ou négative).

*k`lowest`* -- fréquence la plus basse à décaler.

*k`keepform`* -- tente de préserver les formants du signal d'entrée ; 0 : ne pas garder les formants ; 1 : conserve les formants en utilisant une méthode de cepstre décalé ; 2 : conserve les formants en utilisant une méthode avec une véritable enveloppe (vaut 0 par défaut).

*k`gain`* -- modification d'amplitude (1 par défaut).

*k`coefs`* -- nombre de coefficients du cepstre utilisés pour la préservation des formants (vaut 80 par défaut).

Cet opcode décale les composantes d'un flot pv à partir d'une certaine fréquence, vers le haut ou vers le bas, d'une quantité fixe (en Hz). On peut l'utiliser pour transformer un spectre harmonique en un spectre inharmonique. L'indicateur *k`keepform`* peut être utilisé pour essayer de préserver les formants pour des modifications du spectre intéressantes et inhabituelles.



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

### Exemple 783. Exemples

```
asig in ; get the signal in

fsig pvsanal asig, 1024, 256, 1024, 1 ; analyse it
ftps pvsshift fsig, 100, 0 ; add 100 Hz to each component
atps pvsynth ftps ; synthesise it
```

En fonction de l'entrée, ceci transformera un son à hauteur définie en un son inharmonique comme celui d'une cloche.

Voici un exemple de l'opcode `pvshift`. Il utilise le fichier `pvshift.csd` [exemples/pvshift.csd].

### Exemple 784. Exemple de l'opcode `pvshift`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

instr 1
ishift = p4; shift amount in Hz
ilowest = p5; lowest frequency to be shifted
ikeepform = p6; 0=no formant keeping, 1=keep by amps, 2=keep by spectral envelope
ifftsize = 1024
ioverlap = ifftsize / 4
iwinsize = ifftsize
iwinshape = 1; von-Hann window
Sfile = "fox.wav"
ain soundin Sfile
fftin pvsanal ain, ifftsize, ioverlap, iwinsize, iwinshape; fft-analysis of file
fshift pvshift fftin, ishift, ilowest, ikeepform; shift frequencies
aout pvsynth fshift; resynthesize
out aout
endin

</CsInstruments>
<CsScore>
i 1 0 2.757 0 0 0; no shift at all
i 1 3 2.757 100 0 0; shift all frequencies by 100 Hz
i 1 6 2.757 200 0 0; by 200 Hz
i 1 9 2.757 200 0 1; keep formants by method 1
i 1 12 2.757 200 0 2; by method 2
i 1 15 2.757 200 1000 0; shift by 200 Hz but just above 1000 Hz
i 1 18 2.757 1000 500 0; shift by 1000 Hz above 500 Hz
i 1 21 2.757 1000 300 0; above 300 Hz
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn*

## Crédits

Auteur : Victor Lazzarini  
Novembre 2004

Nouveau greffon dans la version 5

Novembre 2004.

# pvsifd

pvsifd — Distribution de Fréquence Instantanée, analyse de magnitude et de phase.

## Description

L'opcode *pvsifd* prend en entrée un signal de taux-a et effectue une analyse de magnitude et de phase en Fréquence Instantanée, en utilisant la TFCT et la Distribution de Fréquence Instantanée, suivant la méthode décrite dans Lazzarini et al, "Time-stretching using the Instantaneous Frequency Distribution and Partial Tracking", Proc.of ICMC05, Barcelona. Il génère deux flots de signal PV, l'un contenant les amplitudes et les fréquences (comme la sortie de *pvsanal*) et l'autre contenant les amplitudes et les phases non réduites.

## Syntaxe

```
ffr, fphs pvsifd ain, ifftsize, ihopsize, iwintype[, iscal]
```

## Exécution

*ffr* -- flot pv en sortie au format AMP\_FREQ

*fphs* -- flot pv en sortie au format AMP\_PHASE

*ifftsize* -- taille de l'analyse TFR, doit être une puissance de deux et un multiple entier de la taille de saut

*ihopsize* -- taille de saut en échantillons

*iwintype* -- type de la fenêtre (0 : Hamming, 1 : Hanning)

*iscal* -- facteur d'amplitude (1 par défaut).



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

Voici un exemple de l'opcode pvsifd. Il utilise le fichier *pvsifd.csd* [examples/pvsifd.csd].

### Exemple 785. Exemple de l'opcode pvsifd.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvsifd.wav -W ;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

  sr = 44100
  ksmpps = 32
  nchnls = 2
  odbfs = 1

  instr 1

  ain disk2 "beats.wav", 1, 0, 1
  fs1,fsi2 pvsifd ain, 2048, 512, 1 ; pvsifd analysis
  fst partials fs1, fsi2, .1, 1,3, 500 ; partial tracking
  aout resyn fst, 1, 1.5, 500, 1 ; resynthesis (up a 5th)
  outs aout, aout

endin
</CsInstruments>
<CsScore>
; sine
f1 0 4096 10 1

i 1 0 2
e
</CsScore>
</CsSoundSynthesizer>

```

L'exemple ci-dessus montre une analyse *pvsifd* enchaînée à un suivi de partiel puis à une resynthèse additive avec interpolation cubique de la phase et transposition de hauteur.

## Crédits

Auteur : Victor Lazzarini  
Juin 2005

Nouveau greffon dans la version 5

Novembre 2004.

# pvsinfo

pvsinfo — Retourne de l'information sur une source au format PVOC-EX.

## Description

Retourne de l'information sur le format de *fsrc*, que celui-ci soit créé par un opcode comme *pvsanal* ou qu'il soit obtenu à partir d'un fichier PVOC-EX par *pvsfread*. Cette information est disponible à l'initialisation et peut être utilisée pour fixer les paramètres d'autres opcodes pvs, en particulier pour créer des tables de fonction (par exemple pour *pvsftw*), ou pour fixer le nombre d'oscillateurs pour *pvsadsyn*.

## Syntaxe

```
ioverlap, inumbins, iwinsize, iformat pvsinfo fsrc
```

## Initialisation

*ioverlap* -- La taille de recouvrement du flot.

*inumbins* -- Le nombre de bins d'analyse (amplitude + fréquence) dans *fsrc*. La taille de TFR sous-jacente est calculée comme  $(inumbins - 1) * 2$ .

*iwinsize* -- La taille de la fenêtre d'analyse. Peut être supérieure à la taille de TFR.

*iformat* -- Le format de trame d'analyse. Si *fsrc* est créé par un opcode, *iformat* vaut toujours 0, ce qui signifie (amplitude + fréquence). Si *fsrc* est défini à partir d'un fichier PVOC-EX, *iformat* peut également valoir 1 ou 2 (amplitude + phase, complexe).

## Exemples

Voici un exemple de l'opcode pvsinfo. Il utilise le fichier *pvsinfo.csd* [examples/pvsinfo.csd].

### Exemple 786. Exemple de l'opcode pvsinfo.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvsinfo.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
; create a PVOC-EX (*.pvx) file with PVANAL first
```

```
idur filelen "fox.pvx" ;find duration of analysis file
kpos line 0,p3,idur ;to ensure we process whole file
fsrc pvsfread kpos, "fox.pvx" ;create fsig from (mono) file

iovl,inb,iws,ifmt pvsinfo fsrc ;get info
print iovl,inb,iws,ifmt ;print info

aout pvsynth fsrc
outs aout, aout

endin
</CsInstruments>
<CsScore>

i 1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

L'exemple produira la sortie suivante :

```
instr 1: iovl = 256.000 inb = 513.000 iws = 2048.000 ifmt = 0.000
```

## Crédits

Auteur : Richard Dobson  
Août 2001

Nouveau dans la version 4.13



# pvsinit

pvsinit — Initialise une variable spectrale (f) à zéro.

## Description

Réalise l'équivalent d'une opération d'initialisation sur une variable-f.

## Syntaxe

```
fsig pvsinit isize[, iolap, iwinsize, iwintype, iformat]
```

## Exécution

*fsig* -- flot pv de sortie mis à zéro.

*isize* -- taille de la trame de TFD.

*iolap* -- taille du recouvrement de l'analyse. Vaut par défaut *isize*/4.

*iwinsize* -- taille de la fenêtre d'analyse. Vaut par défaut *isize*.

*iwintype* -- type de la fenêtre d'analyse. Vaut par défaut 1, Hanning.

*iformat* -- format pvsdata. Vaut par défaut 0 : PVS\_AMP\_FREQ.

## Exemples

### Exemple 787. Exemples

```
fsig pvsinit 1024
```

## Crédits

Auteur : Victor Lazzarini  
Novembre 2004

Nouveau greffon dans la version 5

Novembre 2004.

# pvsin

pvsin — Récupère un fsig à partir de l'entrée d'un bus logiciel ; un équivalent pvs à chani.

## Description

Cet opcode récupère un fsig à partir du bus logiciel d'entrée pvs, que l'on peut utiliser pour recevoir des données venant d'une source externe via l'API de Csound 5. Un canal est créé s'il n'existe pas déjà. Dans ce cas, le canal du fsig est initialisé avec les paramètres donnés. Il est important de noter que les bus pvs d'entrée et de sortie sont indépendants et qu'ils ne partagent pas leurs données.

## Syntaxe

```
fsig pvsin kchan[, isize, iolap, iwinsize, iwintype, iformat]
```

## Initialisation

*isize* -- taille initiale de TFD, par défaut 1024.

*iolap* -- taille du recouvrement, par défaut *isize*/4.

*iwinsize* -- taille de la fenêtre d'analyse, par défaut *isize*.

*iwintype* -- type de la fenêtre, Hanning (1) par défaut (voir *pvsanal*)

*iformat* -- format des données, 0 par défaut (PVS\_AMP\_FREQ). Les autres valeurs possibles sont 1 (PVS\_AMP\_PHASE), 2 (PVS\_COMPLEX) ou 3 (PVS\_TRACKS).

## Exécution

*fsig* -- fsig récupéré.

*kchan* -- Numéro du canal. S'il n'existe pas, un canal est créé.

## Exemples

### Exemple 788. Exemples

```
fsig pvsin 0 ; get data from pvs in bus channel 0
```

## Crédits

Auteur : Victor Lazzarini  
Août 2006

# pvslock

pvslock — Verrouille en fréquence un fsig d'entrée.

## Description

Cet opcode recherche les crêtes spectrales puis il verrouille les fréquences autour de ces crêtes. C'est l'analogie d'un verrouillage de phase dans le traitement PV statique. On peut l'utiliser pour améliorer la qualité de l'étirement des durées et de la transposition des hauteurs dans le traitement PV.

## Syntaxe

fsig **pvslock** fsigin, klock

## Exécution

*fsig* -- flot pv de sortie.

*fsigin* -- flot pv d'entrée.

*klock* -- verrouillage de fréquence, 1 -> verrouillé, 0 -> non verrouillé (passant).



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

Voici un exemple de l'opcode pvslock. Il utilise le fichier *pvslock.csd* [examples/pvslock.csd].

### Exemple 789. Exemple de l'opcode pvslock.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvslock.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gifil ftgen 0, 0, 0, 1, "fox.wav", 0, 0, 1
```

```
instr 1

klock = p4
fsig    pvstanal 1, 1, 1, gifil ; no further transformations
fsigout pvslock  fsig, klock ; lock frequency
aout    pvsynth  fsigout
        outs     aout, aout

endin
</CsInstruments>
<CsScore>

i 1 0 2.6 1 ; locked
i 1 3 2.6 0 ; not locked

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
Novembre 2004

Nouveau greffon dans la version 5

Novembre 2004.

# pvsmaska

pvsmaska — Modifie les amplitudes en utilisant une table de fonction, avec mise à l'échelle dynamique.

## Description

Modifie les amplitudes de *fsrc* en utilisant une table de fonction, avec mise à l'échelle dynamique.

## Syntaxe

```
fsig pvsmaska fsrc, ifn, kdepth
```

## Initialisation

*ifn* -- La table-f à utiliser. Si le *fsrc* donné a N bins d'analyse, la table *ifn* doit être de taille supérieure ou égale à N. Il n'est pas nécessaire de normaliser la table, mais ses valeurs doivent être comprises entre 0 et 1. Elle peut provenir de la partition de la manière habituelle ou bien de l'orchestre en utilisant *pvsinfo* pour trouver la taille de *fsrc*, (retournée par *pvsinfo* dans *inbins*), qui peut ensuite être passée à *figen* pour créer la table-f.

## Exécution

*kdepth* -- Contrôle le degré de modification appliqué à *fsrc*, en utilisant une simple échelle linéaire. 0 laisse les amplitudes inchangées, 1 applique le profil complet de *ifn*.

Noter que les tailles de TFR en puissances de deux sont particulièrement adaptées au traitement basé sur des tables, car le nombre de bins d'analyse (*inbins*) est alors une puissance de deux plus un, pour lequel une table-f correspondant exactement peut être créée. Dans ce cas il est important que la table-f soit créée avec un taille de *inbins* au lieu d'une puissance de deux, car sinon la première valeur de la table sera copiée dans le point de garde, ce qui ne convient pas pour cet opcode.



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

Voici un exemple de l'opcode pvsmaska. Il utilise le fichier *pvsmaska.csd* [examples/pvsmaska.csd].

### Exemple 790. Exemple de l'opcode pvsmaska.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>
```

```

sr = 44100
ksmps = 16
nchnls = 1
Odbfs = 1

;; example written by joachim heintz 2009

; function table for defining amplitude peaks (from the example of Richard Dobson)
giTab ftgen 0, 0, 513, 8, 0, 2, 1, 3, 0, 4, 1, 6, 0, 10, 1, 12, 0, 16, 1, 32, 0, 1, 0, 436, 0

instr 1
imod = p4; degree of midification (0-1)
ifftsize = 1024
ioverlap = ifftsize / 4
iwinsize = ifftsize
iwinshape = 1; von-Hann window
Sfile = "fox.wav"
ain soundin Sfile
fftin pvsanal ain, ifftsize, ioverlap, iwinsize, iwinshape; fft-analysis of file
fmask pvsmaska fftin, giTab, imod
aout pvsynth fmask; resynthesize
out aout
endin

</CsInstruments>
<CsScore>
i 1 0 2.757 0
i 1 3 2.757 1
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn*

## Crédits

Auteur : Richard Dobson  
Août 2001

Nouveau dans la version 4.13

# pvsmix

pvsmix — Mélange "sans accroc" deux signaux pv.

## Description

Mélange "sans accroc" deux signaux pv. Cet opcode combine les composantes les plus proéminentes de deux flots pvoc en un seul flot mélangé.

## Syntaxe

```
fsig pvsmix fsigin1, fsigin2
```

## Exécution

*fsig* -- flot pv de sortie

*fsigin1* -- flot pv d'entrée

*fsigin2* -- flot pv d'entrée, qui doit avoir le même format que *fsigin1*.



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

Voici un exemple de l'opcode pvsmix. Il utilise le fichier *pvsmix.csd* [examples/pvsmix.csd].

### Exemple 791. Exemple de l'opcode pvsmix.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvsmix.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisin ftgen 1, 0, 2048, 10, 1

instr 1
```

```
asig1 disk2 "fox.wav", 1 ;signal in 1
asig2 oscil .3, 100, gisin ;signal in 2
fsig1 pvsanal asig1,1024,256,1024,0 ;pvoc analysis
fsig2 pvsanal asig2,1024,256,1024,0 ;of both signals
fsall pvmix fsig1, fsig2
asig pvsynth fsall
      outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
Novembre 2004

Nouveau greffon dans la version 5

Novembre 2004.



# pvsmorph

pvsmorph — Effectue un morphing (ou interpolation) entre deux fsigs sources.

## Description

Effectue un morphing (ou interpolation) entre deux fsigs sources.

## Syntaxe

```
fsig pvsmorph fsig1, fsig2, kampint, kfrqint
```

## Exécution

L'opération de cet opcode est identique à celle de *pvcross* sauf que l'on utilise des *fsigs* plutôt que des fichiers d'analyse, et qu'il n'y a pas de préservation de l'enveloppe spectrale. Les amplitudes et les fréquences de *fsig1* sont interpolées avec celles de *fsig2* en fonction des valeurs de *kampint* et de *kfrqint* respectivement. Celles-ci sont comprises entre 0 et 1, où 0 signifie *fsig1* et 1 *fsig2*. Toute valeur entre les deux interpole les amplitudes et/ou les fréquences des deux fsigs.

Avec cet opcode, on peut effectuer un morphing sur des entrées audio en temps réel, en utilisant *pvsanal* pour générer *fsig1* et *fsig2*. Ceux-ci doivent avoir le même format.



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

Voici un exemple de l'opcode pvsmorph. Il utilise le fichier *pvsmorph.csd* [examples/pvsmorph.csd].

### Exemple 792. Exemple de l'opcode pvsmorph.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o pvsmorph.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1
```

```
;; example written by joachim heintz 2009

giSine ftgen 0, 0, 4096, 10, 1

instr 1
iampint1 = p4
iampint2 = p5
ifrqint1 = p6
ifrqint2 = p7
kampint linseg iampint1, p3, iampint2
kfrqint linseg ifrqint1, p3, ifrqint2
ifftsize = 1024
ioverlap = ifftsize / 4
iwinsize = ifftsize
iwinshape = 1; von-Hann window
Sfile1 = "fox.wav"
ain1 soundin Sfile1
ain2 buzz .2, 50, 100, giSine
fftin1 pvsanal ain1, ifftsize, ioverlap, iwinsize, iwinshape
fftin2 pvsanal ain2, ifftsize, ioverlap, iwinsize, iwinshape
fmorph pvsmorph fftin1, fftin2, kampint, kfrqint
aout pvsynth fmorph
out aout * .5
endin

</CsInstruments>
<CsScore>
i 1 0 3 0 0 1 1
i 1 3 3 1 0 1 0
i 1 6 3 0 1 0 1
e
</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode pvsmorph. Il utilise le fichier *pvsmorph2.csd* [examples/pvs-morph2.csd].

### Exemple 793. Exemple de l'opcode pvsmorph.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009
;; this example uses the files "flute-C-octave0.wav" and
;; "saxophone-alto-C-octave0.wav" from www.archive.org/details/OpenPathMusic44V2

giSine ftgen 0, 0, 4096, 10, 1

instr 1
iampint1 = p4; value for interpolating the amplitudes at the beginning ...
iampint2 = p5; ... and at the end
ifrqint1 = p6; value for interpolating the frequencies at the beginning ...
ifrqint2 = p7; ... and at the end
kampint linseg iampint1, p3, iampint2
```

```

kfrqint linseg ifrqint1, p3, ifrqint2
ifftsize = 1024
ioverlap = ifftsize / 4
iwinsize = ifftsize
iwinshape = 1; von-Hann window
Sfile1 = "flute-C-octave0.wav"
Sfile2 = "saxophone-alto-C-octave0.wav"
ain1 soundin Sfile1
ain2 soundin Sfile2
fftin1 pvsanal ain1, ifftsize, ioverlap, iwinsize, iwinshape
fftin2 pvsanal ain2, ifftsize, ioverlap, iwinsize, iwinshape
fmorph pvsmorph fftin1, fftin2, kampint, kfrqint
aout pvsynth fmorph
    out aout * .5
endin

instr 2; moving randomly in certain borders between two spectra
iampintmin = p4; minimum value for amplitudes
iampintmax = p5; maximum value for amplitudes
ifrqintmin = p6; minimum value for frequencies
ifrqintmax = p7; maximum value for frequencies
imovefreq = p8; frequency for generating new random values
kampint randomi iampintmin, iampintmax, imovefreq
kfrqint randomi ifrqintmin, ifrqintmax, imovefreq
ifftsize = 1024
ioverlap = ifftsize / 4
iwinsize = ifftsize
iwinshape = 1; von-Hann window
Sfile1 = "flute-C-octave0.wav"
Sfile2 = "saxophone-alto-C-octave0.wav"
ain1 soundin Sfile1
ain2 soundin Sfile2
fftin1 pvsanal ain1, ifftsize, ioverlap, iwinsize, iwinshape
fftin2 pvsanal ain2, ifftsize, ioverlap, iwinsize, iwinshape
fmorph pvsmorph fftin1, fftin2, kampint, kfrqint
aout pvsynth fmorph
    out aout * .5
endin

</CsInstruments>
<CsScore>
i 1 0 3 0 0 1 1; amplitudes from flute, frequencies from saxophone
i 1 3 3 1 1 0 0; amplitudes from saxophone, frequencies from flute
i 1 6 3 0 1 0 1; amplitudes and frequencies moving from flute to saxophone
i 1 9 3 1 0 1 0; amplitudes and frequencies moving from saxophone to flute
i 2 12 3 .2 .8 .2 .8 5; amps and freqs moving randomly between the two spectra
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn*

## Crédits

Auteur : Victor Lazzarini  
 Avril 2007  
 Nouveau dans Csound 5.06

# pvsmooth

pvsmooth — Lisse les fonctions temporelles d'amplitude et de fréquence d'un flot pv en utilisant des filtres passe-bas RII du premier ordre en parallèle avec une fréquence de coupure variable.

## Description

Lisse les fonctions temporelles d'amplitude et de fréquence d'un flot pv en utilisant un filtre passe-bas RII du premier ordre avec une fréquence de coupure variable. Cet opcode utilise le même filtre que l'opcode *tone*, mais il agit séparément sur les fonctions temporelles d'amplitude et de fréquence qui constituent le flot pv. La fréquence de coupure est un paramètre au taux de contrôle, mais à la différence de *tone* et de *tonek*, elle n'est pas spécifiée en Hz, mais en fractions du 1/2 taux de trame (actuellement le taux d'échantillonnage du flot pv), ce qui est plus facile à comprendre. Cela signifie que la fréquence de coupure la plus haute vaut 1 et que la plus basse vaut 0 ; plus la fréquence de coupure est basse et plus les fonctions sont lisses ce qui donne un effet plus prononcé.

Ce sont des filtres appliqués à des signaux de contrôle si bien que l'effet est fondamentalement un floutage de l'évolution spectrale. Les effets produits ressemblent plus ou moins à ceux de *pvsblur*, mais avec deux différences importantes : 1. le lissage des amplitudes et celui des fréquences utilisent des ensembles séparés de filtres ; 2. le coût de calcul n'est pas plus élevé si l'on désire plus de "floutage" (lissage).

## Syntaxe

```
fsig pvsmooth fsigin, kacf, kfcf
```

## Exécution

*fsig* -- flot pv de sortie

*fsigin* -- flot pv d'entrée

*kacf* -- valeur de la fréquence de coupure pour le filtrage de la fonction d'amplitude, entre 0 et 1 en fractions du 1/2 taux de trame.

*kfcf* -- valeur de la fréquence de coupure pour le filtrage de la fonction de fréquence, entre 0 et 1 en fractions du 1/2 taux de trame.



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

Voici un exemple de l'opcode pvsmooth. Il utilise le fichier *pvsmooth.csd* [exemples/pvsmooth.csd].

### Exemple 794. Exemple de l'opcode pvsmooth.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime output leave only the line below:
; -o pvsmooth.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kacf = p4
kfcf = p5
asig soundin "fox.wav"
fsig pvscanal asig, 1024, 256, 1024, 1 ; analyse it
ftps pvsmooth fsig, kacf, kfcf
atps pvsynth ftps ; synthesise it
outs atps*3, atps*3

endin
</CsInstruments>
<CsScore>
;      amp  freq
i 1 0 3 0.01 0.01 ;smooth amplitude and frequency with cutoff frequency of filter at 1% of 1/2 frame-rate
i 1 + 3 1 0.01 ;no smoothing on amplitude, but frequency with cf at 1% of 1/2 frame-rate (ca 0.86 Hz)
i 1 + 10 .001 1 ;smooth amplitude with cf at 0.1% of 1/2 frame-rate (ca 0.086 Hz)
;and no smoothing of frequency
e
</CsScore>
</CsSoundSynthesizer>
```

La formule du calcul de la fréquence de coupure du filtre étant : (taux de trame) / (taille du saut) = (nouveaux départs de trame par seconde) (en Hz), on en déduit le pourcentage du demi taux de trame. Par exemple, pour la première note de l'exemple, le taux de trame est  $44100 / 256 = 172,265625$  Hz (= 172 nouveaux départs de trame par seconde). La moitié du taux de trame vaut 86 Hz, et un pour cent de ce taux vaut 0.86 Hz.

## Crédits

Auteur : Victor Lazzarini  
Mai 2006

Nouveau greffon dans la version 5

Mai 2006.

# pvsout

pvsout — Écrit un fsig sur le bus de sortie pvs.

## Description

Cet opcode écrit un fsig sur un canal du bus de sortie pvs. Noter que le bus de sortie pvs et le bus d'entrée pvs sont séparés et indépendants. Un nouveau canal est créé si le canal spécifié n'existe pas.

## Syntaxe

```
pvsout fsig, kchan
```

## Exécution

*fsig* -- fsig en entrée.

*kchan* -- numéro de canal du bus de sortie pvs.

## Exemples

### Exemple 795. Exemples

```
asig      in                               ; input
fsig      pvsanal asig, 1024, 256, 1024, 1 ; analysis
          pvsout fsig, 0                   ; write signal to pvs out bus channel 0
```

## Crédits

Auteur : Victor Lazzarini  
Août 2006

# pvsosc

pvsosc — Simulateur d'oscillateur basé sur PVS.

## Description

Génère des spectres de signal périodique au format AMP-FREQ, avec le choix parmi quatre type de forme d'onde :

1. pseudo dent de scie (harmoniques en  $1/n$ , où  $n$  est le numéro de l'harmonique)
2. pseudo carrée (comme la précédente mais seulement avec les harmoniques de rang impair)
3. impulsion (tous les harmoniques avec le même poids)
4. cosinus

Les formes d'onde complexes (c'est-à-dire tous les types sauf le cosinus) contiennent tous les harmoniques jusqu'à la fréquence de Nyquist. Cela fait de *pvsosc* une option pour la génération de formes d'onde périodiques à bande limitée. De plus, on peut changer de type en utilisant une variable de taux- $k$ .

## Syntaxe

```
fsig pvsosc kamp, kfreq, ktype, isize [, ioverlap] [, iwinsize] [, iwintype] [, iformat]
```

## Initialisation

*fsig* -- flot pv de sortie mis à zéro.

*isize* -- taille de la trame d'analyse.

*ioverlap* -- (Facultatif) taille du saut, vaut *isize*/4 par défaut.

*iwinsize* -- (Facultatif) taille de fenêtre, vaut *isize* par défaut.

*iwintype* -- (Facultatif) type de fenêtre, Hanning par défaut. On a actuellement le choix entre :

- 0 = fenêtre de Hamming
- 1 = fenêtre de von Hann

*iformat* -- (Facultatif) format des données, 0 par défaut ce qui produit des données AMP:FREQ. C'est actuellement la seule option.

## Exécution

*kamp* -- amplitude du signal. Noter que l'amplitude effective du signal peut varier légèrement autour de cette valeur en fonction du type de forme d'onde et de la fréquence. Généralement, l'amplitude a tendance à dépasser *kamp* pour les hautes fréquences (> 1000 Hz) et à être inférieure pour les basses fréquences. De plus, à cause du traitement par recouvrement-addition, lorsque l'on fait la resynthèse avec *pvsynth*, les glissements de fréquence causeront une fluctuation de l'amplitude autour de *kamp*.

*kfreq* -- fréquence fondamentale en Hz.

*ktype* -- type d'onde : 1. pseudo dent de scie, 2. pseudo carrée, 3. impulsion, et tout autre valeur pour cosine.

## Exemples

Voici un exemple de l'opcode `pvsosc`. Il utilise le fichier `pvsosc.csd` [exemples/pvsosc.csd].

### Exemple 796. Exemple de l'opcode `pvsosc`

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsosc.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
; a band-limited sawtooth-wave oscillator
fsig pvsosc 10000, 440, 1, 1024 ; generate wave spectral signal
asig pvsynth fsig                ; resynthesise it
out asig
endin

instr 2
; a band-limited square-wave oscillator
fsig pvsosc 10000, 440, 2, 1024 ; generate wave spectral signal
asig pvsynth fsig                ; resynthesise it
out asig
endin

instr 3
; a pulse oscillator
fsig pvsosc 10000, 440, 3, 1024 ; generate wave spectral signal
asig pvsynth fsig                ; resynthesise it
out asig
endin

instr 4
; a cosine-wave oscillator
fsig pvsosc 10000, 440, 4, 1024 ; generate wave spectral signal
asig pvsynth fsig                ; resynthesise it
out asig
endin

</CsInstruments>
<CsScore>

i 1 0 1
i 2 2 1
i 3 4 1
i 4 6 1

e
```



```
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn*

## Crédits

Auteur : Victor Lazzarini  
Août 2006

# pvspitch

pvspitch — Suit la hauteur et l'amplitude d'un signal PVS.

## Description

Suit la hauteur et l'amplitude d'un signal PVS et les restitue dans des variables de taux-k.

## Syntaxe

`kfr, kamp pvspitch fsig, kthresh`

## Exécution

*kamp* -- amplitude de la fréquence fondamentale

*kfr* -- fréquence fondamentale

*fsig* -- flot pv en entrée

*kthresh* -- seuil d'analyse (compris entre 0 et 1). Les valeurs élevées éliminent les composantes de faible amplitude de l'analyse.

L'algorithme de détection de hauteur implémenté dans *pvspitch* est basé sur l'hypothèse de J. F. Schouten du processus neuronal du cerveau utilisé pour déterminer la hauteur d'un son d'après l'analyse de fréquence de la membrane basilaire. Sauf pour certaines considérations plus loin, *pvspitch* cherche essentiellement le plus grand facteur commun des crêtes spectrales du son analysé pour trouver la hauteur qui peut lui être attribuée.

En général, les sons analysés présentant une caractéristique de hauteur auront des crêtes dans leur spectre suivant la position de leurs harmoniques. Il y a cependant quelques exceptions. Certains sons dont la représentation spectrale est continue peuvent provoquer une sensation de hauteur. De tels sons sont expliqués par le centroïde ou centre de gravité du spectre et dépassent le cadre de la méthode de détection de hauteur implémentée par *pvspitch*. (L'utilisation d'opcodes tels que *pvscent* peut être plus appropriée dans ce cas).

*pvspitch* est capable (en utilisant un *fsig* d'analyse généré par *pvsanal*) de localiser les crêtes spectrales d'un signal. Le paramètre de seuil (*kthresh*) est de la plus grande importance, car son ajustement peut introduire des harmoniques faibles mais significatifs pour le calcul du fondamental. Cependant, si l'on donne une valeur trop faible à *kthresh*, cela peut amener des partiels sans relation dans l'algorithme d'analyse ce qui compromettra la précision de la méthode. Ces étapes initiales simulent la réponse de la membrane basilaire en identifiant les caractéristiques physiques du son analysé. Le choix de *kthresh* dépend du niveau actuel du signal analysé, car ses valeurs (comprises entre 0 et 1) couvrent tout l'intervalle dynamique d'un bin d'analyse (de -inf à 0dBFS).

Il est important de se souvenir que l'entrée de l'opcode *pvspitch* est supposée se caractériser par des partiels importants dans le spectre. Si ce n'est pas le cas, le résultat retourné par l'opcode peut n'avoir aucune relation avec la hauteur du signal entrant. Si une trame contenant plusieurs partiels sans rapport a été analysée, le plus grand facteur commun de ces valeurs de fréquence autorisant des "harmoniques" adjacents sera choisi. Ainsi, des trames bruiteuses peuvent être caractérisées par une sortie basse fréquence de *pvspitch*. Ce fait permet un type primitif de détection de transitoire instrumental, car la portion d'attaque de certains sons instrumentaux contient des composants inharmoniques. Si l'on connaît la fréquence la plus basse de

la mélodie analysée, alors toutes les fréquences détectées sous ce seuil représentent une lecture erronée due à la présence de partiels sans rapport.

Afin de faciliter un test efficace de l'algorithme de *pvspitch*, une valeur d'amplitude proportionnelle à celle qui est observée dans la trame de signal est également retournée (*kamp*). On peut ainsi utiliser les résultats de *pvspitch* pour piloter un oscillateur dont on peut comparer à l'écoute la hauteur avec celle du signal original (dans l'exemple ci-dessous, un oscillateur génère un signal qui se trouve une quinte au-dessus de la hauteur détectée).

## Exemples

Voici un exemple de l'opcode *pvspitch*. Il utilise le fichier *pvspitch.csd* [examples/pvspitch.csd]. Cette exemple utilise l'entrée audio en temps réel mais on peut tout aussi bien utiliser un fichier son en entrée.

### Exemple 797. Exemple de l'opcode *pvspitch*

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvspitch.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 1

giwave ftgen 0, 0, 4096, 10, 1, 0.5, 0.333, 0.25, 0.2, 0.1666

instr 1

ifftsize = 1024
iwtype = 1      /* cleaner with hanning window */

a1 inch 1 ;Realtime audio input
;a1 soundin "input.wav" ;Use this line for file input

fsig pvsanal a1, ifftsize, ifftsize/4, ifftsize, iwtype
kfr, kamp pvspitch fsig, 0.01

adm oscil      kamp, kfr * 1.5, giwave ;Generate note a fifth above detected pitch

      out      adm

endin

</CsInstruments>
<CsScore>

i 1 0 30

e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn, pvscent*

## Crédits

Auteur : Alan OCinneide

Août 2005, ajouté par Victor Lazzarini, Août 2006

Une partie du texte a été adaptée de l'article de Alan OCinneide "Introducing PVSPITCH: A pitch tracking opcode for Csound" dans la parution du Csound Journal de l'hiver 2006. L'article est disponible ici : [www.csoundjournal.com/2006winter/pvspitch.html](http://www.csoundjournal.com/2006winter/pvspitch.html) [http://www.csoundjournal.com/2006winter/pvspitch.html]

# pvstanal

pvstanal — Traitement par analyse de vocodeur de phase avec détection/traitement d'attaque.

## Description

*pvstanal* implémente une analyse par vocodeur de phase en lisant des tables de fonction contenant des sources de son échantillonné avec *GEN01*. *pvstanal* accepte ainsi les tables à allocation différée.

Cet opcode permet une mise à l'échelle du temps indépendante de celle de la fréquence. Le temps progresse en interne mais il est contrôlé par un paramètre de mise à l'échelle du tempo ; lorsqu'une attaque est détectée, l'échelonnement du temps est momentanément stoppé pour éviter le brouillage des attaques. La qualité de l'effet est généralement améliorée avec le verrouillage de phase activé.

*pvstanal* met aussi à l'échelle la hauteur, indépendamment de la fréquence, en utilisant un facteur de transposition (taux-k).

## Syntaxe

fsig **pvstanal** ktimescal, kamp, kpitch, ktab, [kdetect, kwrap, ioffset, ifftsize, ihop, idbthresh]

## Initialisation

*ifftsize* -- taille de la TFR (puissance de deux), vaut par défaut 2048.

*ihop* -- taille du saut, vaut par défaut 512

*ioffset* -- décalage du début de la lecture dans la table, en secondes

*idbthresh* -- seuil pour la détection des attaques, basé sur le rapport de spectre de puissance en dB entre deux fenêtres successives. Un rapport détecté au-dessus du seuil stoppe momentanément l'échelonnement du temps, pour éviter le brouillage (vaut 1 par défaut). Par défaut, tout ce qui dépasse une différence de puissance de 1 dB entre trames sera détecté comme un attaque.

## Exécution

*ktimescal* -- rapport de mise à l'échelle du temps, < 1 étirement, > 1 contraction.

*kamp* -- mise à l'échelle de l'amplitude

*kpitch* -- mise à l'échelle de la hauteur de grain (1=hauteur normale, < 1 plus grave, > 1 plus aigu ; négatif, à l'envers)

*kdetect* -- 0 ou 1, pour désactiver/activer les détections/traitements d'attaque. Le détecteur d'attaque cherche les différences de puissance entre les fenêtres d'analyse. S'il trouve plus que ce qui a été spécifié dans le paramètre *dbthresh*, une attaque est déclarée. La mise à l'échelle du temps est momentanément suspendue afin que les attaques ne soient pas modifiées.

*ktab* -- table de fonction du signal source. Les tables à allocation différée (voir *GEN01*) sont acceptées, mais l'opcode attend une source mono. On peut changer de table au taux-k.

*kwrap* -- 0 ou 1, pour désactiver/activer la lecture cyclique de table (vaut 1 par défaut).

## Exemples

Voici un exemple de l'opcode `pvstanal`. Il utilise le fichier `pvstanal.csd` [examples/pvstanal.csd].

### Exemple 798. Exemple de l'opcode `pvstanal`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvstanal.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gifil      ftgen      0, 0, 0, 1, "fox.wav", 0, 0, 1

instr 1

fsig      pvstanal    p4, 1, p5, gifil, p6, p7
aout      pvsynth     fsig
          outs        aout, aout

endin

instr 2

kspeed     randi      2, 2, 2 ;speed randomly between -2 and 2
kpitch     randi      2, 2, 2 ;pitch between 2 octaves lower or higher
fsig       pvstanal    kspeed, 1, octave(kpitch), gifil
aout       pvsynth     fsig
          outs        aout, aout

endin

</CsInstruments>
<CsScore>
;          speed pch det wrap
i 1 0 2.757 1      1 0 0
i 1 3 .      2      1 0 0
i 1 6 .      2      1 0 1
i 1 9 .      1      .75
i 2 12 10 ;random scratching
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
Février 2010

Nouveau greffon dans la version 5.13

Février 2005.

# pvstencil

pvstencil — Transforme un flot pvoc selon une table de fonction de masquage.

## Description

Transforme un flot pvoc selon une table de fonction de masquage ; si l'amplitude du flot pvoc tombe sous la valeur de la fonction pour un canal pvoc spécifique, un gain est appliqué à ce canal.

Les amplitudes du flot pvoc sont comparées à une table de masquage ; si elles tombent sous les valeurs de la table, elles sont pondérées par *kgain*. Avant l'opération, les valeurs de la table sont pondérées par *klevel*, qui peut être utilisé comme contrôle de l'importance du masquage.

Les tables doivent avoir une taille d'au moins  $fftsiz/2$  ; pour la plupart des *GENS* il est important d'utiliser un point de garde (taille en puissance-de-deux plus un), cependant ceci n'est pas nécessaire avec *GEN43*.

Un des usages typiques de *pvstencil* est la réduction de bruit. Une empreinte de bruit peut être analysée avec *pvanal* en un fichier PVOC-EX et chargée dans une table avec *GEN43*. On peut ensuite utiliser celle-ci comme table de masquage pour *pvstencil* et l'importance de la réduction est contrôlée par *kgain*. Si l'on ignore la post-normalisation, les amplitudes moyennes de l'empreinte de bruit originale sont conservées. Cela fournit un bon point de départ pour une réduction de bruit réussie (si bien que *klevel* peut généralement être proche de 1).

D'autres effets de transformation sont possibles, tels que le filtrage et le "masquage inverse".

## Syntaxe

```
fsig pvstencil fsigin, kgain, klevel, iftable
```

## Exécution

*fsig* -- flot pv de sortie

*fsigin* -- flot pv d'entrée

*kgain* -- gain du "pochoir"

*klevel* -- niveau de la fonction de masquage (pondère la ftable avant le "pochoir").

*iftable* -- table de la fonction de masquage



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

### Exemple 799. Exemples

```
fsig    pvsanal    asig, 1024, 256, 1024, 1
fclean  pvstencil  fsig, 0, 1, 1
aclean  pvsynth    fclean
```

## Crédits

Auteur : Victor Lazzarini  
Novembre 2004

Nouveau greffon dans la version 5

Novembre 2004.



# pvstrace

pvstrace — Ne retient que les N bins les plus forts.

## Description

Traite un flot PV en ne retenant que les N bins avec la plus grande amplitude, annulant les autres.

## Syntaxe

```
fsig pvstrace fsigin, kn  
fsig, kBins[] pvstrace fsigin, kn[, isort]
```

## Initialisation

*isort* -- s'il est non nul, les bins reportés dans kBins sont triés par amplitude décroissante. Facultatif, vaut 0 par défaut.

## Exécution

*fsig* -- Flot PV en sortie.

*kBins[]* -- un tableau contenant  $\text{tailleTFR}/2 + 1$  valeurs, dont les N premières valeurs reportent les kn numéros de bin retenus par pvstrace. Les autres positions sont mises à zéro. Il peut être trié en option.

*fsigin* -- Flot PV en entrée.

*kn* -- Nombre de bins à retenir.



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

Voici un exemple de l'utilisation de l'opcode *pvstrace*. Il utilise le fichier *pvstrace.csd* [exemples/pvs-trace.csd].

### Exemple 800. Exemple de l'opcode *pvstrace*.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac   ;;realtime audio out  
;-iadc   ;;uncomment -iadc if realtime audio input is needed too  
; For Non-realtime ouput leave only the line below:
```

```

; -o pvstrace.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kn = p4
asig diskin2 "beats.wav", 1
fsig pvsanal asig, 1024, 256, 1024, 1; analyse it
ftps pvstrace fsig, kn ; keep kn bins
atps pvsynth ftps ; synthesise it
outs atps, atps

endin
</CsInstruments>
<CsScore>

i1 0 2 5
i1 + 2 10
i1 + 2 20
i1 + 2 40
i1 + 2 80
i1 + 2 160
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*pvsanal, pvsynth, pvsadsyn*

## Crédits

Auteur : Victor Lazzarini  
2017

Nouveau greffon dans la version 6.09

# pvsvoc

pvsvoc — Combine l'enveloppe spectrale d'un fsig avec l'excitation (les fréquences) d'un autre fsig.

## Description

Cet opcode fournit un support pour la synthèse croisée des amplitudes et des fréquences. Il prend les amplitudes d'un fsig en entrée et les combine avec les fréquences d'un autre fsig. C'est une version spectrale du célèbre vocodeur.

## Syntaxe

fsig **pvsvoc** famp, fexc, kdepth, kgain [,kcoefs]

## Exécution

*fsig* -- flot pv de sortie

*famp* -- flot pv d'entrée duquel sont extraites les amplitudes

*fexc* -- flot pv d'entrée duquel sont prises les fréquences

*kdepth* -- importance de l'effet, affectant la quantité de fréquences prélevées sur le second fsig : 0, la sortie est le signal *famp*, 1 la sortie combine les amplitudes de *famp* avec les fréquences de *fexc*.

*kgain* -- gain de renforcement/atténuation appliqué à la sortie.

*kcoefs* -- nombre de coefficients du cepstre utilisés dans l'estimation de l'enveloppe spectrale (vaut 80 par défaut).



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

Voici un exemple de l'opcode pvsvoc. Il utilise le fichier *pvsvoc.csd* [examples/pvsvoc.csd].

### Exemple 801. Exemple de l'opcode pvsvoc.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvsvoc.wav -W ;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

  sr = 44100
  ksmps = 32
  nchnls = 2
  0dbfs = 1

  gisaw ftgen 1, 0, 2048, 10, 1, 0.5, 0.3, 0.25, 0.2 ;sawtooth-like

  instr 1

  asig in ;get the signal in
  asyn poscil .6, 150, gisaw ;excitation signal of 150 Hz

  famp pvsanal asig, 1024, 256, 1024, 1 ;analyse in signal
  fexc pvsanal asyn, 1024, 256, 1024, 1 ;analyse excitation signal
  ftps pvsvoc famp, fexc, 1, 1 ;cross it
  atps pvsynth ftps ;synthesise it
  outs atps, atps

endin
</CsInstruments>
<CsScore>

i 1 0 10
e
</CsScore>
</CsoundSynthesizer>

```

L'exemple ci-dessus montre une opération typique de synthèse croisée. Le signal d'entrée (disons un signal vocal) est utilisé pour son spectre d'amplitude. Un oscillateur avec une forme d'onde complexe arbitraire produit le signal d'excitation, donnant au son vocal sa hauteur.

## Crédits

Auteur : Victor Lazzarini  
Avril 2005

Nouveau greffon dans la version 5

Avril 2005.

# pvsynth

pvsynth — Resynthèse par recouvrement-addition de TFR.

## Description

Resynthèse de données de vocodeur de phase (signal-f) par recouvrement-addition de TFR.

## Syntaxe

```
ares pvsynth fsrc, [iinit]
```

## Exécution

*ares* -- signal audio en sortie

*fsrc* -- signal d'entrée

*iinit* -- pas encore implémenté.

## Exemples

**Exemple 802. Exemple (utilisant une table-f provenant de la partition, en supposant un fsig avec *fftsize* = 1024)**

```
; score ftable using cubic spline to define shaped peaks
f1 0 513 8 0 2 1 3 0 4 1 6 0 10 1 12 0 16 1 32 0 1 0 436 0

asig    buzz      20000, 199, 50, 1          ; pulsewave source
fsig    pvsanal   asig, 1024, 256, 1024, 0    ; create fsig
kmod    linseg    0, p3/2, 1, p3/2, 0        ; simple control sig

fsigout pvsmaska fsig, 2, kmod                ; apply weird eq to fsig
aout    pvsynth   fsigout                     ; resynthesize,
        dispfft   aout, 0.1, 1024             ; and view the effect
```

Voici un exemple de l'opcode pvsynth. Il utilise le fichier *pvsynth.csd* [examples/pvsynth.csd].

**Exemple 803. Exemple de l'opcode pvsynth.**

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o pvsynth.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
```

```
ksmps = 16
nchnls = 1
0dbfs = 1

;; example written by joachim heintz 2009

instr 1
  ifftsize = 1024
  ioverlap = ifftsize / 4
  iwinsize = ifftsize
  iwinshape = 1 ; von-Hann window
  Sfile = "fox.wav"
  ain soundin Sfile
  fftin pvsanal ain, ifftsize, ioverlap, iwinsize, iwinshape; fft-analysis of the audio-signal
  aout pvsynth fftin; resynthesis
  out aout
endin

</CsInstruments>
<CsScore>
i 1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvsanal, pvsadsyn*

## Crédits

Auteur : Richard Dobson  
Août 2001

Nouveau dans la version 4.13

Février 2004. Grâce à une note de Francisco Vila, l'exemple a été mis à jour.

# pvswarp

pvswarp — Déforme l'enveloppe spectrale d'un signal PVS.

## Description

Déforme l'enveloppe spectrale d'un signal PVS par translation et pondération.

## Syntaxe

```
fsig pvswarp fsigin, kscal, kshift[, klowest, kmeth, kgain, kcoefs]
```

## Exécution

*fsig* -- flot pv de sortie

*fsigin* -- flot pv d'entrée

*kscal* -- rapport de pondération de l'enveloppe spectrale. Les valeurs > 1 étirent l'enveloppe et les valeurs < 1 la compresse.

*kshift* -- translation de l'enveloppe spectrale (en Hz). Les valeurs > 0 décalent l'enveloppe linéairement vers le haut et les valeurs < 0 la décalent vers le bas.

*klowest* -- fréquence décalée la plus basse (n'affecte que *kshift*, vaut 0 par défaut).

*kmethod* -- méthode d'extraction de l'enveloppe spectrale. 1 : méthode du cepstre décalé ; 2 : méthode d'enveloppe véritable (vaut 1 par défaut).

*kgain* -- pondération de l'amplitude (1 par défaut).

*kcoefs* -- nombre de coefficients du cepstre utilisés dans la préservation des formants (vaut 80 par défaut).



### Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes pvs. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

Voici un exemple de l'opcode pvswarp. Il utilise le fichier *pvswarp.csd* [exemples/pvswarp.csd].

### Exemple 804. Exemple de l'opcode pvswarp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;;realtime audio out
```

```

;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pvswarp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kscal = p4
kshift = p5
asig soundin "fox.wav" ; get the signal in
fsig pvsanal asig, 1024, 256, 1024, 1 ; analyse it
ftps pvswarp fsig, kscal, kshift ; warp it
atps pvsynth ftps ; synthesise it
outs atps/2, atps/2

endin
</CsInstruments>
<CsScore>
;change scale
i 1 0 3 1
i 1 + . 1.5
i 1 + . 3
i 1 + . .25
;change shift
i 1 + . 1 0
i 1 + . . 300
i 1 + . . 0
i 1 + . . -300
e
</CsScore>
</CsoundSynthesizer>

```

Utilisé avec des sons vocaux, il décale les formants ce qui donne un changement du timbre des voyelles, semblable à l'effet de l'inhalation d'hélium sur la voix d'un chanteur (l'effet "donald duck").

## Voir aussi

*pvsanal, pvsynth, pvsadsyn*

## Crédits

Auteur : Victor Lazzarini  
 Novembre 2004

Nouveau greffon dans la version 5

Novembre 2004.



# pvs2tab

**pvs2tab** — Copie des données spectrales dans des tableaux de taux-k (ou variables-t). Appelé aussi **pvs2array**.

## Description

Copie une trame **pvs** dans une variable-t. Pour le moment, seuls les formats AMP+FREQ et AMP+PHASE sont autorisés. Les variables-t étant une version antérieure des tableaux de taux-k, l'opcode fonctionne aussi avec celles-ci. L'opcode *pvs2array* est un alias de celui-ci.

## Syntaxe

```
kframe pvs2tab tvar|kvar[], fsig  
kframe pvs2tab kmags[], kfregs[], fsig
```

## Exécution

*kframe* -- numéro de la trame copiée. On peut l'utiliser pour détecter quand une nouvelle trame a été copiée.

*tvar|kvar[]* -- variable-t ou tableau de taux-k contenant la sortie. Elle est alimentée à chaque période-k, mais elle peut ne pas contenir une nouvelle trame, les trames **pvs** étant produites selon leur propre rythme qui est indépendant de *kr*. Généralement, ce vecteur doit être suffisamment grand pour contenir les échantillons de la trame, c'est-à-dire  $N+2$  ( $N$  est la taille de la TFD). S'il n'est pas assez grand, seule une portion de la trame sera copiée ; s'il est trop grand, des points non utilisés se trouveront aux indices plus élevés.

*kmags[], kfregs[]* -- tableaux de taux-k contenant les valeurs de magnitude et de fréquence. Généralement ces tableaux doivent avoir une taille de  $N/2 + 1$ , où  $N$  est la taille d'une trame de TFD. S'ils sont plus petits, seule une partie de la trame de TFD sera copiée ; s'ils sont plus grands, il y aura des points inutilisés aux indices les plus hauts.

*fsig* -- *fsig* d'entrée à copier.

*iwinsize* -- taille de la fenêtre d'analyse valant par défaut *isize*.

## Exemples

### Exemple 805. Exemple

```
karr[]  init    1026  
al      inch    1  
fsigl   pvsanal al, 1024, 256, 1024, 1  
kframe  pvs2tab karr, fsigl
```

## Crédits

Auteur : Victor Lazzarini  
Octobre 2011

Nouveau greffon dans la version 5.14

Octobre 2011.

## pyassign Opcodes

**pyassign** — Affecte la valeur de la variable de Csound donnée à une variable Python, écrasant son contenu précédent.

### Syntaxe

```
pyassign "variable", kvalue  
pyassigni "variable", ivalue  
pylassign "variable", kvalue  
pylassigni "variable", ivalue  
pyassignt ktrigger, "variable", kvalue  
pylassignt ktrigger, "variable", kvalue
```

### Description

Affecte la valeur de la variable de Csound donnée à une variable Python, écrasant son contenu précédent. L'objet Python résultant sera un nombre en virgule flottante.

### Crédits

Copyright (c) 2002 Maurizio Umberto Puxeddu. Tous droits réservés. Certaines parties, copyright (c) 2004 et 2005 Michael Gogins. Ce document a été mis à jour le 25 juillet 2004 et le 1er février 2005 par Michael Gogins.

# pycall Opcodes

**pycall** — Invoque l'objet Python callable spécifié au taux-k ou au taux-i (suffixe i), en lui passant les arguments donnés. L'appel est exécuté dans l'environnement global et le résultat (la valeur retournée) est copié dans les variables de Csound spécifiées en sortie.

## Syntaxe

	<b>pycall</b> "callable", karg1, ...
kresult	<b>pycall11</b> "callable", karg1, ...
kresult1, kresult2	<b>pycall12</b> "callable", karg1, ...
kr1, kr2, kr3	<b>pycall13</b> "callable", karg1, ...
kr1, kr2, kr3, kr4	<b>pycall14</b> "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5	<b>pycall15</b> "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6	<b>pycall16</b> "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7	<b>pycall17</b> "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8	<b>pycall18</b> "callable", karg1, ...
	<b>pycall1t</b> ktrigger, "callable", karg1, ...
kresult	<b>pycall11t</b> ktrigger, "callable", karg1, ...
kresult1, kresult2	<b>pycall12t</b> ktrigger, "callable", karg1, ...
kr1, kr2, kr3	<b>pycall13t</b> ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4	<b>pycall14t</b> ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5	<b>pycall15t</b> ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6	<b>pycall16t</b> ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7	<b>pycall17t</b> ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8	<b>pycall18t</b> ktrigger, "callable", karg1, ...
	<b>pycall1i</b> "callable", karg1, ...
iresult	<b>pycall11i</b> "callable", iarg1, ...
iresult1, iresult2	<b>pycall12i</b> "callable", iarg1, ...
ir1, ir2, ir3	<b>pycall13i</b> "callable", iarg1, ...
ir1, ir2, ir3, ir4	<b>pycall14i</b> "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5	<b>pycall15i</b> "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6	<b>pycall16i</b> "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6, ir7	<b>pycall17i</b> "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6, ir7, ir8	<b>pycall18i</b> "callable", iarg1, ...
<b>pycalln</b> "callable", nresults, kresult1, ..., kresultn, karg1, ...	
<b>pycallni</b> "callable", nresults, iresult1, ..., iresultn, iarg1, ...	
	<b>pylcall</b> "callable", karg1, ...
kresult	<b>pylcall11</b> "callable", karg1, ...

kresult1, kresult2	<b>pylcall12</b>	"callable", karg1, ...
kr1, kr2, kr3	<b>pylcall13</b>	"callable", karg1, ...
kr1, kr2, kr3, kr4	<b>pylcall14</b>	"callable", karg1, ...
kr1, kr2, kr3, kr4, kr5	<b>pylcall15</b>	"callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6	<b>pylcall16</b>	"callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7	<b>pylcall17</b>	"callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8	<b>pylcall18</b>	"callable", karg1, ...
	<b>pylcall1t</b>	ktrigger, "callable", karg1, ...
kresult	<b>pylcall11t</b>	ktrigger, "callable", karg1, ...
kresult1, kresult2	<b>pylcall12t</b>	ktrigger, "callable", karg1, ...
kr1, kr2, kr3	<b>pylcall13t</b>	ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4	<b>pylcall14t</b>	ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5	<b>pylcall15t</b>	ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6	<b>pylcall16t</b>	ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7	<b>pylcall17t</b>	ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8	<b>pylcall18t</b>	ktrigger, "callable", karg1, ...
	<b>pylcall1i</b>	"callable", karg1, ...
iresult	<b>pylcall11i</b>	"callable", iarg1, ...
iresult1, iresult2	<b>pylcall12i</b>	"callable", iarg1, ...
ir1, ir2, ir3	<b>pylcall13i</b>	"callable", iarg1, ...
ir1, ir2, ir3, ir4	<b>pylcall14i</b>	"callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5	<b>pylcall15i</b>	"callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6	<b>pylcall16i</b>	"callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6, ir7	<b>pylcall17i</b>	"callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6, ir7, ir8	<b>pylcall18i</b>	"callable", iarg1, ...
<b>pylcalln</b>	"callable", nresults, kresult1, ..., kresultn, karg1, ...	
<b>pylcallni</b>	"callable", nresults, iresult1, ..., iresultn, iarg1, ...	

## Description

Cette famille d'opcodes appelle l'objet Python callable spécifié au taux-k ou au taux-i (suffixe i), en lui passant les arguments donnés. L'appel est exécuté dans l'environnement global et le résultat (la valeur retournée) est copié dans les variables de Csound spécifiées en sortie.

Ils passent n'importe quel nombre de paramètres qui sont transformés en nombres en virgule flottante dans l'interpréteur Python.

Les opcodes *pycall/pycalli*, *pycall1/pycall1i* ... *pycall8/pycall8i* permettent d'avoir de 0 à 8 résultats en fonction de leur préfixe numérique (0 est omis).

Les opcodes *pycalln/pycallni* peuvent avoir n'importe quel nombre de résultats : le nom de l'objet callable est suivi du nombre d'arguments en sortie, puis viennent la liste des variables de sortie de Csound et la liste des paramètres à transmettre.

La valeur retournée par l'objet callable doit être `None` pour *pycall* ou *pycalli*, un nombre en virgule flottante pour *pycall1i* ou *pycall1i* et un tuple (de taille appropriée) de nombres en virgule flottante pour les opcodes *pycall2/pycall2i* ... *pycall8/pycall8i* et *pycalln/pycallni*.

## Exemples

### Exemple 806. Appel d'une fonction C ou Python

En supposant qu'au préalable nous avons défini ou importé une fonction nommée `get_number_from_pool` comme ceci :

```
from random import random, choice

# un ensemble de 100 nombres
pool = [i ** 1.3 for i in range(100)]

def get_number_from_pool(n, p):
    # substituer un nouveau nombre à un ancien nombre ?
    if random() < p:
        i = choice(range(len(pool)))
        pool[i] = n

    # retourner un nombre pris aléatoirement dans l'ensemble
    return choice(pool)
```

le code d'orchestre suivant

```
k2    pycall1 "get_number_from_pool", k1, p6
```

donnera à *k2* une valeur extraite aléatoirement d'un ensemble de nombres évoluant dans le temps. On peut passer de nouveaux éléments à l'ensemble et contrôler le taux de changement depuis l'orchestre.

### Exemple 807. Appel d'un objet fonctionnel

Une implémentation plus générique de l'exemple précédent utilise un objet fonctionnel simple :

```
from random import random, choice

class GetNumberFromPool:
    def __init__(self, e, begin=0, end=100, step=1):
        self.pool = [i ** e for i in range(begin, end, step)]

    def __call__(self, n, p):
        # substituer un nouveau nombre à un ancien nombre ?
        if random() < p:
            i = choice(range(len(pool)))
            pool[i] = n

        # retourner un nombre pris aléatoirement dans l'ensemble
        return choice(pool)

get_number_from_pool1 = GetNumberFromPool(1.3)
get_number_from_pool2 = GetNumberFromPool(1.5, 50, 250, 2)
```

Alors, le code d'orchestre suivant :

```
k2    pycall1 "get_number_from_pool1", k1, p6
k4    pycall1 "get_number_from_pool2", k3, p7
```

donnera à  $k2$  et à  $k4$  des valeurs prises aléatoirement dans l'ensemble de nombres évoluant dans le temps. On peut passer de nouveaux éléments à l'ensemble (ici  $k1$  et  $k3$  et contrôler le taux de changement (ici  $p6$  et  $p7$ ) depuis l'orchestre.

Comme on peut le voir, il est possible de particulariser l'initialisation de l'ensemble ou de créer plusieurs ensembles.

## Crédits

Copyright (c) 2002 Maurizio Umberto Puxeddu. Tous droits réservés. Certaines parties, copyright (c) 2004 et 2005 Michael Gogins. Ce document a été mis à jour le 25 juillet 2004 et le 1er février 2005 par Michael Gogins.

## pyeval Opcodes

**pyeval** — Evalue une expression Python générique et met le résultat dans une variable de Csound au taux-k ou au taux-i (suffixe i).

### Syntaxe

```
kresult pyeval "expression"
iresult pyevali "expression"
kresult pyleval "expression"
iresult pylevali "expression"
kresult pyevalt ktrigger, "expression"
kresult pylevalt ktrigger, "expression"
```

### Description

Ces opcodes évaluent une expression Python générique et mettent le résultat dans une variable de Csound au taux-k ou au taux-i (suffixe i).

L'évaluation de l'expression doit donner un nombre en virgule flottante ou un objet convertible en nombre flottant.

Ils peuvent être utilisés pour transférer des données d'un objet Python dans une variable de Csound.

### Exemple du groupe d'opcodes pyleval

Le code :

```
k1          pyeval      "v1"
```

copie le contenu de la variable Python *v1* dans la variable *k1* de Csound à chaque cycle de contrôle.

### Crédits

Copyright (c) 2002 Maurizio Umberto Puxeddu. Tous droits réservés. Certaines parties, copyright (c) 2004 et 2005 Michael Gogins. Ce document a été mis à jour le 25 juillet 2004 et le 1er février 2005 par Michael Gogins.



# pyexec Opcodes

pyexec — Exécute un script depuis un fichier au taux-k ou au taux-i (suffixe i).

## Syntaxe

```
pyexec "filename"
pyexeci "filename"
pylexec "filename"
pylexeci "filename"
pyexec ktrigger, "filename"
pylexec ktrigger, "filename"
```

## Description

Exécute un script depuis un fichier au taux-k ou au taux-i (suffixe i).

Ce n'est pas la même chose que d'appeler le script avec la commande `system()`, car le code est exécuté par l'interpréteur embarqué.

Le code contenu dans le fichier spécifié est exécuté dans l'environnement global pour les opcodes *pyexec* et *pyexeci* et dans l'environnement privé pour les opcodes *pylexec* et *pylexeci*.

Ces opcodes n'effectuent aucune transmission de message. Cependant, comme leurs instructions ont accès aux espaces de noms globaux et privés, ils peuvent interagir avec des objets préalablement créés dans ces environnements.

Les versions « locales » des opcodes *pyexec* sont utiles lorsque le code exécuté par différentes instances d'un instrument ne doit pas interagir.

## Exemple du groupe d'opcodes pyexec

### Exemple 808. Orchestre (pyexec.orc)

```
sr=44100
kr=4410
ksmps=10
nchnls=1

;Si vous n'exécutez pas CsoundAC, la ligne suivante est
;nécessaire pour initialiser l'interpréteur Python
;pyinit

pyruni "import random"

pyexeci "pyexec1.py"

instr 1

pyexec          "pyexec2.py"

pylexeci        "pyexec3.py"
```

```

        pylexec          "pyexec4.py"

    endin

```

### Exemple 809. Partition (pyexec.sco)

```

i1 0 0.01
i1 0 0.01

```

### Exemple 810. Le script pyexec1.py

```

import time, os

print
print "Bienvenue dans Csound !"

try:
    s = ', %s?' % os.getenv('USER')
except:
    s = '?'

print 'Quel son voulez-vous écouter aujourd'hui, %s ?' % s
answer = raw_input()

```

### Exemple 811. Le script pyexec2.py

```

print 'votre réponse est "%s"' % answer

```

### Exemple 812. Le script pyexec3.py

```

message = 'un nombre aléatoire privé : %f' % random.random()

```

### Exemple 813. Le script pyexec4.py

```

print message

```

Si j'exécute cet exemple sur ma machine, j'obtiens quelque chose comme ceci :

```

Using ../../csound.xmg
Csound Version 4.19 (Mar 23 2002)
Embedded Python interpreter version 2.2
orchname: pyexec.orc
scorename: pyexec.sco
sorting score ...
... done
orch compiler:
11 lines read
    instr 1
Csound Version 4.19 (Mar 23 2002)
displays suppressed

Bienvenue dans Csound !
Quel son voulez-vous écouter aujourd'hui, maurizio ?

je réponds alors

un son

```

Csound continue l'exécution normale

```
votre réponse est "un son"
un nombre aléatoire privé : 0.884006
new alloc for instr 1:
votre réponse est "un son"
un nombre aléatoire privé : 0.884006
votre réponse est "un son"
un nombre aléatoire privé : 0.889868
votre réponse est "un son"
un nombre aléatoire privé : 0.884006
votre réponse est "un son"
un nombre aléatoire privé : 0.889868
votre réponse est "un son"
un nombre aléatoire privé : 0.884006
votre réponse est "un son"
...
```

Dans le même instrument un message est créé dans l'espace de noms privé et affiché, apparaissant différent pour chaque instance.

## Crédits

Copyright (c) 2002 Maurizio Umberto Puxeddu. Tous droits réservés. Certaines parties, copyright (c) 2004 et 2005 Michael Gogins. Ce document a été mis à jour le 25 juillet 2004 et le 1er février 2005 par Michael Gogins.

# pyinit Opcodes

pyinit — Initialise l'interpréteur Python.

## Syntaxe

`pyinit`

## Description

Dans Csound, il faut d'abord invoquer l'opcode *pyinit* dans l'en-tête de l'orchestre pour initialiser l'interpréteur Python, avant d'utiliser n'importe quel autre des opcodes Python.

Mais si l'on utilise les opcodes Python dans la version CsoundAC de Csound ou depuis un frontal en python utilisant le module csnd6, il n'est pas nécessaire d'invoquer *pyinit*, car l'interpréteur est déjà initialisé. Dans ce cas, CsoundAC (ou le module python csnd6) crée automatiquement une interface Python à l'API de Csound. Dans CsoundAC cela existe sous la forme d'une instance globale de la classe `CsoundAC.CppSound`, nommée `csound`. Dans un frontal en python qui importe le module csnd6, le nom de la variable contenant l'instance de Csound dépend du code du frontal. Ainsi, le code Python écrit dans l'orchestre de Csound a accès à l'objet global `csound`.

L'instance courante de Csound dans laquelle *pyinit* a été appelé est mémorisée dans une variable globale de python appelée `_CSOUND_`. Celle-ci contient l'adresse mémoire de l'instance et l'on peut l'utiliser avec le module csnd6 via l'appel `csoundGetInstance(instance)`. Cette fonction python retourne un objet que l'on peut utiliser avec toutes les fonctions de l'API de Csound.

## Crédits

Copyright (c) 2002 Maurizio Umberto Puxeddu. Tous droits réservés. Certaines parties, copyright (c) 2004 et 2005 Michael Gogins et (c) 2013, V. Lazzarini.

# pyrun Opcodes

pyrun — Exécute une instruction Python ou un bloc d'instructions.

## Syntaxe

```
pyrun "statement"
pyruni "statement"
pylrun "statement"
pylruni "statement"
pyrunt ktrigger, "statement"
pylrunt ktrigger, "statement"
```

## Description

Exécute l'instruction Python spécifiée au taux-k (*pyrun* et *pylrun*) ou au taux-i (*pyruni* et *pylruni*).

L'instruction est exécutée dans l'environnement global pour *pyrun* et *pyruni* ou dans l'environnement local pour *pylrun* et *pylruni*.

Ces opcodes n'effectuent aucune transmission de message. Cependant, comme leurs instructions ont accès aux espaces de noms globaux et privés, ils peuvent interagir avec des objets préalablement créés dans ces environnements.

Les versions « locales » des opcodes *pyexec* sont utiles lorsque le code exécuté par différentes instances d'un instrument ne doit pas interagir.

## Exemple du groupe d'opcodes pyrun

### Exemple 814. Orchestre

```
sr=44100
kr=4410
ksmps=10
nchnls=1

;Si vous n'exécutez pas CsoundAC, la ligne suivante est
;nécessaire pour initialiser l'interpréteur Python
;pyinit

pyruni "import random"

instr 1
    ; Ce message est stocké dans l'espace de noms principal
    ; et il est le même pour chaque instance
    pyruni "message = 'un nombre aléatoire global : %f' % random.random()"
    pyrun "print message"

    ; Ce message est stocké dans l'espace de noms privé
    ; et il est différent pour différentes instances
    pylruni "message = 'un nombre aléatoire privé : %f' % random.random()"
    pylrun "print message"
```

*endin*

### Exemple 815. Partition

*i1* 0 0.1

En exécutant cette partition on obtient des paires de messages entrelacées des deux instances de l'instrument 1.

Le premier message de chaque paire est stocké dans l'espace de noms principal et ainsi la seconde instance écrase le message de la première instance. En conséquence, le premier message sera le même pour les deux instances.

Le second message est différent pour les deux instances, étant stocké dans l'espace de noms privé.

## Crédits

Copyright (c) 2002 Maurizio Umberto Puxeddu. Tous droits réservés. Certaines parties, copyright (c) 2004 et 2005 Michael Gogins. Ce document a été mis à jour le 25 juillet 2004 et le 1er février 2005 par Michael Gogins.

# pwd

pwd — Demande au système d'exploitation le nom du répertoire courant.

## Description

Opcodes du greffon cs\_date.

**pwd** appelle le système d'exploitation pour déterminer le répertoire courant. **pwd** ne fonctionne que pendant les périodes d'initialisation.

## Syntaxe

Sres **pwd**

## Exécution

Sres -- la chaîne de caractères retournée.

## Exemple

Voici un exemple de l'opcode pwd. Il utilise le fichier *pwd.csd* [examples/pwd.csd].

### Exemple 816. Exemple de l'opcode pwd.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          ; -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o system.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
; Waits for command to execute before continuing
Swd pwd
puts Swd, 1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for minimal time.
i 1 0 0.1
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch  
Juillet 2012

Nouveau dans la version 5.18



# qinf

qinf — Teste si l'argument est un nombre infini.

## Description

Retourne le nombre de fois où l'argument n'est pas un nombre, avec le signe du premier infini.

## Syntaxe

**qinf**(x) (aucune restriction de taux)

## Exemples

Voici un exemple de l'opcode qinf. Il utilise le fichier *qinf.csd* [exemples/qinf.csd].

### Exemple 817. Exemple de l'opcode qinf.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
--rtaudio=alsa -o dac:hw:0
</CsOptions>
<CsInstruments>
nchnls = 2
ksmps = 400

#define WII_B          #3#
#define WII_A          #4#
#define WII_R_A        #304#
#define WII_PITCH       #20#
#define WII_ROLL        #21#

gkcnt init 1

instr 1
  i1 wiiconnect 3,1

      wiirange $WII_PITCH., -20, 0
  kt wiidata $WII_B.
  ka wiidata $WII_A.
  kra wiidata $WII_R_A.
  gka wiidata $WII_PITCH.
  gkp wiidata $WII_ROLL.
; If the B (trigger) button is pressed then activate a note
  if (kt==0) goto ee
  if (qinf(gka)) goto ee
  if (qinf(gkp)) goto ee
  event "i", 2, 0, 5
  gkcnt = gkcnt + 1
  printk2 kb
endin

instr 2
  a1 oscil ampdbs(gka), 440+gkp, 1
  outs a1, a1
```

```
    endin

    </CsInstruments>

    <CsScore>
    f1 0 4096 10 1
    i1 0 300

    </CsScore>

    </CsoundSynthesizer>
```

## Crédits

Ecrit par John ffitich.

Nouveau dans Csound 5.14

# qnan

qnan — Teste si l'argument n'est pas un nombre.

## Description

Retourne le nombre de fois où l'argument n'est pas un nombre.

## Syntaxe

**qnan**(x) (aucune restriction de taux)

## Exemples

Voici un exemple de l'opcode qnan. Il utilise le fichier *qnan.csd* [examples/qnan.csd].

### Exemple 818. Exemple de l'opcode qnan.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
--rtaudio=alsa -o dac:hw:0
</CsOptions>
<CsInstruments>
nchnls = 2
ksmps = 400

#define WII_B          #3#
#define WII_A          #4#
#define WII_R_A        #304#
#define WII_PITCH      #20#
#define WII_ROLL        #21#

gkcnt init 1

instr 1
  i1 wiiconnect 3,1

      wiirange $WII_PITCH., -20, 0
  kt wiidata $WII_B.
  ka wiidata $WII_A.
  kra wiidata $WII_R_A.
  gka wiidata $WII_PITCH.
  gkp wiidata $WII_ROLL.
; If the B (trigger) button is pressed then activate a note
  if (kt==0) goto ee
  if (qnan(gka)) goto ee
  if (qnan(gkp)) goto ee
  event "i", 2, 0, 5
  gkcnt = gkcnt + 1
  printk2 kb
endin

instr 2
  a1 oscil ampdbs(gka), 440+gkp, 1
  outs a1, a1
```

```
    endin

    </CsInstruments>

    <CsScore>
    f1 0 4096 10 1
    i1 0 300

    </CsScore>

    </CsoundSynthesizer>
```

## Crédits

Ecrit par John ffitich.

Nouveau dans Csound 5.14

# r2c

r2c — Conversion du format réel au format complexe.

## Description

Convertit un tableau de valeurs réelles en un tableau de valeurs complexes réelles-imaginaires entrelancées, en fixant la partie imaginaire à 0. La taille du tableau de sortie est le double de celle du tableau d'entrée. C'est une opération utilitaire pour faciliter les opérations à valeurs complexes sur des tableaux réels.

## Syntaxe

```
kout[] r2c kin[]
```

## Exécution

*kout[]* -- tableau de sortie contenant les valeurs complexes réelles-imaginaires. Créé s'il n'existe pas.

*kin[]* -- tableau d'entrée contenant les valeurs réelles.

## Exemples

Voici un exemple de l'opcode r2c. Il utilise le fichier *r2c.csd* [examplesr2c.csd].

### Exemple 819. Exemple de l'opcode r2c.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-d -o dac
</CsOptions>
<CsInstruments>
ksmps = 64

instr 1
  ifftsize = 1024
  kcnt init 0
  kIn[] init ifftsize
  kOut[] init ifftsize

  a1 oscili 0dbfs/2, 440

  if kcnt >= ifftsize then
    kCmplx[] r2c kIn
    kSpec[] fft kCmplx
    kCmplx fftinv kSpec
    kOut c2r kCmplx
    kcnt = 0
  endif

  kIn[] shiftin a1
  a2 shiftout kOut
  kcnt += ksmps
```

```
    out a2  
endin  
</CsInstruments>  
<CsScore>  
i1 0 10  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux.

## Crédits

Auteur : Victor Lazzarini  
NUI Maynooth  
2014

Nouveau dans la version 6.04

# rand

rand — Génère une suite contrôlée de nombres aléatoires.

## Description

La sortie est une suite contrôlée de nombres aléatoires entre *-amp* et *+amp*.

## Syntaxe

```
ares rand xamp [, iseed] [, isel] [, ioffset]  
kres rand xamp [, iseed] [, isel] [, ioffset]
```

## Initialisation

*iseed* (facultatif, par défaut=0,5) -- une graine pour la formule du calcul récursif des nombres pseudo-aléatoires. Une valeur comprise entre 0 et 1 produira une sortie initiale de *kamp \* iseed*. Avec une valeur supérieure à 1, la graine proviendra de l'horloge du système. Avec une valeur négative, la réinitialisation de la graine sera ignorée. La valeur par défaut est 0,5.

*isel* (facultatif, par défaut=0) -- s'il est nul, un nombre sur 16 bit est généré. S'il est non nul, un nombre sur 31 bit est généré. La valeur par défaut est 0.

*ioffset* (facultatif, par défaut=0) -- une valeur de base ajoutée au résultat aléatoire. Nouveau dans la version 4.03 de Csound.

## Exécution

*kamp*, *xamp* -- intervalle sur lequel les nombres aléatoires sont distribués.

*ares*, *kres* -- nombre aléatoire produit.

La formule pseudo-aléatoire interne produit des valeurs uniformément distribuées sur l'intervalle allant de *-kamp* à *+kamp*. *rand* génère ainsi un bruit blanc uniforme avec une valeur moyenne quadratique (RMS) de *kamp / (racine de 2)*.

La valeur de *ares* ou de *kres* se trouve dans un intervalle semi-ouvert qui contient *-xamp*, but mais pas *+xamp*.

## Exemples

Voici un exemple de l'opcode rand. Il utilise le fichier *rand.csd* [examples/rand.csd].

### Exemple 820. Exemple de l'opcode rand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>
```

```

; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o rand.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;same values every time

krnd rand 100
  printk .5, krnd ; look
aout oscili 0.8, 440+krnd, 1 ; & listen
outs aout, aout

endin

instr 2 ;different values every time

krnd rand 100, 10 ; seed from system clock
  printk .5, krnd ; look
aout oscili 0.8, 440+krnd, 1 ; & listen
outs aout, aout

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave.

i 1 0 1
i 2 2 1
e

</CsScore>
</CsoundSynthesizer>

```

L'exemple produit la sortie suivante :

```

i 1 1 time      0.00067:    50.00305
i 1 1 time      0.50000:    62.71362
i 1 1 time      1.00000:   -89.31885

WARNING: Seeding from current time 472230558

i 2 2 time      2.00067:   -70.65735
i 2 2 time      2.50000:    69.15283
i 2 2 time      3.00000:   -48.79761

```

## Voir aussi

*randh, randi*

## Crédits

Grâce à une note de John ffitch, j'ai changé les noms des paramètres.



# randh

randh — Génère des nombres aléatoires et les maintient pendant une certaine durée.

## Description

Génère des nombres aléatoires et les maintient pendant une certaine durée.

## Syntaxe

```
ares randh xamp, xcps [, iseed] [, isize] [, ioffset]
```

```
kres randh kamp, kcps [, iseed] [, isize] [, ioffset]
```

## Initialisation

*iseed* (facultatif, par défaut=0,5) -- une graine pour la formule du calcul récursif des nombres pseudo-aléatoires. Une valeur comprise entre 0 et +1 produira une sortie initiale de *kamp* \* *iseed*. Avec une valeur négative, la réinitialisation de la graine sera ignorée. Avec une valeur supérieure à 1, la graine proviendra de l'horloge du système ; c'est la meilleure option pour générer une séquence aléatoire différente à chaque utilisation.

*isize* (facultatif, par défaut=0) -- s'il est nul, un nombre sur 16 bit est généré. S'il est non nul, un nombre sur 31 bit est généré. La valeur par défaut est 0.

*ioffset* (facultatif, par défaut=0) -- une valeur de base ajoutée au résultat aléatoire. Nouveau dans la version 4.03 de Csound.

## Exécution

*kamp*, *xamp* -- intervalle sur lequel les nombres aléatoires sont distribués.

*kcps*, *xcps* -- fréquence à laquelle de nouveaux nombres aléatoires sont générés.

La formule pseudo-aléatoire interne produit des valeurs uniformément distribuées sur l'intervalle allant de  $-kamp$  à  $+kamp$ . *rand* génère ainsi un bruit blanc uniforme avec une valeur moyenne quadratique (RMS) de  $kamp / (\text{racine de } 2)$ .

Les autres unités produisent un bruit à bande limitée : les paramètres *kcps* et *xcps* permettent de choisir un taux de génération des nouveaux nombres aléatoires inférieur aux fréquences d'échantillonnage ou de contrôle. *randh* maintient chaque nouveau nombre durant le cycle spécifié.

## Exemples

Voici un exemple de l'opcode *randh*. Il utilise le fichier *randh.csd* [examples/randh.csd].

### Exemple 821. Exemple de l'opcode randh.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o randh.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;same values every time

krnd randh 100, 10
  printk2 krnd ; look
aout oscili 0.8, 440+krnd, 1 ; & listen
outs aout, aout

endin

instr 2 ;different values every time

krnd randh 100, 10, 10 ; seed from system clock
  printk2 krnd ; look
aout oscili 0.8, 440+krnd, 1 ; & listen
outs aout, aout

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave.

i 1 0 1
i 2 2 1
e
</CsScore>
</CsoundSynthesizer>

```

L'exemple produit la sortie suivante :

```

i1 50.00000
i1 50.00305
i1 97.68677
i1 -44.25354
i1 -61.56006
i1 -75.91248
i1 67.57202
i1 12.83875
i1 5.39551
i1 -95.18738

```

WARNING: Seeding from current time 684387922

```

i2 -13.81226
i2 -16.49475
i2 69.51904
i2 35.04944
i2 47.47925
i2 63.25378
i2 -59.61914
i2 50.93079

```

```
i2    -6.46362
i2    5.89294
```

## Voir aussi

*rand, randi*

# randi

rand — Génère une suite contrôlée de nombres aléatoires avec interpolation entre chaque nouveau nombre.

## Description

Génère une suite contrôlée de nombres aléatoires avec interpolation entre chaque nouveau nombre.

## Syntaxe

```
ares randi xamp, xcps [, iseed] [, isize] [, ioffset]  
kres randi kamp, kcps [, iseed] [, isize] [, ioffset]
```

## Initialisation

*iseed* (facultatif, par défaut=0,5) -- une graine pour la formule du calcul récursif des nombres pseudo-aléatoires. Une valeur comprise entre 0 et +1 produira une sortie initiale de *kamp* \* *iseed*. Avec une valeur négative, la réinitialisation de la graine sera ignorée. Avec une valeur supérieure à 1, la graine proviendra de l'horloge du système ; c'est la meilleure option pour générer une séquence aléatoire différente à chaque utilisation.

*isize* (facultatif, par défaut=0) -- s'il est nul, un nombre sur 16 bit est généré. S'il est non nul, un nombre sur 31 bit est généré. La valeur par défaut est 0.

*ioffset* (facultatif, par défaut=0) -- une valeur de base ajoutée au résultat aléatoire. Nouveau dans la version 4.03 de Csound.

## Exécution

*kamp*, *xamp* -- intervalle sur lequel les nombres aléatoires sont distribués.

*kcps*, *xcps* -- fréquence à laquelle de nouveaux nombres aléatoires sont générés.

La formule pseudo-aléatoire interne produit des valeurs uniformément distribuées sur l'intervalle allant de *-kamp* à *+kamp*. *rand* génère ainsi un bruit blanc uniforme avec une valeur moyenne quadratique (RMS) de *kamp* / (racine de 2).

Les autres unités produisent un bruit à bande limitée : les paramètres *kcps* et *xcps* permettent de choisir un taux de génération des nouveaux nombres aléatoires inférieur aux fréquences d'échantillonnage ou de contrôle. *randi* produit une interpolation linéaire entre chaque nouveau nombre et le précédent.

## Exemples

Voici un exemple de l'opcode *randi*. Il utilise le fichier *randi.csd* [examples/rand\_i.csd].

### Exemple 822. Exemple de l'opcode *randi*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o randi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;same values every time

krnd randi 100, 10
  printk .5, krnd ; look
aout oscili 0.8, 440+krnd, 1 ; & listen
outs aout, aout

endin

instr 2 ;different values every time

krnd randi 100, 10, 10 ; seed from system clock
  printk .5, krnd ; look
aout oscili 0.8, 440+krnd, 1 ; & listen
outs aout, aout

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave.

i 1 0 1
i 2 2 1
e
</CsScore>
</CsoundSynthesizer>

```

L'exemple produit la sortie suivante :

```

i 1 1 time      0.00067:    50.00000
i 1 1 time      0.50000:   -75.81672
i 1 1 time      1.00000:    95.93833

WARNING: Seeding from current time 1482746120

i 2 2 time      2.00067:   -17.94434
i 2 2 time      2.50000:   -14.11875
i 2 2 time      3.00000:   -72.41545

```

## Voir aussi

*rand*, *randh*

# random

`random` — Génère une suite contrôlée de nombres pseudo-aléatoires entre des valeurs minimale et maximale.

## Description

Génère une suite contrôlée de nombres pseudo-aléatoires entre des valeurs minimale et maximale.

## Syntaxe

```
ares random kmin, kmax
ires random imin, imax
kres random kmin, kmax
```

## Initialisation

*imin* -- limite inférieure de l'intervalle

*imax* -- limite supérieure de l'intervalle

## Exécution

*kmin* -- limite inférieure de l'intervalle

*kmax* -- limite supérieure de l'intervalle

L'opcode *random* est semblable à *linrand* et à *trirand* mais parfois je [Gabriel Maldonado] le trouve plus pratique car il permet de fixer arbitrairement les valeurs du minimum et du maximum.

## Exemples

Voici un exemple de l'opcode *random*. Il utilise le fichier *random.csd* [examples/random.csd].

### Exemple 823. Exemple de l'opcode *random*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o random.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
```

```

0dbfs = 1

instr 1 ;same values every time

krnd random 100, 1000
printk .5, krnd ; look
aout oscili 0.8, 440+krnd, 1 ; & listen
outs aout, aout

endin

instr 2 ;different values every time

seed 0
krnd random 100, 1000 ; seed from system clock
printk .5, krnd ; look
aout oscili 0.8, 440+krnd, 1 ; & listen
outs aout, aout

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave.

i 1 0 1
i 2 2 1
e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

i 1 time 0.00067: 894.58566
i 1 time 0.50000: 748.44281
i 1 time 1.00000: 328.29916

WARNING: Seeding from current time 1656666052

i 2 time 2.00067: 690.71466
i 2 time 2.50000: 459.42445
i 2 time 3.00000: 100.85594

```

## Voir aussi

*linrand, randomh, randomi, trirand*

## Crédits

Auteur : Gabriel Maldonado

# randomh

randomh — Génère des nombres aléatoires dans des limites définies par l'utilisateur et les maintient pendant une certaine durée.

## Description

Génère des nombres aléatoires dans des limites définies par l'utilisateur et les maintient pendant une certaine durée.

## Syntaxe

```
ares randomh kmin, kmax, xcps [,imode] [,ifirstval]
kres randomh kmin, kmax, kcps [,imode] [,ifirstval]
```

## Initialisation

*imode* (facultatif, 0 par défaut) -- mode de génération de la première valeur retournée (voir ci-dessous)

*ifirstval* (facultatif, 0 par défaut) -- première valeur retournée

## Exécution

*kmin* -- limite inférieure de l'intervalle

*kmax* -- limite supérieure de l'intervalle

*kcps*, *xcps* -- taux de génération des points aléatoires

L'opcode *randomh* est semblable à *randh* mais il permet à l'utilisateur de fixer arbitrairement les valeurs du minimum et du maximum.

Si *imode* = 0 (par défaut), la valeur de l'argument *kmin* est retournée pendant  $1/kcps$  (resp.  $1/xcps$ ) secondes au début de la note. Puis, le processus normal continue avec la génération et le maintien d'un nouveau nombre aléatoire toutes les  $1/kcps$  (resp.  $1/xcps$ ) secondes.

Si *imode* = 2, la valeur de l'argument *ifirstval* est retournée pendant  $1/kcps$  (resp.  $1/xcps$ ) secondes au début de la note. Puis, le processus normal continue avec la génération et le maintien d'un nouveau nombre aléatoire toutes les  $1/kcps$  (resp.  $1/xcps$ ) secondes.

Si *imode* = 3, le processus de génération commence avec un nombre aléatoire dès l'initialisation de la note.

## Exemples

Voici un exemple de l'opcode *randomh*. Il utilise le fichier *randomh.csd* [examples/randomh.csd].

### Exemple 824. Exemple de l'opcode randomh.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.



```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o randomh.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

seed 0

; Instrument #1.
instr 1
; Choose a random frequency between 220 and 440 Hz.
; Generate new random numbers at 10 Hz.
kmin   init 220
kmax   init 440
kcps   init 10
imode   =   p4
ifstval =   p5

printf_i "\nMode: %d\n", 1, imode
k1 randomh kmin, kmax, kcps, imode, ifstval
printk2 k1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second,
; each time with a different mode.
i 1 0 1
i 1 1 1 2 330
i 1 2 1 3
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie inclura des lignes comme celles-ci :

```

Mode: 0
i1 220.00000
i1 396.26079
i1 240.75446
i1 364.24577
...

Mode: 2
i1 330.00000
i1 416.50935
i1 356.11619
i1 433.59324
...

Mode: 3
i1 261.17741
i1 402.00891
i1 393.86592

```

```
il    307.19839
...
```

## Voir aussi

*randh, random, randomi*

## Crédits

Auteur : Gabriel Maldonado

Les arguments *imode* et *ifirstval* ont été ajoutés par François Pinot, en janvier 2011, après une discussion avec Peiman Khosravi sur la liste csnd.

Exemple écrit par Kevin Conder et adapté pour les nouveaux arguments par François Pinot.

# randomi

randomi — Génère une suite contrôlée de nombres aléatoires avec interpolation entre chaque nouveau nombre.

## Description

Génère une suite contrôlée de nombres aléatoires avec interpolation entre chaque nouveau nombre.

## Syntaxe

```
ares randomi kmin, kmax, xcps [,imode] [,ifirstval]
kres randomi kmin, kmax, kcps [,imode] [,ifirstval]
```

## Initialisation

*imode* (facultatif, 0 par défaut) -- mode du premier cycle d'interpolation (voir ci-dessous)

*ifirstval* (facultatif, 0 par défaut) -- première valeur retournée

## Exécution

*kmin* -- limite inférieure de l'intervalle

*kmax* -- limite supérieure de l'intervalle

*kcps*, *acps* -- taux de génération des points aléatoires

L'opcode *randomi* est semblable à *randi* mais il permet à l'utilisateur de fixer arbitrairement les valeurs du minimum et du maximum.

Si *imode* = 0 (par défaut), la valeur de l'argument *kmin* est retournée pendant  $1/kcps$  (resp.  $1/xcps$ ) secondes au début de la note, avant que le premier nombre aléatoire ne soit généré. Puis le processus d'interpolation démarre, d'abord entre *kmin* et le premier nombre aléatoire généré, et ensuite entre les nombres aléatoires générés successivement, chaque cycle d'interpolation durant  $1/kcps$  (resp.  $1/xcps$ ) secondes.

Si *imode* = 1, un nombre aléatoire est généré à l'initialisation et l'interpolation commence immédiatement entre la valeur de l'argument *kmin* et ce nombre aléatoire.

Si *imode* = 2, un nombre aléatoire est généré à l'initialisation et l'interpolation commence immédiatement entre la valeur de l'argument *ifirstval* et ce nombre aléatoire.

Si *imode* = 3, deux nombres aléatoires sont générés à l'initialisation et servent de bornes pour le premier cycle d'interpolation.

## Exemples

Voici un exemple de l'opcode *randomi*. Il utilise le fichier *randomi.csd* [examples/randomi.csd].

### Exemple 825. Exemple de l'opcode *randomi*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o randomi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

seed 0

; Instrument #1.
instr 1
; Choose a random frequency between 220 and 440.
; Generate new random numbers at 10 Hz.
kmin   init 220
kmax   init 440
kcps   init 10
imode   =    p4
ifstval =    p5

printf_i "\nMode: %d\n", 1, imode
k1 randomi kmin, kmax, kcps, imode, ifstval
printks "k1 = %f\n", 0.1, k1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
; each time with a different mode.
i 1 0 1
i 1 1 1 1
i 1 2 1 2 330
i 1 3 1 3
e

</CsScore>
</CsoundSynthesizer>

```

La sortie contiendra des lignes comme celles-ci :

```

Mode: 0
k1 = 220.000000
k1 = 220.000000
k1 = 220.146093
k1 = 246.827703
k1 = 395.595775
...

Mode: 1
k1 = 220.000000
k1 = 224.325329
k1 = 274.370074
k1 = 343.216049
k1 = 414.324347
...

```

```
Mode: 2
k1 = 330.000000
k1 = 292.628171
k1 = 334.519777
k1 = 290.610602
k1 = 394.905366
...
```

```
Mode: 3
k1 = 360.727674
k1 = 431.680412
k1 = 380.625254
k1 = 289.267139
k1 = 303.038109
...
```

## Voir aussi

*randi*, *random*, *randomh*

## Crédits

Auteur : Gabriel Maldonado

Les arguments *imode* et *ifirstval* ont été ajoutés par François Pinot, en janvier 2011, après une discussion avec Peiman Khosravi sur la liste csnd.

Exemple écrit par Kevin Conder et adapté pour les nouveaux arguments par François Pinot.

# rbjeq

rbjeq — Opcode de filtrage et d'égalisation paramétrique avec 7 types de filtre, basé sur un algorithme de Robert Bristow-Johnson.

## Description

Opcode de filtrage et d'égalisation paramétrique avec 7 types de filtre, basé sur un algorithme de Robert Bristow-Johnson.

## Syntaxe

ar **rbjeq** asig, kfco, klvl, kQ, kS[, imode]

## Initialisation

*imode* ( facultatif, 0 par défaut) - somme de :

- 1 : l'initialisation est ignorée (à n'utiliser qu'avec des notes liées ou réinitialisées)

et exactement une seule des valeurs suivantes pour sélectionner le type de filtre :

- 0 : filtre passe-bas résonant.  $kQ$  contrôle la résonance : à la fréquence de coupure ( $kfco$ ), le gain en amplitude est  $kQ$  (par exemple 20 dB pour  $kQ = 10$ ), les valeurs supérieures de  $kQ$  produisant un pic de résonance plus étroit. Si  $kQ$  vaut racine carrée de 0.5 (environ 0.7071), il n'y a pas de résonance et le filtre a une réponse ressemblant beaucoup à celle de *butterlp*. Si  $kQ$  est inférieur à racine carrée de 0.5, il n'y a pas de résonance et le filtre a une réponse de -6 dB / octave approximativement de  $kfco * kQ$  à  $kfco$ . Au-delà de  $kfco$ , l'atténuation est toujours de -12 dB / octave.



### NOTE

Le filtre passe-bas *rbjeq* se comporte sensiblement comme "ar **pareq** asig, kfco, 0, kQ, 2" mais il est calculé plus rapidement.

- 2 : filtre passe-haut résonant. Les paramètres sont les mêmes que pour le filtre passe-bas, mais le filtre équivalent est *butterhp* si  $kQ$  vaut 0.7071, et "ar **pareq** asig, kfco, 0, kQ, 1" dans les autres cas.
- 4 : filtre passe-bande.  $kQ$  contrôle la largeur de bande qui vaut  $kfco / kQ$ , et doit toujours être inférieur à  $sr / 2$ . La largeur de bande est mesurée entre les points à -3 dB (gain en amplitude = 0.7071), au-delà desquels la pente est de +/- 6 dB / octave. Ce type de filtre ressemble beaucoup à "ar **butterbp** asig, kfco, kfco / kQ".
- 6 : filtre réjecteur de bande avec les mêmes paramètres que le filtre passe-bande et une réponse semblable à celle de *butterbr*.
- 8 : EQ peak. Le gain en amplitude vaut 1 (0 dB) à 0 Hz et à  $sr / 2$ , et  $klvl$  à la fréquence centrale ( $kfco$ ). Ainsi,  $klvl$  contrôle le renforcement (s'il est supérieur à 1) ou l'atténuation (s'il est inférieur à 1).  $klvl$  à 1 produit une réponse plate. Comme pour les filtres passe-bande et réjecteur de bande, la largeur de bande est déterminées par  $kfco / kQ$  (qui doit être encore inférieur à  $sr / 2$ ) ; cependant, elle se trouve cette fois-ci entre les points situés à racine carrée de  $klvl$  (autrement dit à mi-renforcement ou mi-atténuation en décibels). NOTE : il faut éviter les valeurs de  $klvl$  excessivement faibles ou élevées, encore que l'opcode

ait été testé avec  $klvl = 0.01$  et  $klvl = 100$ .  $klvl = 0$  est toujours une erreur, contrairement au cas de *pareq* qui accepte un niveau de zéro.

- 10 : EQ low shelf, contrôlé par *klvl* et *kS* (*kQ* est ignoré par ce type de filtre). Le gain en amplitude est de *klvl* à la fréquence zéro tandis que le niveau des hautes fréquences (proches de  $sr / 2$ ) n'est pas changé. A la fréquence de coupure (*kfco*), le gain est de racine carrée de *klvl* (mi-renforcement ou mi-atténuation en décibels). Le paramètre *kS* contrôle la raideur de la pente de la réponse en fréquence (voir ci-dessous).
- 12 : EQ high shelf. Très semblable à l'EQ low shelf, mais il affecte la région des hautes fréquences.

La valeur par défaut de *imode* est zéro (filtre passe-bas, initialisation réalisée).

## Exécution

*ar* -- le signal de sortie.

*asig* -- le signal d'entrée.



### NOTE

Si l'entrée contient des sections silencieuses, il peut y avoir un ralentissement significatif sur les processeurs Intel du aux nombres dénormalisés. Dans de tels cas, il est recommandé de traiter le signal d'entrée avec l'opcode *denorm* avant le filtrage par *rbjeq* (et actuellement avec plusieurs autres filtres).

*kfco* -- fréquence de coupure ou fréquence centrale, selon le type de filtre, en Hz. Doit être supérieure à zéro et inférieure à  $sr / 2$  (l'intervalle compris entre  $sr * 0.0002$  et  $sr * 0.49$  devrait être sûr).

*klvl* -- niveau de renforcement ou d'atténuation, exprimé comme gain d'amplitude (par exemple, 1 : réponse plate, 4 : renforcement de 12 dB, 0.1 : atténuation de 20 dB) ; les valeurs nulle ou négatives sont interdites. Il est reconnu seulement par les types peak et shelf EQ (8, 10, 12) et ignoré par les autres filtres.

*kQ* -- résonance (également *kfco* / (largeur de bande) dans plusieurs types de filtre). N'est pas utilisé par les shelf EQs (*imode* = 10 et 12). La signification exacte de ce paramètre dépend du type de filtre (voir ci-dessus), mais il doit toujours être supérieur à zéro, et habituellement (*kfco* / *kQ*) doit être inférieur à  $sr / 2$ .

*kS* -- paramètre de pente pour les filtres shelf. Doit être supérieur à zéro ; plus la valeur est grande et plus la pente est raide, avec résonance si  $kS > 1$  (cependant, une valeur trop grande de *kS* peut rendre le filtre instable). Si *kS* vaut exactement 1, la pente est aussi raide que possible sans résonance. Noter que l'effet de *kS* - spécialement s'il est supérieur à 1 - dépend aussi de *klvl* et qu'il n'a pas d'unité bien définie.

## Exemples

Voici un exemple de l'opcode *rbjeq*. Il utilise le fichier *rbjeq.csd* [examples/rbjeq.csd].

### Exemple 826. Exemple de l'opcode *rbjeq*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
```

```

; For Non-realtime ouput leave only the line below:
; -o rbjeq.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

imode = p4
a1 vco2 .3, 155.6 ; sawtooth wave
kfco expon 8000, p3, 200 ; filter frequency
asig rbjeq a1, kfco, 1, kfco * 0.005, 1, imode
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 5 0 ;lowpass
i 1 6 5 2 ;highpass
i 1 12 5 4 ;bandpass
i 1 18 5 8 ;equalizer

e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Algorithme original de Robert Bristow-Johnson.

Version orchestre de Csound par Josep M Comajuncosas, août 1999.

Converti en C (avec optimisations et correction des bogues) par Istvan Varga, Décembre 2002.



# readclock

readclock — Lit la valeur d'une horloge interne.

## Description

Lit la valeur d'une horloge interne.

## Syntaxe

```
ir readclock inum
```

## Initialisation

*inum* -- le numéro d'une horloge. Il y a 32 horloges numérotées de 0 à 31. Toutes les autres valeurs correspondent à l'horloge numéro 32.

*ir* -- valeur, lors de la phase d'initialisation, de l'horloge spécifiée par *inum*.

## Exécution

Entre deux opcodes *clockon* et *clockoff*, le temps CPU utilisé est accumulé dans l'horloge. La précision dépend de la machine et elle est de l'ordre de la milliseconde sur les systèmes UNIX et Windows. L'opcode *readclock* lit la valeur courante d'une horloge pendant une phase d'initialisation.

## Exemples

Voici un exemple de l'opcode *readclock*. Il utilise le fichier *readclock.csd* [examples/readclock.csd].

### Exemple 827. Exemple de l'opcode readclock.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o readclock.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1.
instr 1
; Start clock #1.
clockon 1
```

```

; Do something that keeps Csound busy.
a1 oscili 10000, 440, 1
out a1
; Stop clock #1.
clockoff 1
; Print the time accumulated in clock #1.
i1 readclock 1
print i1
endin

</CsInstruments>
<CsScore>

; Initialize the function tables.
; Table 1: an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for one second starting at 0:00.
i 1 0 1
; Play Instrument #1 for one second starting at 0:01.
i 1 1 1
; Play Instrument #1 for one second starting at 0:02.
i 1 2 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

instr 1: i1 = 0.000
instr 1: i1 = 90.000
instr 1: i1 = 180.000

```

## Voir aussi

*clockoff, clockon*

## Crédits

Auteur : John ffitch  
 Université de Bath/Codemist Ltd.  
 Bath, UK  
 Juillet 1999

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.56 de Csound

# readf

readf — Lit une ligne de texte depuis un fichier externe.

## Description

Opcodes du greffon cs\_date.

Lit une ligne de texte depuis un fichier externe à chaque k-cycle.

## Syntaxe

Sres, kline **readf** ifilename

## Initialisation

*ifilename* -- un entier N indiquant un fichier nommé "input.N" ou une chaîne de caractères (entre guillemets, espaces autorisés) contenant le nom du fichier externe. Si c'est une chaîne de caractères, elle peut être un nom de chemin complet avec un répertoire spécifié ou bien un simple nom de fichier. Dans ce dernier cas, le fichier est d'abord cherché dans le répertoire courant, puis dans SSDIR et finalement dans SFDIR.

## Exécution

*Sres* -- variable contenant la ligne lue depuis *ifilename*.

*kline* -- numéro de la ligne lue ou -1 si la fin du fichier est atteinte.

Cet opcode permet de lire une ligne de texte depuis un fichier externe nommé. Il peut y avoir n'importe quel nombre d'opcodes *readf* dans un instrument ou dans un orchestre, mais ils lisent séparément depuis le même fichier ou depuis différents fichiers.

## Exemples

Voici un exemple de l'opcode *readf*. Il utilise le fichier *readf.csd* [examples/readf.csd].

### Exemple 828. Exemple de l'opcode *readf*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-n
</CsOptions>

<CsInstruments>
instr 1
  Swd
  pwd
  printf_i      "Working directory is '%s'\n", 1, Swd
  prints       "Reading myself =):\n"

  read:
    Sline, kLinNum readf "readf.csd"
    printf "Line %d: %s", kLinNum, kLinNum, Sline
```

```
    if kLinNum != -1 then
        kgoto read
    else
        turnoff
    endif
endin
</CsInstruments>

<CsScore>
i1 0 1
e
</CsScore>

</CsoundSynthesizer>
```

## Voir aussi

*readfi.*

## Crédits

John ffitch et Joachim Heintz

2012 ; nouveau dans la version 5.17.12

# readfi

readfi — Lit une ligne de texte depuis un fichier externe.

## Description

Opcodes du greffon cs\_date.

Lit une ligne de texte depuis un fichier externe une seule fois, à l'initialisation.

## Syntaxe

```
Sres, iline readfi ifilename
```

## Initialisation

*ifilename* -- un entier N indiquant un fichier nommé "input.N" ou une chaîne de caractères (entre guillemets, espaces autorisés) contenant le nom du fichier externe. Si c'est une chaîne de caractères, elle peut être un nom de chemin complet avec un répertoire spécifié ou bien un simple nom de fichier. Dans ce dernier cas, le fichier est d'abord cherché dans le répertoire courant, puis dans SSDIR et finalement dans SFDIR.

*iline* -- numéro de la ligne lue ou -1 si la fin du fichier est atteinte.

*Sres* -- variable contenant la ligne lue depuis *ifilename*.

Cet opcode permet de lire une ligne de texte depuis un fichier externe nommé. Il peut y avoir n'importe quel nombre d'opcodes *readfi* dans un instrument ou dans un orchestre, mais ils lisent séparément depuis le même fichier ou depuis différents fichiers.

## Exemples

Voici un exemple de l'opcode *readfi*. Il utilise le fichier *readfi.csd* [examples/readfi.csd].

### Exemple 829. Exemple de l'opcode *readfi*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-n
</CsOptions>

<CsInstruments>
instr 1
  Swd pwd
  printf_i "Working directory is '%s'\n", 1, Swd
  prints "Reading myself =):\n"
read:
  Sline, iLineNum readfi "readfi.csd"
  printf_i "Line %d: %s", iLineNum, iLineNum, Sline
  if iLineNum != -1 igoto read
endin
</CsInstruments>
```

```
<CsScore>  
i1 0 0.1  
e  
</CsScore>  
  
</CsoundSynthesizer>
```

## Voir aussi

*readf.*

## Crédits

John ffitch et Joachim Heintz

2012 ; nouveau dans la version 5.17.12

# readk

readk — Lit périodiquement la valeur d'un signal de contrôle de l'orchestre depuis un fichier externe.

## Description

Lit périodiquement la valeur d'un signal de contrôle de l'orchestre depuis un fichier externe dans un format spécifique.

## Syntaxe

```
kres readk ifilename, iformat, iprd
```

## Initialisation

*ifilename* -- un entier N indiquant un fichier nommé "readk.N" ou une chaîne de caractères (entre guillemets, espaces autorisés) contenant le nom du fichier externe. Si c'est une chaîne de caractères, elle peut être un nom de chemin complet avec un répertoire spécifié ou bien un simple nom de fichier. Dans ce dernier cas, le fichier est d'abord cherché dans le répertoire courant, puis dans SSDIR et finalement dans SFDIR.

*iformat* -- spécifie le format des données d'entrée :

- 1 = entiers signés sur 8 bit (char)
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers longs en ASCII (plein texte)
- 8 = flottants en ASCII (plein texte)

Noter que les formats A-law et U-law ne sont pas disponibles, et que tous les formats sauf les deux derniers sont binaires. Le fichier d'entrée doit être un fichier de données brutes sans en-tête.

*iprd* -- le taux (période) en secondes, arrondi à la période de contrôle de l'orchestre la plus proche, auquel le signal est lu depuis le fichier. Une valeur de 0 implique une période de contrôle (le minimum imposé), qui lira les nouvelles valeurs au taux de contrôle de l'orchestre. Avec des périodes plus longues, les mêmes valeurs seront répétées pendant plus d'une période de contrôle.

## Exécution

*kres* -- le signal lu depuis *ifilename*.

Cette opcode permet de lire la valeur d'un signal généré au taux de contrôle depuis un fichier externe nommé. Le fichier ne doit pas contenir d'en-tête d'information mais il doit contenir une suite temporelle de valeurs de contrôle échantillonnées régulièrement. Pour les formats de texte ASCII, les valeurs doivent être séparées par au moins un espace. Il peut y avoir n'importe quel nombre d'opcodes *readk* dans un instrument ou dans un orchestre et il peuvent lire à partir du même ou depuis différents fichiers.

## Exemples

Voici un exemple de l'opcode `readk`. Il utilise le fichier `readk.csd` [examples/readk.csd].

### Exemple 830. Exemple de l'opcode `readk`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o readk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

0dbfs = 1
; By Andres Cabrera 2008

instr 1
; Read a number from the file every 0.5 seconds
kfibo readk "fibonacci.txt", 7, 0.5
kpitchclass = 8 + ((kfibo % 12)/100)
printk2 kpitchclass
kcps = cpspch( kpitchclass )
printk2 kcps
a1 oscil 0.5, kcps, 1
out a1
endin

</CsInstruments>
<CsScore>
f 1 0 1024 10 1
i 1 0 10
e

</CsScore>
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode `readk`. Il utilise le fichier `readk-2.csd` [examples/readk-2.csd].

### Exemple 831. Exemple 2 de l'opcode `readk`.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac          ;;realtime audio out
;-iadc          ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o readk-2.wav -W ;; for file output any platform
```



```

</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 2^10, 10, 1

instr 1 ;writes a control signal to a file
kfreq randh 100, 1, 2, 1, 500 ;generates one random number between 400 and 600 per second
dumpk kfreq, "dumpk.txt", 8, 1 ;writes the control signal
printk 1, kfreq ;prints it
endin

instr 2 ;reads the file written by instr 1
kfreq readk "dumpk.txt", 8, 1
printk 1, kfreq ;prints it
aout poscil .2, kfreq, giSine
outs aout, aout
endin

</CsInstruments>
<CsScore>
i 1 0 5
i 2 5 5
e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie comprendra des lignes comme celles-ci :

WARNING: Seeding from current time 683384022

```

i 1 time 1.00033: 463.64510
i 1 time 2.00000: 463.64510
i 1 time 3.00000: 483.14200
i 1 time 4.00000: 567.55973
i 1 time 5.00000: 576.37060
i 1 time 6.00000: 460.66550

i 2 time 6.00033: 463.64510
i 2 time 7.00000: 463.64510
i 2 time 8.00000: 483.14200
i 2 time 9.00000: 567.55970
i 2 time 10.00000: 576.37060
i 2 time 11.00000: 460.66550

```

## Voir aussi

*dumpk, dumpk2, dumpk3, dumpk4, readk2, readk3, readk4*

## Crédits

Par : John ffitch and Barry L. Vercoe

1999 ou avant

# readk2

readk2 — Lit périodiquement les valeurs de deux signaux de contrôle de l'orchestre depuis un fichier externe.

## Description

Lit périodiquement les valeurs de deux signaux de contrôle de l'orchestre depuis un fichier externe.

## Syntaxe

```
kr1, kr2 readk2 ifilename, iformat, iprd
```

## Initialisation

*ifilename* -- un entier N indiquant un fichier nommé "readk.N" ou une chaîne de caractères (entre guillemets, espaces autorisés) contenant le nom du fichier externe. Si c'est une chaîne de caractères, elle peut être un nom de chemin complet avec un répertoire spécifié ou bien un simple nom de fichier. Dans ce dernier cas, le fichier est d'abord cherché dans le répertoire courant, puis dans *SSDIR* et finalement dans *SFDIR*.

*iformat* -- spécifie le format des données d'entrée :

- 1 = entiers signés sur 8 bit (char)
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers longs en ASCII (plein texte)
- 8 = flottants en ASCII (plein texte)

Noter que les formats A-law et U-law ne sont pas disponibles, et que tous les formats sauf les deux derniers sont binaires. Le fichier d'entrée doit être un fichier de données brutes sans en-tête.

*iprd* -- le taux (période) en secondes, arrondi à la période de contrôle de l'orchestre la plus proche, auquel les signaux sont lus depuis le fichier. Une valeur de 0 implique une période de contrôle (le minimum imposé), qui lira les nouvelles valeurs au taux de contrôle de l'orchestre. Avec des périodes plus longues, les mêmes valeurs seront répétées pendant plus d'une période de contrôle.

## Exécution

*kr1*, *kr2* -- les signaux lus depuis *ifilename*.

Cette opcode permet de lire les valeurs de deux signaux générés au taux de contrôle depuis un fichier externe nommé. Le fichier ne doit pas contenir d'en-tête d'information mais il doit contenir une suite temporelle de valeurs de contrôle échantillonnées régulièrement. Pour les formats binaires, les échantillons individuels de chaque signal sont alternés. Pour les formats de texte ASCII, les valeurs doivent être séparées par au moins un espace. Les deux "canaux" d'une trame peuvent se trouver sur la même ligne ou être séparés par un caractère de retour à la ligne. Il peut y avoir n'importe quel nombre d'opcodes *readk2* dans un instrument ou dans un orchestre et il peuvent lire à partir du même ou depuis différents fichiers.

## Exemples

Voici un exemple de l'opcode `readk2`. Il utilise le fichier `readk2.csd` [examples/readk2.csd].

### Exemple 832. Exemple de l'opcode `readk2`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o readk2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 2^10, 10, 1

instr 1 ;writes two control signals to a file
kfreq randh 100, 1, 2, 1, 500 ;generates one random number between 400 and 600 per second
kdb randh 12, 1, 2, 1, -12 ;amplitudes in dB between -24 and 0
dumpk2 kfreq, kdb, "dumpk2.txt", 8, 1 ;writes the control signals
prints "WRITING:\n"
printks "kfreq = %f, kdb = %f\n", 1, kfreq, kdb ;prints them
endin

instr 2 ;reads the file written by instr 1
kf,kdb readk2 "dumpk2.txt", 8, 1
prints "READING:\n"
printks "kfreq = %f, kdb = %f\n", 1, kf, kdb ;prints again
kdb lineto kdb, .1 ;smoothing amp transition
aout poscil ampdb(kdb), kf, giSine
outs aout, aout
endin

</CsInstruments>
<CsScore>
i 1 0 5
i 2 5 5
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie comprendra des lignes comme celles-ci :

```
kfreq = 429.202551, kdb = -20.495694
kfreq = 429.202551, kdb = -20.495694
kfreq = 407.275258, kdb = -23.123776
kfreq = 475.264472, kdb = -9.300846
kfreq = 569.979181, kdb = -7.315527
kfreq = 440.103457, kdb = -0.058331

kfreq = 429.202600, kdb = -20.495700
kfreq = 429.202600, kdb = -20.495700
kfreq = 407.275300, kdb = -23.123800
```

```
kfreq = 475.264500, kdb = -9.300800  
kfreq = 569.979200, kdb = -7.315500  
kfreq = 440.103500, kdb = -0.058300
```

## Voir aussi

*dumpk, dumpk2, dumpk3, dumpk4, readk, readk3, readk4*

## Crédits

Par : John ffitch and Barry L. Vercoe

1999 ou avant

# readk3

readk3 — Lit périodiquement les valeurs de trois signaux de contrôle de l'orchestre depuis un fichier externe.

## Description

Lit périodiquement les valeurs de trois signaux de contrôle de l'orchestre depuis un fichier externe.

## Syntaxe

```
kr1, kr2, kr3 readk3 ifilename, iformat, iprd
```

## Initialisation

*ifilename* -- un entier N indiquant un fichier nommé "readk.N" ou une chaîne de caractères (entre guillemets, espaces autorisés) contenant le nom du fichier externe. Si c'est une chaîne de caractères, elle peut être un nom de chemin complet avec un répertoire spécifié ou bien un simple nom de fichier. Dans ce dernier cas, le fichier est d'abord cherché dans le répertoire courant, puis dans *SSDIR* et finalement dans *SFDIR*.

*iformat* -- spécifie le format des données d'entrée :

- 1 = entiers signés sur 8 bit (char)
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers longs en ASCII (plein texte)
- 8 = flottants en ASCII (plein texte)

Noter que les formats A-law et U-law ne sont pas disponibles, et que tous les formats sauf les deux derniers sont binaires. Le fichier d'entrée doit être un fichier de données brutes sans en-tête.

*iprd* -- le taux (période) en secondes, arrondi à la période de contrôle de l'orchestre la plus proche, auquel les signaux sont lus depuis le fichier. Une valeur de 0 implique une période de contrôle (le minimum imposé), qui lira les nouvelles valeurs au taux de contrôle de l'orchestre. Avec des périodes plus longues, les mêmes valeurs seront répétées pendant plus d'une période de contrôle.

## Exécution

*kr1*, *kr2*, *kr3* -- les signaux lus depuis *ifilename*.

Cette opcode permet de lire les valeurs de trois signaux générés au taux de contrôle depuis un fichier externe nommé. Le fichier ne doit pas contenir d'en-tête d'information mais il doit contenir une suite temporelle de valeurs de contrôle échantillonnées régulièrement. Pour les formats binaires, les échantillons individuels de chaque signal sont alternés. Pour les formats de texte ASCII, les valeurs doivent être séparées par au moins un espace. Les trois "canaux" d'une trame peuvent se trouver sur la même ligne ou être séparés par un caractère de retour à la ligne. Il peut y avoir n'importe quel nombre d'opcodes *readk3* dans un instrument ou dans un orchestre et il peuvent lire à partir du même ou depuis différents fichiers.

## Exemples

Voici un exemple de l'opcode `readk3`. Il utilise le fichier `readk3.csd` [examples/readk3.csd].

### Exemple 833. Exemple de l'opcode `readk3`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o readk3.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 2^10, 10, 1

instr 1 ;writes three control signals to a file
kfreq randh 100, 1, 2, 1, 500 ;generates one random number between 400 and 600 per second
kdb randh 12, 1, 2, 1, -12 ;amplitudes in dB between -24 and 0
kpan randh .5, 1, 2, 1, .5 ;panning between 0 and 1
dumpk3 kfreq, kdb, kpan, "dumpk3.txt", 8, 1 ;writes the control signals
prints "WRITING:\n"
printks "kfreq = %f, kdb = %f, kpan = %f\n", 1, kfreq, kdb, kpan ;prints them
endin

instr 2 ;reads the file written by instr 1
kf,kdb,kp readk3 "dumpk3.txt", 8, 1
prints "READING:\n"
printks "kfreq = %f, kdb = %f, kpan = %f\n", 1, kf, kdb, kp ;prints again
kdb lineto kdb, .1 ;smoothing amp transition
kp lineto kp, .1 ;smoothing pan transition
aout poscil ampdb(kdb), kf, giSine
aL, aR pan2 aout, kp
outs aL, aR
endin

</CsInstruments>
<CsScore>
i 1 0 5
i 2 5 5
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie comprendra des lignes comme celles-ci :

```
WRITING:
kfreq = 473.352855, kdb = -15.197657, kpan = 0.366764
kfreq = 473.352855, kdb = -15.197657, kpan = 0.366764
kfreq = 441.426368, kdb = -19.026206, kpan = 0.207327
kfreq = 452.965140, kdb = -21.447486, kpan = 0.553270
kfreq = 585.106328, kdb = -11.903852, kpan = 0.815665
kfreq = 482.056760, kdb = -4.046744, kpan = 0.876537
```

```
READING:
kfreq = 473.352900, kdb = -15.197700, kpan = 0.366800
kfreq = 473.352900, kdb = -15.197700, kpan = 0.366800
kfreq = 441.426400, kdb = -19.026200, kpan = 0.207300
kfreq = 452.965100, kdb = -21.447500, kpan = 0.553300
kfreq = 585.106300, kdb = -11.903900, kpan = 0.815700
kfreq = 482.056800, kdb = -4.046700, kpan = 0.876500
```

## Voir aussi

*dumpk, dumpk2, dumpk3, dumpk4, readk, readk2, readk4*

## Crédits

Par : John ffitch and Barry L. Vercoe

1999 ou avant

# readk4

`readk4` — Lit périodiquement les valeurs de quatre signaux de contrôle de l'orchestre depuis un fichier externe.

## Description

Lit périodiquement les valeurs de quatre signaux de contrôle de l'orchestre depuis un fichier externe.

## Syntaxe

```
kr1, kr2, kr3, kr4 readk4 ifilename, iformat, iprd
```

## Initialisation

*ifilename* -- un entier N indiquant un fichier nommé "readk.N" ou une chaîne de caractères (entre guillemets, espaces autorisés) contenant le nom du fichier externe. Si c'est une chaîne de caractères, elle peut être un nom de chemin complet avec un répertoire spécifié ou bien un simple nom de fichier. Dans ce dernier cas, le fichier est d'abord cherché dans le répertoire courant, puis dans *SSDIR* et finalement dans *SFDIR*.

*iformat* -- spécifie le format des données d'entrée :

- 1 = entiers signés sur 8 bit (char)
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers longs en ASCII (plein texte)
- 8 = flottants en ASCII (plein texte)

Noter que les formats A-law et U-law ne sont pas disponibles, et que tous les formats sauf les deux derniers sont binaires. Le fichier d'entrée doit être un fichier de données brutes sans en-tête.

*iprd* -- le taux (période) en secondes, arrondi à la période de contrôle de l'orchestre la plus proche, auquel les signaux sont lus depuis le fichier. Une valeur de 0 implique une période de contrôle (le minimum imposé), qui lira les nouvelles valeurs au taux de contrôle de l'orchestre. Avec des périodes plus longues, les mêmes valeurs seront répétées pendant plus d'une période de contrôle.

## Exécution

*kr1*, *kr2*, *kr3*, *kr4* -- les signaux lus depuis *ifilename*.

Cette opcode permet de lire les valeurs de quatre signaux générés au taux de contrôle depuis un fichier externe nommé. Le fichier ne doit pas contenir d'en-tête d'information mais il doit contenir une suite temporelle de valeurs de contrôle échantillonnées régulièrement. Pour les formats binaires, les échantillons individuels de chaque signal sont alternés. Pour les formats de texte ASCII, les valeurs doivent être séparées par au moins un espace. Les quatre "canaux" d'une trame peuvent se trouver sur la même ligne ou être séparés par un caractère de retour à la ligne. Il peut y avoir n'importe quel nombre d'opcodes *readk4* dans un instrument ou dans un orchestre et il peuvent lire à partir du même ou depuis différents fichiers.



## Exemples

Voici un exemple de l'opcode `readk4`. Il utilise le fichier `readk4.csd` [examples/readk4.csd].

### Exemple 834. Exemple de l'opcode `readk4`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o readk4.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 2^10, 10, 1

instr 1 ;writes four control signals to a file
kcf      randh      950, 1, 2, 1, 1050 ;generates one random number between 100 and 2000 per second
kq       randh      10, 1, 2, 1, 11 ;generates another random number between 1 and 21 per second
kdb      randh      9, 1, 2, 1, -15 ;amplitudes in dB between -24 and -6
kpan     randh      .5, 1, 2, 1, .5 ;panning between 0 and 1
          dumpk4     kcf, kq, kdb, kpan, "dumpk4.txt", 8, 1 ;writes the control signals
          prints      "WRITING:\n"
          printks     "kcf = %f, kq = %f, kdb = %f, kpan = %f\n", 1, kcf, kq, kdb, kpan ;prints them
endin

instr 2 ;reads the file written by instr 1
kcf,kq,kdb,kp readk4 "dumpk4.txt", 8, 1
          prints      "READING:\n"
          printks     "kcf = %f, kq = %f, kdb = %f, kpan = %f\n", 1, kcf, kq, kdb, kp ;prints values
kdb      lineto     kdb, .1 ;smoothing amp transition
kp       lineto     kp, .1 ;smoothing pan transition
anoise   rand       ampdb(kdb), 2, 1
kbw      =          kcf/kq ;bandwidth of resonant filter
abp      reson      anoise, kcf, kbw
aout     balance    abp, anoise
aL, aR   pan2       aout, kp
          outs        aL, aR
endin

</CsInstruments>
<CsScore>
i 1 0 5
i 2 5 5
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie comprendra des lignes comme celles-ci :

```
WRITING:
kcf = 1122.469723, kq = 11.762839, kdb = -14.313445, kpan = 0.538142
kcf = 1122.469723, kq = 11.762839, kdb = -14.313445, kpan = 0.538142
```

```
kcf = 1148.638412, kq = 12.040490, kdb = -14.061868, kpan = 0.552205  
kcf = 165.796855, kq = 18.523179, kdb = -15.816977, kpan = 0.901528  
kcf = 147.729960, kq = 13.071911, kdb = -11.924531, kpan = 0.982518  
kcf = 497.430113, kq = 13.605512, kdb = -21.586611, kpan = 0.179229
```

READING:

WARNING: Seeding from current time 3308160476

```
kcf = 1122.469700, kq = 11.762800, kdb = -14.313400, kpan = 0.538100  
kcf = 1122.469700, kq = 11.762800, kdb = -14.313400, kpan = 0.538100  
kcf = 1148.638400, kq = 12.040500, kdb = -14.061900, kpan = 0.552200  
kcf = 165.796900, kq = 18.523200, kdb = -15.817000, kpan = 0.901500  
kcf = 147.730000, kq = 13.071900, kdb = -11.924500, kpan = 0.982500  
kcf = 497.430100, kq = 13.605500, kdb = -21.586600, kpan = 0.179200
```

## Voir aussi

*dumpk, dumpk2, dumpk3, dumpk4, readk, readk2, readk3*

## Crédits

Par : John ffitich and Barry L. Vercoe

1999 ou avant

# readscore

readscore — Lecture, prétraitement et planification d'une partition depuis une chaîne.

## Description

*readscore* produit un ou plusieurs événements de partition. Il gère les chaînes de caractères dans les mêmes conditions que la partition standard, y compris le prétraitement (report, tri, rampes, etc). Les chaînes sur plusieurs lignes sont acceptées et elles sont délimitées par `{ { }`.

## Syntaxe

`readscore Sin`

## Initialisation

« *Sin* » -- une chaîne de caractères entre guillemets ou délimitée par `{ { }` et contenant un ou plusieurs événements de partition.

## Exemples

Voici un exemple de l'opcode *readscore*. Il utilise le fichier *readscore.csd* [examples/readscore.csd].

### Exemple 835. Exemple de l'opcode *readscore*.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-d -o dac
</CsOptions>
<CsInstruments>

instr 1
a1 flooper2 1000,p4,2,5,0.1,1
  out a1
endin

instr 2
ires readscore {{
{12 COUNT
i1 $COUNT 1 [1 + $COUNT/12]
}
}}
endin

</CsInstruments>
<CsScore>
f0 12
f 1 0 0 1 "fox.wav" 0 0 1

i2 0 1
```

```
</CsScore>
</CsoundSynthesizer>
```

On peut utiliser des opcodes de chaîne de caractères comme *sprintfk* pour générer les chaînes à passer à *readscore* comme ceci :

```
Sfil = "/Volumes/Bla/file.aif"
String sprintfk {{i 2 0 %f "%s" %f %f %f %f}}, idur, Sfil, p5, p6, knorm, iskip
readscore String
```

## Voir aussi

*event*, *event\_i*, *schedule*, *schedwhen*, *schedkwhen*, *schedkwhennamed*

## Crédits

Auteur : Victor Lazzarini, 2013

# readscratch

readscratch — Retourne une valeur enregistrée dans l'instance d'un instrument.

## Description

L'opcode *readscratch* retourne l'une des quatre valeurs scalaires que lon peut enregistrer dans l'instance d'un instrument.

## Syntaxe

```
ival readscratch[index]
```

## Initialisation

*ival* -- variable pour le résultat.

*index* -- quelle valeur lire ; vaut zéro par défaut.

## Exemples

Voici un exemple de l'opcode readscratch. Il utilise le fichier *readscratch.csd* [examples/readscratch.csd].

### Exemple 836. Exemple de l'opcode readscratch.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ilast readscratch
writescratch p2
ilen readscratch 1
writescratch p3, 1
printf_i "last run at %f for %f\n", ilen, ilast, ilen
endin

</CsInstruments>
<CsScore>
i 1 0 1
i 1 2 3
i 1 6 10
e
</CsScore>
```

`</CsoundSynthesizer>`

## Voir aussi

*writescratch,*

## Crédits

Auteur : John ffitch  
Mars 2013

Nouveau dans la version 6.00 de Csound.

# rect2pol

rect2pol — Conversion du format rectangulaire au format polaire.

## Description

Convertit un tableau d'entrée du format réel-imaginaire au format module-argument.

## Syntaxe

```
kout[] rect2pol kin[]
```

## Exécution

*kout[]* -- tableau contenant les valeurs complexes module-argument de sortie. Créé s'il n'existe pas.

*kin[]* -- tableau contenant les valeurs complexes réelles-imaginaires d'entrée.

## Exemples

Voici un exemple de l'opcode rect2pol. Il utilise le fichier *rect2pol.csd* [examples/rect2pol.csd].

### Exemple 837. Exemple de l'opcode rect2pol.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-d -o dac
</CsOptions>
<CsInstruments>
/* ksmpls needs to be an integer div of
   hopsize */
ksmps = 64

instr 1

  ihopsize = 256 ; hopsize
  ifftsize = 1024 ; FFT size
  iolaps = ifftsize/ihopsize ; overlaps
  ibw = sr/ifftsize ; bin bandwidth
  kcnt init 0 ; counting vars
  krow init 0

  kOla[] init ifftsize ; overlap-add buffer
  kIn[] init ifftsize ; input buffer
  kOut[][] init iolaps, ifftsize ; output buffers

  a1 diskin2 "fox.wav",1,0,1 ; audio input

  /* every hopsize samples */
  if kcnt == ihopsize then
    /* window and take FFT */
    kWin[] window kIn,krow*ihopsize
    kSpec[] rfft kWin
```

```

kSpec rect2pol kSpec

/* reduce mags between high and low freqs */
ilow = 0
ihigh = 1000
ki = int(ilow/ibw)
until ki >= int(ihigh/ibw) do
    kSpec[ki] = kSpec[ki]*0.1
    ki += 2
od

kSpec pol2rect kSpec

/* IFFT + window */
kRow[] rfft kSpec
kWin window kRow, krow*ihopsize
/* place it on out buffer */
kOut setrow kWin, krow

/* zero the ola buffer */
kOla = 0
/* overlap-add */
ki = 0
until ki == iolaps do
    kRow getrow kOut, ki
    kOla = kOla + kRow
    ki += 1
od

/* update counters */
krow = (krow+1)%iolaps
kcnt = 0
endif

/* shift audio in/out of buffers */
kIn shiftin a1
a2 shiftout kOla
out a2/iolaps

/* increment counter */
kcnt += ksmpls

endin

</CsInstruments>
<CsScore>
i1 0 10
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux.

## Crédits

Auteur : Victor Lazzarini  
 NUI Maynooth  
 2014

Nouveau dans la version 6.04



# reinit

reinit — Suspend une exécution tandis que se déroule une phase spéciale d'initialisation.

## Description

Suspend une exécution tandis que se déroule une phase spéciale d'initialisation.

Chaque fois que cette instruction est rencontrée durant une phase d'exécution, celle-ci est temporairement suspendue tandis qu'une phase spéciale d'initialisation, commençant à *label* et allant jusqu'à *return* ou *endin*, a lieu. L'exécution reprend ensuite à partir de l'endroit où elle fut interrompue.

## Syntaxe

```
reinit label
```

## Exemples

Les instructions suivantes génèrent un signal de contrôle exponentiel dont les valeurs vont de 440 à 880 exactement dix fois pendant la durée p3. Elles utilisent le fichier *reinit.csd* [examples/reinit.csd].

### Exemple 838. Exemple de l'opcode reinit.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o reinit.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

instr 1

reset:
    timeout 0, p3/10, contin
    reinit reset

contin:
    kLine expon 440, p3/10, 880
    aSig oscil 10000, kLine, 1
    out aSig
    rireturn

endin

</CsInstruments>
```

```
<CsScore>

f1 0 4096 10 1

i1 0 10
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*rigoto, rireturn*

# release

release — Indique si une note est dans sa phase de « relâchement ».

## Description

Fournit le moyen de savoir quand un message note off est reçu pour la note courante. Seul un message note off ayant le même numéro de note MIDI que celui qui a déclenché la note sera rapporté par *release*.

## Syntaxe

```
kflag release
```

## Exécution

*kflag* -- indique si la note est dans sa phase de « relâchement ». (1 si un note off est reçu, 0 sinon)

*release* retourne l'état de la note courante. Si la note courante est dans sa phase de « relâchement » (c'est-à-dire si sa durée a été étendue avec l'opcode *xtratim* et si elle vient d'être désactivée), l'argument de sortie *kflag* prend la valeur 1. Sinon (dans la phase d'entretien de la note courante), *kflag* vaut 0.

Cet opcode est utile pour implémenter des enveloppes complexes avec relâchement. Lorsqu'il est utilisé avec *xtratim* il peut fournir une alternative au comportement prédéterminé des opcodes "r" tels que *linsegr* et *expsegr*, dans lesquels le temps de relâchement est fixé à la durée maximale spécifiée dans l'instrument actif.

## Exemples

Voir les exemples de *xtratim*.

## Voir aussi

*xtratim*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.47 de Csound.

# remoteport

remoteport — Définit le port à utiliser sur le système distant.

## Description

Définit le port à utiliser avec les opcodes *insremot*, *midremot*, *insglobal* et *midglobal*.

## Syntaxe

```
remoteport iportnum
```

## Initialisation

*iportnum* -- numéro du port à utiliser. S'il est nul ou négatif, le port 40002 est sélectionné par défaut.

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Novembre, 2006

Nouveau dans la version 5.05 de Csound.

# remove

`remove` — Supprime la définition d'un instrument.

## Description

Supprime la définition d'un instrument tant qu'il n'est pas utilisé.

## Syntaxe

```
remove insnum
```

## Initialisation

*insnum* -- numéro ou nom de l'instrument à effacer

## Exécution

Tant que l'instrument indiqué n'est pas actif, *remove* efface l'instrument et la mémoire qui lui est associée. A employer avec précaution car son utilisation peut conduire à un plantage dans certains cas.

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Juin, 2006

Nouveau dans la version 5.04 de Csound

# repluck

repluck — Modèle physique de corde pincée.

## Description

*repluck* est une implémentation du modèle physique de corde pincée. On peut contrôler le point d'excitation, le point de lecture, le filtre, et un signal audio additionnel, *excite*. *excite* est utilisé pour exciter la "corde". Basé sur l'algorithme de Karplus-Strong.

## Syntaxe

```
ares repluck iplk, kamp, icps, kpick, krefl, excite
```

## Initialisation

*iplk* -- Le point d'excitation est *iplk*, qui représente une fraction de la longueur de la corde (0 à 1). Un point d'excitation de zéro signifie l'absence d'excitation initiale.

*icps* -- La corde produit une hauteur de *icps*.

## Exécution

*kamp* -- Amplitude de la note.

*kpick* -- Fraction de la longueur de la corde où sera lue la sortie.

*krefl* -- le coefficient de réflexion, indiquant l'amortissement et le taux d'extinction. Il doit être strictement compris entre 0 et 1 (il n'acceptera pas 0 ni 1).

## Exécution

*excite* -- Un signal d'excitation de la corde.

## Exemples

Voici un exemple de l'opcode repluck. Il utilise le fichier *repluck.csd* [examples/repluck.csd].

### Exemple 839. Exemple de l'opcode repluck.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o repluck.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1

iplk = 0.75
kamp = .8
icps = 110
krefl = p4
kpick = p5

axcite oscil 1, 1, 1
asig repluck iplk, kamp, icps, kpick, krefl, axcite
asig dcblock2 asig ;get rid of DC offset
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave.

s
i 1 0 1 0.95 0.75 ;sounds heavier (=p5)
i 1 + 1 <
i 1 + 1 <
i 1 + 1 <
i 1 + 10 0.6

s
i 1 0 1 0.95 0.15 ;sounds softer (=p5)
i 1 + 1 <
i 1 + 1 <
i 1 + 1 <
i 1 + 10 0.6
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*wgpluck2*

## Crédits

Auteur : John ffitch  
 Université de Bath/Codemist Ltd.  
 Bath, UK  
 1997

Nouveau dans la version 3.47

# reshapearray

reshapearray — Reformate un tableau en gardant sa capacité.

## Description

Opcode du greffon emugens.

L'opcode *reshapearray* change le format d'un tableau, à condition que cette opération ne modifie pas sa capacité. On peut l'utiliser pour convertir un tableau 1D en tableau 2D ou vice-versa, ou simplement pour modifier la taille de ses dimensions, tant que leur produit reste constant.

## Syntaxe

```
reshapearray array[, isize0 [, isize1 ]
```

## Initialisation

*array[]* -- Le tableau à reformater (l'opération se fait sans réaffectation).

*isize0* -- La taille de la première dimension.

*isize1* -- La taille de la seconde dimension (0 pour les tableaux 1D). Vaut 0 par défaut.

## Exmples

Voici un exemple de l'opcode reshapearray. Il utilise le fichier *reshapearray.csd* [exemples/reshapearray.csd].

### Exemple 840. Exemple de l'opcode reshapearray.

```
<CsoundSynthesizer>
<CsOptions>
;-odac      ;;;realtime audio out

</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 64
nchnls = 2
0dbfs = 1

; This is the example file for reshapearray

/*

reshapearray

  Reshape an array, maintaining the capacity of the array
  (it does NOT resize the array).

  You can reshape a 2D array to another array of equal capacity
  of reshape a 1D array to a 2D array, or a 2D array to a 1D
```



```

    array

    reshapearray array[], inumrows, inumcols=0

    works with i and k arrays, at i-time and k-time

*/

instr 1
    ivalues[] fillarray 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
    printarray ivalues
    reshapearray ivalues, 3, 4
    printarray ivalues
endin

instr 2
    kxs[][] init 3, 4
    kxs fillarray 0, 1, 2, 3, \
                10, 11, 12, 13, \
                20, 21, 22, 23
    reshapearray kxs, 4, 3
    printarray kxs, 1, "", "kxs after"
    turnoff
endin

</CsInstruments>

<CsScore>
i 1 0 0.01
i 2 1 0.05
</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

*printarray, getrow, slicearray,*

## Crédits

Auteur : Eduardo Moguillansky 2018

Nouveau greffon dans la version 6.12

# reson

reson — Un filtre à résonance du second ordre.

## Description

Un filtre à résonance du second ordre.

## Syntaxe

```
ares reson asig, xcf, xbw [, iscl] [, iskip]
```

## Initialisation

*iscl* (facultatif, 0 par défaut) -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc. Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

*iskip* (facultatif, 0 par défaut) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*ares* -- le signal de sortie au taux audio.

*asig* -- le signal d'entrée au taux audio.

*xcf* -- la fréquence centrale du filtre, ou position fréquentielle de la crête de la réponse.

*xbw* -- largeur de bande du filtre (la différence en Hz entre les points haut et bas à mi-puissance).

*reson* est un filtre de second ordre dans lequel *kcf* contrôle la fréquence centrale, ou position fréquentielle de la crête de la réponse, et *kbw* contrôle sa largeur de bande (la différence en fréquence entre les points haut et bas à mi-puissance).

## Exemples

Voici un exemple de l'opcode *reson*. Il utilise le fichier *reson.csd* [examples/reson.csd].

### Exemple 841. Exemple de l'opcode *reson*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
```

```

<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o reson.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1

asaw vco2 .2, 220 ;sawtooth
kcf line 220, p3, 1760 ;vary cut-off frequency from 220 to 1280 Hz
kbw = p4 ;vary bandwidth of filter too
ares reson asaw, kcf, kbw
asig balance ares, asaw
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 4 10 ;bandwidth of filter = 10 Hz
i 1 + 4 50 ;50 Hz and
i 1 + 4 200 ;200 Hz
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*areson, aresonk, atone, atonek, port, portk, resonk, tone, tonek*

## Crédits

Paramètres de taux audio introduits dans la version 6.02

Octobre 2013.

# resonk

resonk — Un filtre à résonance du second ordre.

## Description

Un filtre à résonance du second ordre.

## Syntaxe

```
kres resonk ksig, kcf, kbw [, iscl] [, iskip]
```

## Initialisation

*iscl* (facultatif, 0 par défaut) -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. (Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc.) Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

*iskip* (facultatif, 0 par défaut) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*kres* -- le signal de sortie au taux de contrôle.

*ksig* -- le signal d'entrée au taux de contrôle.

*kcf* -- la fréquence centrale du filtre, ou position fréquentielle du pic de la réponse.

*kbw* -- largeur de bande du filtre (la différence en Hz entre les points haut et bas à mi-puissance).

*resonk* est semblable à *reson* à part le fait que sa sortie se fait au taux de contrôle plutôt qu'au taux audio.

## Exemples

Voici un exemple de l'opcode *resonk*. Il utilise le fichier *resonk.csd* [examples/resonk.csd].

### Exemple 842. Exemple de l'opcode *resonk*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform
```

```

-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o resonk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisin ftgen 0, 0, 2^10, 10, 1

instr 1

ksig randomh 400, 1800, 150
aout poscil .2, 1000+ksig, gisin
outs aout, aout
endin

instr 2

ksig randomh 400, 1800, 150
khp line 1, p3, 400 ;vary high-pass
ksig resonk ksig, khp, 50
aout poscil .2, 1000+ksig, gisin
outs aout, aout
endin

</CsInstruments>
<CsScore>
i 1 0 5
i 2 5.5 5
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*areson, aresonk, atone, atonek, port, portk, reson, tone, tonek*

## Crédits

Auteur : Robin Whittle  
 Australie  
 Mai 1997

# resonr

*resonr* — Un filtre passe-bande avec une réponse en fréquence variable.

## Description

Implémentation d'un filtre passe-bande du second ordre à deux pôles et deux zéros, avec une réponse en fréquence variable.

## Syntaxe

```
ares resonr asig, xcf, xbw [, iscl] [, iskip]
```

## Initialisation

Les variables d'initialisation facultatives de *resonr* sont identiques aux variables de taux-i de *reson*.

*iscl* (facultatif, 0 par défaut) -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc. Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

*iskip* (facultatif, 0 par défaut) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal d'entrée à filtrer

*xcf* -- fréquence de coupure ou de résonance du filtre, mesurée en Hz

*xbw* -- largeur de bande du filtre (la différence en Hz entre les points à mi-puissance inférieur et supérieur).

*resonr* et *resonz* sont des variations du résonateur passe-bande classique à deux pôles (*reson*). Ces deux filtres ont deux zéros dans leur fonction de transfert en plus des deux pôles. Les zéros de *resonz* se trouvent à  $z = 1$  et à  $z = -1$ . Les zéros de *resonr* se trouvent à  $+racine\_carrée(R)$  et à  $-racine\_carrée(R)$ , où  $R$  est le rayon des pôles dans le plan complexe des  $z$ . L'ajout de zéros à *resonr* et à *resonz* améliore la sélectivité de la magnitude de la réponse de ces filtres aux fréquences de coupure proches de 0, ceci au prix d'une moins grande sélectivité aux fréquences supérieures à la crête de la fréquence de coupure.

*resonr* et *resonz* sont très proches du gain constant lorsque la fréquence centrale glisse, ce qui donne un contrôle plus efficace de la magnitude de la réponse qu'avec les résonateurs à deux pôles traditionnels tels que *reson*.

*resonr* et *resonz* produisent une sonorité considérablement différente de celle de *reson*, spécialement pour les faibles fréquences centrales ; la méthode par tâtonnement est la meilleure façon de déterminer quel résonateur est le plus adapté à une application particulière.

## Exemples

Voici un exemple des opcodes `resonr` et `resonz`. Il utilise le fichier `resonr.csd` [exemples/resonr.csd].

### Exemple 843. Exemple des opcodes `resonr` et `resonz`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o resonr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Sean Costello */
; Orchestra file for resonant filter sweep of a sawtooth-like waveform.
; The outputs of reson, resonr, and resonz are scaled by coefficients
; specified in the score, so that each filter can be heard on its own
; from the same instrument.

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1

    idur      =      p3
    ibegfreq  =      p4                ; beginning of sweep frequency
    iendfreq  =      p5                ; ending of sweep frequency
    ibw       =      p6                ; bandwidth of filters in Hz
    ifreq     =      p7                ; frequency of gbuzz that is to be filtered
    iamp      =      p8                ; amplitude to scale output by
    ires      =      p9                ; coefficient to scale amount of reson in output
    iresr     =      p10               ; coefficient to scale amount of resonr in output
    iresz     =      p11               ; coefficient to scale amount of resonz in output

; Frequency envelope for reson cutoff
    kfreq     linseg ibegfreq, idur * .5, iendfreq, idur * .5, ibegfreq

; Amplitude envelope to prevent clicking
    kenv      linseg 0, .1, iamp, idur - .2, iamp, .1, 0

; Number of harmonics for gbuzz scaled to avoid aliasing
    iharms    =      (sr*.4)/ifreq

    asig      gbuzz 1, ifreq, iharms, 1, .9, 1      ; "Sawtooth" waveform
    ain       =      kenv * asig                    ; output scaled by amp envelope
    ares      reson ain, kfreq, ibw, 1
    aresr     resonr ain, kfreq, ibw, 1
    aresz     resonz ain, kfreq, ibw, 1

    out       ares * ires + aresr * iresr + aresz * iresz

endin

</CsInstruments>
<CsScore>
```

```

/* Written by Sean Costello */
f1 0 8192 9 1 1 .25 ; cosine table for gbuzz generator

i1 0 10 1 3000 200 100 4000 1 0 0 ; reson output with bw = 200
i1 10 10 1 3000 200 100 4000 0 1 0 ; resonr output with bw = 200
i1 20 10 1 3000 200 100 4000 0 0 1 ; resonz output with bw = 200
i1 30 10 1 3000 50 200 8000 1 0 0 ; reson output with bw = 50
i1 40 10 1 3000 50 200 8000 0 1 0 ; resonr output with bw = 50
i1 50 10 1 3000 50 200 8000 0 0 1 ; resonz output with bw = 50
e

</CsScore>
</CsoundSynthesizer>

```

## Historique Technique

*resonr* et *resonz* ont été décrits à l'origine dans un article de Julius O. Smith et James B. Angell.<sup>1</sup> Smith et Angell recommandait la forme *resonz* (zéros à +1 et -1) quand l'efficacité calculatoire était la préoccupation principale car il y a une multiplication de moins par échantillon, tandis que *resonr* (zéros à + et - la racine carrée du rayon des pôles R) était recommandé pour les situations où l'on voulait un pic central parfait à gain constant.

Ken Steiglitz, dans un article ultérieur<sup>2</sup>, démontra que *resonz* avait un gain constant au pic réel du filtre, à l'opposé de *resonr*, qui affichait un gain constant à la position angulaire des pôles. Steiglitz recommandait aussi *resonz* pour ses encoches dans la courbe du gain plus raides à zéro et à la fréquence de Nyquist. Le livre récent de Steiglitz<sup>3</sup> présente une discussion technique détaillée de *reson* et de *resonz*, tandis que le livre de Dodge et Jerse's<sup>4</sup> illustre les différences dans les courbes de réponse de *reson* et de *resonz*.

## Références

1. Smith, Julius O. et Angell, James B., "A Constant-Gain Resonator Tuned by a Single Coefficient," *Computer Music Journal*, vol. 6, no. 4, pp. 36-39, Hiver 1982.
2. Steiglitz, Ken, "A Note on Constant-Gain Digital Resonators," *Computer Music Journal*, vol. 18, no. 4, pp. 8-10, Hiver 1994.
3. Ken Steiglitz, *A Digital Signal Processing Primer, with Applications to Digital Audio and Computer Music*. Addison-Wesley Publishing Company, Menlo Park, CA, 1996.
4. Dodge, Charles et Jerse, Thomas A., *Computer Music: Synthesis, Composition, and Performance*. New York: Schirmer Books, 1997, 2nde édition, pp. 211-214.

## Voir aussi

*resonz*

## Crédits

Auteur : Sean Costello  
Seattle, Washington  
1999

Nouveau dans la version 3.55 de Csound.

Paramètres de taux audio introduits dans la version 6.02



Octobre 2013.

# resonx

resonx — Emule une série de filtres utilisant l'opcode *reson*.

## Description

*resonx* est équivalent à un filtre constitué de plusieurs couches de filtres *reson* avec les mêmes arguments, connectés en série. L'utilisation d'une série d'un nombre important de filtres permet une pente de coupure plus raide. Ils sont plus rapides que l'équivalent obtenu à partir du même nombre d'instances d'opcodes classiques dans un orchestre Csound, car il n'y aura qu'un cycle d'initialisation et une seule passe de *k* cycles de contrôle à la fois et la boucle audio sera entièrement contenue dans la mémoire cache du processeur.

## Syntaxe

```
ares resonx asig, xcf, xbw [, inumlayer] [, iscl] [, iskip]
```

## Initialisation

*inumlayer* (optional) -- (facultatif) -- nombre d'éléments dans la série de filtre. La valeur par défaut est 4.

*iscl* (facultatif, par défaut 0) -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. (Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc.) Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal d'entrée

*xcf* -- la fréquence centrale du filtre, ou position fréquentielle de la crête de la réponse.

*xbw* -- largeur de bande du filtre (la différence en Hz entre les points haut et bas à mi-puissance).

## Exemples

Voici un exemple de l'opcode *resonx*. Il utilise le fichier *resonx.csd* [examples/resonx.csd].

### Exemple 844. Exemple de l'opcode *resonx*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o resonx.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; unfiltered noise

kenv linseg 0, p3*.5, 1, p3*.5, 0 ;envelope
asig rand 0.7 ;white noise
outs asig*kenv, asig*kenv

endin

instr 2 ; filtered noise

kenv linseg 0, p3*.5, 1, p3*.5, 0 ;envelope
asig rand 0.7
kcf line 300, p3, 2000
afilt resonx asig, kcf, 300, 4
asig balance afilt, asig
outs asig*kenv, asig*kenv

endin
</CsInstruments>
<CsScore>

i 1 0 2
i 2 3 2

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*atonex, tonex*

## Crédits

Auteur : Gabriel Maldonado (adapté par John ffitich)  
Italie

Nouveau dans la version 3.49 de Csound

Paramètres de taux audio introduits dans la version 6.02

Octobre 2013.

# resonxk

resonxk — Pile de filtres à résonance de signal de contrôle.

## Description

*resonxk* est équivalent à un groupe de filtres *resonk*, avec les mêmes arguments, connectés en série. Plus le nombre de filtres est grand et plus raide est la coupure.

## Syntaxe

```
kres resonxk ksig, kcf, kbw[, inumlayer, iscl, istor]
```

## Initialisation

*inumlayer* - nombre d'éléments de la pile de filtres. La valeur par défaut est 4. La valeur maximale est 10.

*iscl* (facultatif, 0 par défaut) -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. (Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc.) Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

*istor* (optional, default=0) -- (facultatif, 0 par défaut) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*kres* - signal de sortie.

*ksig* - signal d'entrée.

*kcf* - la fréquence centrale du filtre, ou position fréquentielle du pic de la réponse.

*kbw* - largeur de bande du filtre (la différence en Hz entre les points haut et bas à mi-puissance).

*resonxk* est bien plus rapide que l'utilisation d'instances individuelles de l'ancien opcode dans un orchestre de Csound, parce que ne sont nécessaires qu'une seule initialisation et qu'un cycle k à la fois, et que la boucle audio est entièrement contenue dans la mémoire cache du processeur.

## Exemples

Voici un exemple de l'opcode *resonxk*. Il utilise le fichier *resonxk.csd* [examples/resonxk.csd].

### Exemple 845. Exemple de l'opcode *resonxk*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o resonxk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisin ftgen 0, 0, 2^10, 10, 1

instr 1

ksig randomh 400, 1800, 150
aout poscil .2, 1000+ksig, gisin
    outs    aout, aout
endin

instr 2

ksig randomh 400, 1800, 150
kcf line 1, p3, 1000 ;vary high-pass
ilay = p4
ksig resonxk ksig, kcf, 100, ilay
aout poscil .2, 1000+ksig, gisin
asig interp ksig ;convert k-rate to a-rate
aout balance asig, aout ;avoid getting asig out of range
    outs    aout, aout
endin

</CsInstruments>
<CsScore>
i 1 0 5
i 2 6 5 1 ;number of filter stack = 1
i 2 12 5 5 ;number of filter stack = 5
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5. (Disponible auparavant seulement dans CsoundAV).

# resony

**resony** — Un banc de filtres passe-bande du second ordre, connectés en parallèle.

## Description

Un banc de filtres passe-bande du second ordre, connectés en parallèle.

## Syntaxe

```
ares resony asig, kbf, kbw, inum, ksep [, isepmode] [, iscl] [, iskip]
```

## Initialisation

*inum* -- nombre de filtres

*isepmode* (facultatif, 0 par défaut) -- si *isepmode* = 0, la séparation des fréquences centrales de chaque filtre est générée logarithmiquement (en utilisant l'octave comme unité de mesure). Si *isepmode* est différent de 0, la séparation des fréquences centrales de chaque filtre est généralement linéaire (en Hz). La valeur par défaut est 0.

*iscl* (facultatif, 0 par défaut) -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc. Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

*iskip* (facultatif, 0 par défaut) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal d'entrée audio

*kbf* -- fréquence de base, c-à-d fréquence centrale en Hz du filtre le plus bas

*kbw* -- largeur de bande en Hz

*ksep* -- séparation de la fréquence centrale des filtres en octaves

**resony** est un banc de filtres passe-bande du second ordre, avec séparation des fréquences, fréquence de base et largeur de bande variables au taux-k, connectés en parallèle (le signal résultant est un mélange de la sortie de chaque filtre). La fréquence centrale de chaque filtre dépend des variables *kbf* et *ksep*. Le nombre maximum de filtres est limité à 100.

## Exemples

Voici un exemple de l'opcode **resony**. Il utilise le fichier *resony.csd* [examples/resony.csd].

### Exemple 846. Exemple de l'opcode resony.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o resonx.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; unfiltered noise

kenv linseg 0, p3*.5, 1, p3*.5, 0 ;envelope
asig rand 0.7 ;white noise
outs asig*kenv, asig*kenv

endin

instr 2 ; filtered noise

ksep = p4 ;vary seperation of center frequency of filters in octaves
kenv linseg 0, p3*.5, 1, p3*.5, 0 ;envelope
asig rand 0.7
kbf line 300, p3, 2000 ;vary base frequency
afilt resony asig, kbf, 300, 4, ksep
asig balance afilt, asig
outs asig*kenv, asig*kenv

endin
</CsInstruments>
<CsScore>

i 1 0 2
i 2 3 2 1
i 2 6 2 3

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1999

Nouveau dans la version 3.56 de Csound.

# resonz

resonz — Un filtre passe-bande avec une réponse en fréquence variable.

## Description

Implémentation d'un filtre passe-bande du second ordre à deux pôles et deux zéros, avec une réponse en fréquence variable.

## Syntaxe

```
ares resonz asig, xcf, xbw [, iscl] [, iskip]
```

## Initialisation

Les variables d'initialisation facultatives de *resonr* et de *resonz* sont identiques aux variables de taux-i de *reson*.

*iscl* -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

*iscl* -- facteur de pondération codé pour les résonateurs. Une valeur de 1 signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de 2 élève le facteur de réponse de façon à ce que sa valeur efficace globale soit égale à 1. Cette égalisation intentionnelle des puissances d'entrée et de sortie suppose que toutes les fréquences sont présentes ; elle est ainsi plus appropriée au bruit blanc. Une valeur de 0 signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

## Exécution

*resonr* et *resonz* sont des variations du résonateur passe-bande classique à deux pôles (*reson*). Ces deux filtres ont deux zéros dans leur fonction de transfert en plus des deux pôles. Les zéros de *resonz* se trouvent à  $z = 1$  et à  $z = -1$ . Les zéros de *resonr* se trouvent à  $+\text{racine\_carrée}(R)$  et à  $-\text{racine\_carrée}(R)$ , où  $R$  est le rayon des pôles dans le plan complexe des  $z$ . L'ajout de zéros à *resonr* et à *resonz* améliore la sélectivité de la magnitude de la réponse de ces filtres aux fréquences de coupure proches de 0, ceci au prix d'une moins grande sélectivité aux fréquences supérieures à la crête de la fréquence de coupure.

*resonr* et *resonz* sont très proches du gain constant lorsque la fréquence centrale glisse, ce qui donne un contrôle plus efficace de la magnitude de la réponse qu'avec les résonateurs à deux pôles traditionnels tels que *reson*.

*resonr* et *resonz* produisent une sonorité considérablement différente de celle de *reson*, spécialement pour les faibles fréquences centrales ; la méthode par tâtonnement est la meilleure façon de déterminer quel résonateur est le plus adapté à une application particulière.

*asig* -- signal d'entrée à filtrer

*xcf* -- fréquence de coupure ou de résonance du filtre, mesurée en Hz



*xbw* -- largeur de bande du filtre (la différence en Hz entre les points à mi-puissance inférieur et supérieur).

## Historique Technique

*resonr* et *resonz* ont été décrits à l'origine dans un article de Julius O. Smith et James B. Angell.<sup>1</sup> Smith et Angell recommandait la forme *resonz* (zéros à +1 et -1) quand l'efficacité calculatoire était la préoccupation principale car il y a une multiplication de moins par échantillon, tandis que *resonr* (zéros à + et - la racine carrée du rayon des pôles R) était recommandé pour les situations où l'on voulait un pic central parfait à gain constant.

Ken Steiglitz, dans un article ultérieur<sup>2</sup>, démontra que *resonz* avait un gain constant au pic réel du filtre, à l'opposé de *resonr*, qui affichait un gain constant à la position angulaire des pôles. Steiglitz recommandait aussi *resonz* pour ses encoches dans la courbe du gain plus raides à zéro et à la fréquence de Nyquist. Le livre récent de Steiglitz<sup>3</sup> présente une discussion technique détaillée de *reson* et de *resonz*, tandis que le livre de Dodge et Jerse's<sup>4</sup> illustre les différences dans les courbes de réponse de *reson* et de *resonz*.

## Références

1. Smith, Julius O. et Angell, James B., "A Constant-Gain Resonator Tuned by a Single Coefficient," *Computer Music Journal*, vol. 6, no. 4, pp. 36-39, Hiver 1982.
2. Steiglitz, Ken, "A Note on Constant-Gain Digital Resonators," *Computer Music Journal*, vol. 18, no. 4, pp. 8-10, Hiver 1994.
3. Ken Steiglitz, *A Digital Signal Processing Primer, with Applications to Digital Audio and Computer Music*. Addison-Wesley Publishing Company, Menlo Park, CA, 1996.
4. Dodge, Charles et Jerse, Thomas A., *Computer Music: Synthesis, Composition, and Performance*. New York: Schirmer Books, 1997, 2nde édition, pp. 211-214.

## Exemples

Voici un exemple des opcode *resonr* et *resonz*. Il utilise le fichier *resonr.csd* [examples/resonr.csd].

### Exemple 847. Exemple des opcode *resonr* et *resonz*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o resonr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Sean Costello */
; Orchestra file for resonant filter sweep of a sawtooth-like waveform.
; The outputs of reson, resonr, and resonz are scaled by coefficients
; specified in the score, so that each filter can be heard on its own
; from the same instrument.

sr = 44100
kr = 4410
```

```

ksmps = 10
nchnls = 1

instr 1

    idur      =      p3
    ibegfreq  =      p4                ; beginning of sweep frequency
    iendfreq  =      p5                ; ending of sweep frequency
    ibw       =      p6                ; bandwidth of filters in Hz
    ifreq     =      p7                ; frequency of gbuzz that is to be filtered
    iamp      =      p8                ; amplitude to scale output by
    ires      =      p9                ; coefficient to scale amount of reson in output
    iresr     =      p10               ; coefficient to scale amount of resonr in output
    iresz     =      p11               ; coefficient to scale amount of resonz in output

; Frequency envelope for reson cutoff
kfreq  linseg ibegfreq, idur * .5, iendfreq, idur * .5, ibegfreq

; Amplitude envelope to prevent clicking
kenv  linseg 0, .1, iamp, idur - .2, iamp, .1, 0

; Number of harmonics for gbuzz scaled to avoid aliasing
iharms =      (sr*.4)/ifreq

asig    gbuzz 1, ifreq, iharms, 1, .9, 1      ; "Sawtooth" waveform
ain     =      kenv * asig                    ; output scaled by amp envelope
ares    reson ain, kfreq, ibw, 1
aresr    resonr ain, kfreq, ibw, 1
aresz    resonz ain, kfreq, ibw, 1

        out ares * ires + aresr * iresr + aresz * iresz

endin

</CsInstruments>
<CsScore>

/* Written by Sean Costello */
f1 0 8192 9 1 1 .25                ; cosine table for gbuzz generator

i1 0 10 1 3000 200 100 4000 1 0 0      ; reson  output with bw = 200
i1 10 10 1 3000 200 100 4000 0 1 0     ; resonr output with bw = 200
i1 20 10 1 3000 200 100 4000 0 0 1     ; resonz output with bw = 200
i1 30 10 1 3000 50 200 8000 1 0 0      ; reson  output with bw = 50
i1 40 10 1 3000 50 200 8000 0 1 0      ; resonr output with bw = 50
i1 50 10 1 3000 50 200 8000 0 0 1      ; resonz output with bw = 50
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*resonr*

## Crédits

Auteur : Sean Costello  
Seattle, Washington  
1999

Nouveau dans la version 3.55 de Csound.

Paramètres de taux audio introduits dans la version 6.02

Octobre 2013.

# resyn

**resyn** — Synthèse additive d'un flot de suivi de partiel avec interpolation cubique de la phase, contrôle de hauteur et modification de l'échelle temporelle de l'entrée.

## Description

L'opcode *resyn* prend en entrée un flot de signal PV TRACKS (tel que généré par l'opcode *partials* par exemple). Il resynthétise le signal avec interpolation linéaire de l'amplitude et interpolation cubique de la phase pour piloter un banc d'oscillateurs interpolants avec pondération de l'amplitude et de la hauteur. *resyn* est une version modifiée *desinsyn*, qui permet la resynthèse de données avec modification de la hauteur et de l'échelle temporelle.

## Syntaxe

```
asig resyn fin, kscal, kpitch, kmaxtracks, ifn
```

## Exécution

*asig* -- signal audio de sortie

*fin* -- flot PV TRACKS d'entrée

*kscal* -- pondération de l'amplitude

*kpitch* -- pondération de la hauteur

*kmaxtracks* -- nombre maximum de pistes dans la resynthèse. En limitant ce dernier, on obtient un effet de filtrage non-linéaire (les pistes sont ordonnées respectivement par date de début et par fréquence ascendante).

*ifn* -- table de fonction contenant une période de fonction sinusoïdale (sinus ou cosinus).

## Exemples

Voici un exemple de l'opcode *resyn*. Il utilise le fichier *resyn.csd* [examples/resyn.csd].

### Exemple 848. Exemple de l'opcode *resyn*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o resyn.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
```

```
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ktracks = p4
ain diskin2 "fox.wav", 1, 0, 1
fs1,fsi2 pvsifd ain, 2048, 512, 1 ; pvsifd analysis
fst partials fs1, fsi2, .1, 1,3, 500 ; partial tracking
aout resyn fst, 1, 1.5, ktracks, 1 ; resynthesis (up a 5th)
outs aout, aout

endin
</CsInstruments>
<CsScore>
;sine
f1 0 4096 10 1

i 1 0 2.7 500
i 1 3 2.7 10 ;non-linear filtering effect

e
</CsScore>
</CsoundSynthesizer>
```

L'exemple ci-dessus montre le suivi de partiel d'un signal d'analyse par distribution de fréquence instantanée et la resynthèse additive avec interpolation cubique de la phase et transposition de hauteur.

## Crédits

Auteur : Victor Lazzarini  
Juin 2005

Nouveau greffon dans la version 5

Novembre 2004.

# return

`return` — Retourne une valeur d'un instrument.

## Description

*return* retourne une valeur d'un instrument durant l'initialisation. La valeur d'un instrument global (instrument 0) peut être retrouvée après la compilation par l'opcode *evalstr*. La récupération de valeurs retournées par d'autres instruments n'est pas encore implémentée.

## Syntaxe

```
return ival
```

## Initialisation

« *ival* » -- une valeur à retourner par un instrument.

## Exemples

Voici un exemple de l'opcode `return` en conjonction avec `evalstr` :

### Exemple 849.

```
ival evalstr "return 2 + 2"
print ival
```

## Crédits

Auteur : Victor Lazzarini, 2013

# reverb

reverb — Réverbère un signal d'entrée avec une réponse en fréquence « de lieu naturel ».

## Description

Réverbère un signal d'entrée avec une réponse en fréquence « de lieu naturel ».

## Syntaxe

```
ares reverb asig, krvt [, iskip]
```

## Initialisation

*iskip* (facultatif, 0 par défaut) -- état initial de l'espace de données de la boucle de retard (cf. *reson*). La valeur par défaut est 0.

## Exécution

*krvt* -- la durée de réverbération (définie comme le temps en secondes pris par un signal pour décroître à 1/1000 ou 60 dB de son amplitude originale).

Une unité *reverb* standard est composée de quatre filtres en peigne *comb* en parallèle suivis de deux unités *alpass* en série. Les durées de boucle sont réglées pour une « réponse de lieu naturel » optimale. Les besoins en mémoire pour cette unité ne sont proportionnels qu'au taux d'échantillonnage, chaque unité ayant besoin d'approximativement 3K mots pour chaque 10 KC. Les unités *comb*, *alpass*, *delay*, *tone* et d'autres unités de Csound permettent d'expérimenter sur des conceptions alternatives de réverbération.

Comme la sortie de la *reverb* standard n'apparaît qu'avec un retard d'environ 1/20 seconde, et souvent avec moins de trois-quarts de la puissance originale, il est normal d'envoyer en sortie à la fois la source et le signal réverbéré. Si *krvt* est fixé par inadvertance à un nombre non positif, il sera automatiquement réinitialisé à 0.01. (Nouveau dans la version 4.07 de Csound.) De plus, comme le son réverbéré persiste longtemps après l'arrêt de la source, il est normal de mettre *reverb* dans un instrument séparé auquel le son est transmis via une *variable globale*, et de laisser cet instrument actif durant toute l'exécution.

## Exemples

Voici un exemple de l'opcode *reverb*. Il utilise le fichier *reverb.csd* [examples/reverb.csd].

### Exemple 850. Exemple de l'opcode reverb.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o reverb.wav -W ;; for file output any platform
```

```
</CsOptions>
<CsInstruments>

  sr = 44100
  ksmps = 32
  0dbfs = 1
  nchnls = 2

  gal init 0

  instr 1

  asig poscil .2, cpspch(p4), 1
    outs asig, asig

  gal += asig      ;add direct signal to global reverb

  endin

  instr 99 ;(highest instr number executed last)

  arev reverb gal, 1.5
    outs arev, arev

  gal = 0 ;clear
  endin

</CsInstruments>
<CsScore>
f 1 0 128 10 1 ;sine

i 1 0 0.1 7.00 ;short sounds
i 1 1 0.1 8.02
i 1 2 0.1 8.04
i 1 3 0.1 8.06

i 99 0 6 ;reverb runs for 6 seconds
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*alpass, comb, valpass, vcomb*

## Crédits

Auteur : William « Pete » Moss  
Université du Texas à Austin  
Austin, Texas USA  
Janvier 2002



# reverb2

reverb2 — Identique à l'opcode nreverb.

## Description

Identique à l'opcode *nreverb*.

## Syntaxe

```
ares reverb2 asig, ktime, khdif [, iskip] [,inumCombs] \  
    [, ifnCombs] [, inumAlpas] [, ifnAlpas]
```

# reverbsc

reverbsc — Réverbération FDN stéréo à 8 lignes à retard, basée sur un travail de Sean Costello.

## Description

Réverbération FDN (Feedback Delay Network) stéréo à 8 lignes à retard, avec matrice de rétroaction, basée sur un modèle physique de jonctions dissipatives de 8 guides d'onde sans perte d'impédance caractéristique égale. Basée sur la version orchestre de Csound de Sean Costello.

## Syntaxe

```
aoutL, aoutR reverbsc ainL, ainR, kfbvl, kfco[, israte[, ipitchm[, iskip]]]
```

## Initialisation

*israte* (facultatif, taux d'échantillonnage de l'orchestre par défaut) -- on suppose un taux d'échantillonnage de *israte*. Il est habituellement fixé à *sr*, mais un réglage différent peut être utile pour des effets spéciaux.

*ipitchm* (facultatif, 1 par défaut) -- amplitude des variations aléatoires ajoutées aux retards, comprise entre 0 et 10. La valeur par défaut est 1, mais elle peut être trop importante et nécessiter une réduction pour les hauteurs tenues telles que les notes de piano.

*iskip* (facultatif, 0 par défaut) -- s'il est différent de zéro, l'initialisation de l'opcode est ignorée, si c'est possible.

## Exécution

*aoutL, aoutR* -- signaux de sortie pour les canaux gauche et droite.

*ainL, ainR* -- canaux d'entrée gauche et droite. Noter que même si l'on n'a un signal d'entrée que sur un des deux canaux, on aura quand même une sortie réverbérée sur deux canaux, ce qui rend cette unité plus adaptée à la réverbération d'une entrée stéréo que l'opcode *freeverb*.

*kfbvl* -- niveau de rétroaction, compris entre 0 et 1. 0.6 donne un bon son de petit lieu "vivant", 0.8 un petit hall et 0.9 un grand hall. 1 signifie une longueur infinie, tandis que les valeurs supérieures rendront l'opcode instable.

*kfco* -- fréquence de coupure des filtres passe-bas du premier ordre dans la boucle de rétroaction des lignes à retard, en Hz. Doit être comprise entre 0 et  $israte/2$  (pas  $sr/2$ ). Moins la valeur est importante et plus la décroissance des hautes fréquences est rapide.

## Exemples

Voici un exemple de l'opcode *reverbsc*. Il utilise le fichier *reverbsc.csd* [exemples/reverbsc.csd].

### Exemple 851. Un exemple de l'opcode *reverbsc*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o reverbosc.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr          = 48000
ksmps      = 32
nchnls     = 2
0dbfs      = 1

instr 1
a1          vco2 0.85, 440, 10
kfrq        port 100, 0.004, 20000
a1          butterlp a1, kfrq
a2          linseg 0, 0.003, 1, 0.01, 0.7, 0.005, 0, 1, 0
a1          = a1 * a2
a2          = a1 * p5
a1          = a1 * p4
denorm      a1, a2
aL, aR      reverbosc a1, a2, 0.85, 12000, sr, 0.5, 1
outs        a1 + aL, a2 + aR
endin

</CsInstruments>
<CsScore>
i 1 0 1 0.71 0.71
i 1 1 1 0 1
i 1 2 1 -0.71 0.71
i 1 3 1 1 0
i 1 4 4 0.71 0.71
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Istvan Varga  
2005

# rewindscore

rewindscore — Recule la position de lecture de l'exécution courante de la partition.

## Description

Recule la position de lecture de l'exécution courante de la partition.

## Syntaxe

`rewindscore`

## Exemples

Voici un exemple de l'opcode `rewindscore`. Il utilise le fichier `rewindscore.csd` [examples/rewindscore.csd].

### Exemple 852. Exemple de l'opcode `rewindscore`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o rewindscore.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kenv expon 1, p3, 0.0001
aout poscil .5*kenv, cpspch(p4), 1
outs aout, aout
endin

instr 100

rewindscore

endin
</CsInstruments>
<CsScore>
f1 0 4096 10 1 ;sine wave

i1 0 1 8.00
i1 + 1 8.03
i1 + 1 8.04
i1 + 1 8.07
```

```
i1 + 1 8.09
i1 + 1 8.10
i1 + 1 8.09
i1 + 1 8.07
i1 + 1 8.03

i100 9 1 ;rewind from 9th second
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*setscorepos*

## Crédits

Auteur : Victor Lazzarini  
2008

Nouveau dans la version 5.09 de Csound.

# rezzy

rezzy — Un filtre passe-bas à résonance.

## Description

Un filtre passe-bas à résonance.

## Syntaxe

```
ares rezzy asig, xfco, xres [, imode, iskip]
```

## Initialisation

*imode* (facultatif, 0 par défaut) -- mode passe-haut ou passe-bas. S'il vaut zéro, *rezzy* est passe-bas. S'il est différent de zéro, *rezzy* est passe-haut. La valeur par défaut est 0. (Nouveau dans la version 3.50 de Csound.)

*iskip* (facultatif, 0 par défaut) -- s'il est différent de zéro, l'initialisation du filtre est ignorée. (Nouveau dans les versions 4.23f13 et 5.0 de Csound.)

## Exécution

*asig* -- signal d'entrée

*xfco* -- fréquence de coupure du filtre en Hz. Depuis la version 3.50, peut-être de taux-i, de taux-k ou de taux-a.

*xres* -- quantité de résonance. Des valeurs entre 1 et 100 sont typiques. La résonance doit valoir un ou plus. Depuis la version 3.50, peut-être de taux-i, de taux-k ou de taux-a.

*rezzy* est un filtre passe-bas à résonance créé empiriquement par Hans Mikelson.

## Exemples

Voici un exemple de l'opcode *rezzy*. Il utilise le fichier *rezzy.csd* [examples/rezzy.csd].

### Exemple 853. Exemple de l'opcode *rezzy*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o rezzy.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

sr = 44100
ksmps = 32
Odbfs = 1
nchnls = 2

instr 1

asaw vco2 .3, 110 ;sawtooth
kcf line 1760, p3, 220 ;vary cut-off frequency from 220 to 1280 Hz
kres = p4 ;vary resonance too
ares rezzy asaw, kcf, kres
asig balance ares, asaw
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 4 10
i 1 + 4 30
i 1 + 4 120 ;lots of resonance
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*biquad, moogvcf*

## Crédits

Auteur : Hans Mikelson  
 Octobre 1998

Nouveau dans la version 3.49 de Csound.

# rfft

rfft — Transformée de Fourier rapide d'un tableau de valeurs réelles.

## Description

Applique une transformée de Fourier rapide directe à un tableau unidimensionnel de valeurs réelles. La sortie est un autre tableau contenant la transformée, non redondante, seulement le spectre non négatif. Si le tableau d'entrée a pour taille une puissance de deux, la taille du tableau de sortie sera égale à celle du tableau d'entrée, les deux premiers points contenant les coefficients de 0 Hz et de la fréquence de Nyquist. Sinon, la sortie aura deux valeurs supplémentaires (taille d'entrée + 2) et le coefficient de Nyquist sera placé en *kin*[taille d'entrée]. Les positions *kin*[1] et *kin*[taille d'entrée+1] seront nulles.

## Syntaxe

```
kout[] rfft kin[]
```

## Exécution

*kout*[] -- tableau de sortie contenant la transformée. Créé s'il n'existe pas.

*kin*[] -- tableau contenant les valeurs réelles d'entrée.

## Exemples

Voici un exemple de l'opcode rfft. Il utilise le fichier *rfft.csd* [examples/rfft.csd].

### Exemple 854. Exemple de l'opcode rfft.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>

<CsOptions>
-d -o dac
</CsOptions>

<CsInstruments>
;ksmps needs to be an integer div of hopsize
ksmps = 64
0dbfs=1

instr 1

  ihopsize = 256 ; hopsize
  ifftsize = 1024 ; FFT size
  iolaps = ifftsize/ihopsize ; overlaps
  ibw = sr/ifftsize ; bin bandwidth
  kcmt init 0 ; counting vars
  krow init 0

  kOla[] init ifftsize ; overlap-add buffer
  kIn[] init ifftsize ; input buffer
  kSw[] init ifftsize
```



```

kOut[][] init iolaps, ifftsize ; output buffers

a1 diskin2 "fox.wav",1,0,1 ; audio input
ks expon 100, p3, 1000
asw vco2 0.15, ks

/* every hopsize samples */
if kcmt == ihopsize then
  /* window and take FFT */
  kWin[] window kIn,krow*ihopsize
  kSpec[] rfft kWin
  kWin window kSw,krow*ihopsize
  kSpec2[] rfft kWin
  kProd[] cmplxprod kSpec, kSpec2

  /* IFFT + window */
  kRow[] rifft kProd + kSpec
  kWin window kRow, krow*ihopsize
  /* place it on out buffer */
  kOut setrow kWin, krow

  /* zero the ola buffer */
  kOla = 0
  /* overlap-add */
  ki = 0
  until ki == iolaps do
    kRow getrow kOut, ki
    kOla = kOla + kRow
    ki += 1
  od

  /* update counters */
  krow = (krow+1)%iolaps
  kcmt = 0
endif

/* shift audio in/out of buffers */
kIn shiftin a1
kSw shiftin asw
a2 shiftout kOla
  out a2/iolaps

/* increment counter */
kcmt += ksmpps

endin

</CsInstruments>

<CsScore>
i1 0 10
</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux.

## Crédits

Auteur : Victor Lazzarini  
 NUI Maynooth  
 2014

Nouveau dans la version 6.04

# rifft

rifft — Transformée de Fourier rapide inverse complexe vers réel.

## Description

Applique une transformée de Fourier rapide inverse à un tableau unidimensionnel de valeurs complexes produisant une sortie réelle. Cette sortie est un autre tableau contenant les valeurs réelles du signal. Si le tableau d'entrée a pour taille une puissance de deux, la taille du tableau de sortie sera égale à celle du tableau d'entrée. Sinon, la sortie aura deux valeurs de moins (taille d'entrée - 2).

## Syntaxe

```
kout[] rifft kin[]
```

## Exécution

*kout[]* -- tableau contenant les valeurs réelles de sortie. Créé s'il n'existe pas.

*kin[]* -- tableau contenant les valeurs complexes d'entrée.

## Exemples

Voici un exemple de l'opcode rifft. Il utilise le fichier *irfft.csd* [examples/irfft.csd].

### Exemple 855. Exemple de l'opcode rifft.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>

<CsOptions>
-d -o dac
</CsOptions>

<CsInstruments>
;ksmps needs to be an integer div of hopsize
ksmps = 64

instr 1

  ihopsize = 256    ; hopsize
  ifftsize = 1024   ; FFT size
  iolaps = ifftsize/ihopsize ; overlaps
  ibw = sr/ifftsize ; bin bandwidth
  kcmt init 0      ; counting vars
  krow init 0

  kOla[] init ifftsize ; overlap-add buffer
  kIn[] init ifftsize ; input buffer
  kOut[][] init iolaps, ifftsize ; output buffers

  a1 diskin2 "fox.wav",1,0,1 ; audio input

  /* every hopsize samples */
```

```

if kcnt == ihopsize then
  /* window and take FFT */
  kWin[] window kIn,krow*ihopsize
  kSpec[] rfft kWin

  /* filter between high and low freqs */
  ilow = 0
  ihigh = 1000
  ki = int(ilow/ibw)
  until ki == int(ihigh/ibw) do
    kSpec[ki] = 0
    ki += 1
  od

  /* IFFT + window */
  kRow[] rfft kSpec
  kWin window kRow, krow*ihopsize
  /* place it on out buffer */
  kOut setrow kWin, krow

  /* zero the ola buffer */
  kOla = 0
  /* overlap-add */
  ki = 0
  until ki == iolaps do
    kRow getrow kOut, ki
    kOla = kOla + kRow
    ki += 1
  od

  /* update counters */
  krow = (krow+1)%iolaps
  kcnt = 0
endif

/* shift audio in/out of buffers */
kIn shiftin a1
a2 shiftout kOla
  out a2/iolaps

/* increment counter */
kcnt += ksmpls

endin

</CsInstruments>

<CsScore>
i1 0 10
</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux.

## Crédits

Auteur : Victor Lazzarini  
 NUI Maynooth  
 2014

Nouveau dans la version 6.04

# rigoto

rigoto — Transfère le contrôle durant une phase de réinitialisation.

## Description

Semblable à *igoto*, mais n'agit que dans une phase de *réinitialisation* (*reinit*) (c'est-à-dire qu'il n'opère pas pendant l'initialisation standard). Cette instruction est utile pour ignorer les unités qui ne doivent pas être réinitialisées.

## Syntaxe

```
rigoto label
```

## Voir aussi

*cigoto*, *igoto*, *reinit*, *rireturn*

# rireturn

rireturn — Termine une phase de réinitialisation.

## Description

Termine une phase de *réinitialisation* (*reinit*) (c'est-à-dire qu'il n'opère pas pendant l'initialisation standard). Cette instruction, ou un *endin*, provoquera la reprise de l'exécution normale.

## Syntaxe

`rireturn`

## Exemples

Les instructions suivantes génèrent un signal de contrôle exponentiel dont les valeurs vont de 440 à 880 exactement dix fois pendant la durée *p3*. Elles utilisent le fichier *reinit.csd* [exemples/reinit.csd].

### Exemple 856. Exemple de l'opcode rireturn.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o reinit.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

instr 1

reset:
    timeout 0, p3/10, contin
    reinit reset

contin:
    kLine expon 440, p3/10, 880
    aSig oscil 10000, kLine, 1
    out aSig
    rireturn

endin

</CsInstruments>
<CsScore>

f1 0 4096 10 1
```

```
i1 0 10  
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*reinit, rigoto*

# rms

rms — Détermine la valeur efficace d'un signal audio.

## Description

Détermine la valeur efficace d'un signal audio. La valeur instantanée passe à travers un filtre passe-bas pour en sortir une valeur moyenne comme dans un VU-mètre.

## Syntaxe

```
kres rms asig [, ihp] [, iskip]
```

## Initialisation

*ihp* (facultatif, 10 par défaut) -- point à mi-puissance (en Hz) d'un d'un filtre passe-bas interne spécial. La valeur par défaut est 10.

*iskip* (facultatif, 0 par défaut) -- disposition initiale de l'espace de données interne (voir *reson*). La valeur par défaut est 0.

## Exécution

*asig* -- signal audio en entrée

*kres* -- valeur efficace du signal d'entrée issue du filtre passe-bas

Les valeurs de sortie *kres* de *rms* suivent la valeur efficace de l'entrée audio *asig*. Cette unité n'est pas un modificateur de signal, mais fonctionne plutôt comme une mesure de la puissance du signal. Elle utilise un filtre passe-bas interne pour rendre la réponse plus lisse. On peut utiliser *ihp* pour contrôler ce lissage. Plus les valeurs sont importantes, plus la mesure est "dynamique".

On peut aussi utiliser cet opcode comme suiveur d'enveloppe.

La sortie *kres* de cet opcode est donnée en amplitude et dépend de *Odbfs*. Pour une sortie en décibels, il faut utiliser *dbamp*

## Exemples

```
arms rms    asig ; get rms value of signal asig
```

Voici un exemple de l'opcode rms. Il utilise le fichier *rms.csd* [examples/rms.csd].

### Exemple 857. Exemple de l'opcode rms.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
```



```

<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc       -d  -m0    ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rms.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 128
nchnls = 1

;Example by Andres Cabrera 2007

0dbfs = 1

FLpanel "rms", 400, 100, 50, 50
gkrms text, gihrms text FLtext "Rms", -100, 0, 0.1, 3, 110, 30, 60, 50
gkihp, gihandle FLtext "ihp", 0, 10, 0.05, 1, 100, 30, 220, 50
gkrms slider, gihrms slider FLslider "", -60, -0.5, -1, 5, -1, 380, 20, 10, 10

FLpanelEnd
FLrun

FLsetVal_i 5, gihandle
; Instrument #1.
instr 1
a1 inch 1

label:
kval rms a1, i(gkihp) ;measures rms of input channel 1
rreturn

kval = dbamp(kval) ; convert to db full scale
printk 0.5, kval
FLsetVal 1, kval, gihrms slider ;update the slider and text values
FLsetVal 1, kval, gihrms text
knewihp changed gkihp ; reinit when ihp text has changed
if (knewihp == 1) then
reinit label ;needed because ihp is an i-rate parameter
endif
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one minute
i 1 0 60
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*balance, gain*

# rnd

rnd — Retourne un nombre aléatoire dans un intervalle unipolaire au taux de l'argument.

## Description

Retourne un nombre aléatoire dans un intervalle unipolaire au taux de l'argument.

## Syntaxe

`rnd(x)` (taux-i ou -k seulement)

Où l'argument entre parenthèses peut être une expression. Ces convertisseurs de valeur échantillonnent une séquence aléatoire globale, mais sans référencer une *racine*. Le résultat peut devenir un terme d'une expression ultérieure.

## Exécution

Retourne un nombre aléatoire dans l'intervalle unipolaire allant de 0 à *x*.

## Exemples

Voici un exemple de l'opcode rnd. Il utilise le fichier *rnd.csd* [examples/rnd.csd].

### Exemple 858. Exemple de l'opcode rnd.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rnd.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Andres Cabrera 2010

sr = 44100
ksmps = 4410
nchnls = 1
0dbfs = 1

instr 1
; Generate a random number from 0 to 10.
irand = rnd(10)
print irand
endin

instr 2
klimit init 10
krand = rnd(klimit)
```

```

    printk 0, krand
endin

</CsInstruments>
<CsScore>

i 1 0 1 ; Generate 1 number
i 1 0 1 ; Generate another number
i 1 0 1 ; yet another number

i 2 2 1 ; 1 second prints 9 values (kr = 10)
e

</CsScore>
</CsoundSynthesizer>

```

La sortie contiendra des lignes comme celles-ci :

```

SECTION 1:
new alloc for instr 1:
instr 1:  irand = 9.735
new alloc for instr 1:
instr 1:  irand = 1.394
new alloc for instr 1:
instr 1:  irand = 7.695
midi channel 1 now using instr 1
B 0.000 .. 2.000 T 2.000 TT 2.000 M: 0.00000
new alloc for instr 2:
i 2 time 2.10000: 5.25005
i 2 time 2.20000: 6.22665
i 2 time 2.30000: 9.69511
i 2 time 2.40000: 7.16822
i 2 time 2.50000: 9.45134
i 2 time 2.60000: 1.34123
i 2 time 2.70000: 2.09879
i 2 time 2.80000: 2.36001
i 2 time 2.90000: 0.03553

```

## Voir aussi

*birnd*

## Crédits

Auteur: Barry L. Vercoe  
 MIT  
 Cambridge, Massachussetts  
 1997

# rnd31

rnd31 — Opcodes aléatoires bipolaires sur 31 bit avec une distribution contrôlée.

## Description

Opcodes aléatoires bipolaires sur 31 bit avec une distribution contrôlée. Ces unités sont portables, c-à-d qu'avec la même valeur de graine on obtiendra la même séquence aléatoire sur tous les systèmes. La distribution des nombres aléatoires générés peut être changée au taux-k.

## Syntaxe

ax **rnd31** ksc1, krpow [, iseed]

ix **rnd31** iscl, irpow [, iseed]

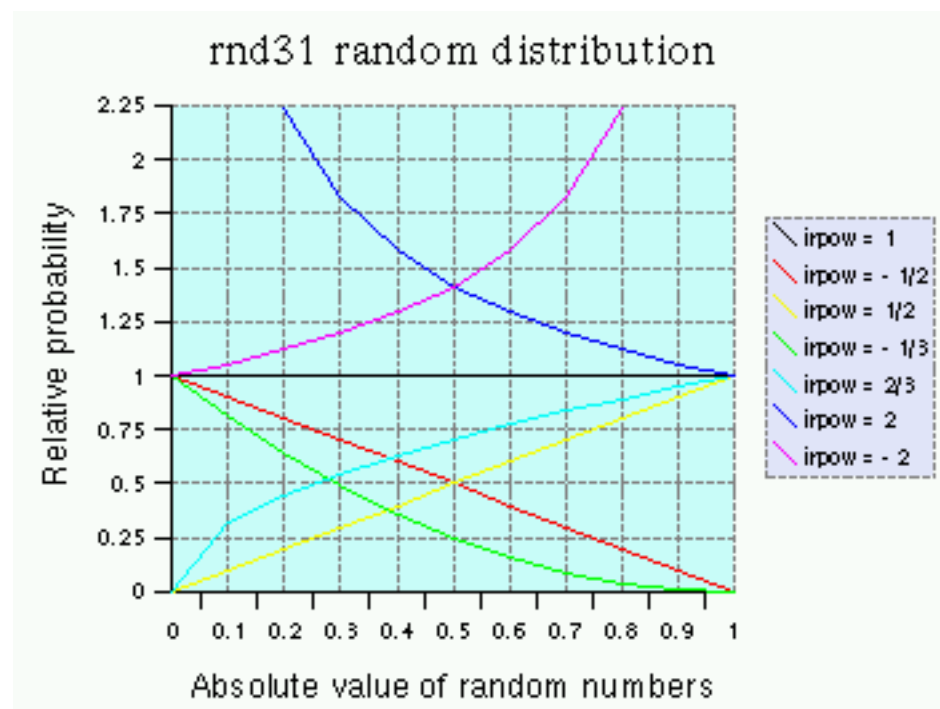
kx **rnd31** ksc1, krpow [, iseed]

## Initialisation

*ix* -- valeur de sortie au taux-i.

*iscl* -- mise à l'échelle de la sortie. Les nombres aléatoires générés sont compris entre *-iscl* et *iscl*.

*irpow* -- contrôle la distribution des nombres aléatoires. Si *irpow* est positif, la distribution aléatoire (*x* compris entre -1 et 1) est  $abs(x)^{(1/irpow)-1}$ ; pour des valeurs négatives de *irpow*, elle vaut  $(1 - abs(x))^{(-1/irpow)-1}$ . En fixant *irpow* à -1, 0 ou 1 on obtiendra une distribution uniforme (c'est aussi plus rapide à calculer).



Un graphique des distributions pour différentes valeurs de *irpow*.

*iseed* (facultatif, par défaut=0) -- valeur de la graine pour le générateur de nombres aléatoires (nombre entier positif compris entre 1 et 2147483646 ( $2^{31} - 2$ )). Avec une valeur nulle ou négative la graine est prise à partir de l'horloge du système (c'est le comportement par défaut). Une graine à partir de l'horloge du système nous garantit la génération de séquences aléatoires différentes, même si plusieurs opcodes aléatoires sont appelés dans un temps très court.

Dans les versions de *taux-a* et de *taux-k* la graine est fixée à l'initialisation de l'opcode. Avec une sortie de *taux-i*, si la graine est nulle ou négative, elle sera prise à partir de l'horloge du système lors du premier appel, puis retournera la valeur suivante de la séquence aléatoire lors des appels successifs ; les valeurs positives de la graine sont fixées à tous les appels de *taux-i*. La graine est locale pour les unités de *taux-a* et *-k*, et globale pour les unités de *taux-i*.



## Notes

- bien que des valeurs de graines allant jusqu'à 2147483646 soient permises, il est recommandé d'utiliser des nombres plus petits ( $< 1000000$ ) pour des raisons de portabilité, car les grands nombres peuvent être arrondis à une valeur différente si l'on utilise des nombres flottants sur 32 bit.
- *rnd31* au *taux-i* avec une graine positive produira toujours la même valeur en sortie (ce n'est pas un bogue). Pour obtenir des valeurs différentes, fixer la graine à 0 dans les appels successifs, ce qui retournera la valeur suivante de la séquence aléatoire.

## Exécution

*ax* -- valeur de sortie au *taux-a*.

*kx* -- valeur de sortie au *taux-k*.

*kscl* -- mise à l'échelle de la sortie. Les nombres aléatoires générés sont compris entre *-kscl* et *kscl*. Semblable à *iscl*, mais il peut être modifié au *taux-k*.

*krpow* -- contrôle la distribution des nombres aléatoires. Semblable à *irpow*, mais il peut être modifié au *taux-k*.

## Exemples

Voici un exemple de l'opcode *rnd31*. Il utilise le fichier *rnd31.csd* [examples/rnd31.csd].

### Exemple 859. Exemple de l'opcode *rnd31* au *taux-a*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rnd31.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create random numbers at a-rate in the range -2 to 2 with
; a triangular distribution, seed from the current time.
a31 rnd31 2, -0.5

; Use the random numbers to choose a frequency.
afreq = a31 * 500 + 100

a1 oscil 30000, afreq, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Voici un exemple de l'opcode `rnd31` au taux-k. Il utilise le fichier `rnd31_krate.csd` [exemples/`rnd31_krate.csd`].

### Exemple 860. Exemple de l'opcode `rnd31` au taux-k.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac            -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o rnd31_krate.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create random numbers at k-rate in the range -1 to 1
; with a uniform distribution, seed=10.
k1 rnd31 1, 0, 10

printks "k1=%f\\n", 0.1, k1
endin

```

```

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

```

```

</CsScore>
</CsoundSynthesizer>

```

La sortie contiendra des lignes comme celles-ci :

```

k1= 0.112106
k1=-0.274665
k1= 0.403933

```

Here is an example of the `rnd31` opcode that uses the number 7 as a seed value. It uses the file `rnd31_seed7.csd` [examples/rnd31\_seed7.csd].

**Exemple 861.** An example of the `rnd31` opcode that uses the number 7 as a seed value.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rnd31_seed7.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; i-rate random numbers with linear distribution, seed=7.
; (Note that the seed was used only in the first call.)
i1 rnd31 1, 0.5, 7
i2 rnd31 1, 0.5
i3 rnd31 1, 0.5

print i1
print i2
print i3
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```
instr 1: i1 = -0.649
instr 1: i2 = -0.761
instr 1: i3 = 0.677
```

Voici un exemple de l'opcode `rnd31` qui utilise l'horloge du système comme graine. Il utilise le fichier `rnd31_time.csd` [exemples/rnd31\_time.csd].

### Exemple 862. Exemple de l'opcode `rnd31` qui utilise l'horloge du système comme graine.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o rnd31_time.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; i-rate random numbers with linear distribution,
; seeding from the current time. (Note that the seed
; was used only in the first call.)
i1 rnd31 1, 0.5, 0
i2 rnd31 1, 0.5
i3 rnd31 1, 0.5

print i1
print i2
print i3
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
instr 1: i1 = -0.691
instr 1: i2 = -0.686
instr 1: i3 = -0.358
```



## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.16

# round

**round** — Retourne la valeur entière la plus proche de  $x$  ; si la partie décimale de  $x$  vaut exactement 0.5, la direction de l'arrondi est indéfinie.

## Description

La valeur entière la plus proche de  $x$  ; si la partie décimale de  $x$  vaut exactement 0.5, la direction de l'arrondi est indéfinie.

## Syntaxe

**round**( $x$ ) (des arguments de `taux-i`, `-k` ou `-a` sont permis)

**round**( $k/i[]$ ) (`k-` ou `i-tableau`)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur réalisent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode `round`. Il utilise le fichier `round.csd` [exemples/round.csd].

### Exemple 863. Exemple de l'opcode `round`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
;-o round.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

idiv init 1

loop:
inumber = 9
il = inumber / idiv
iro = round(il)
print inumber, idiv, iro ;print number / idiv = result using round
idiv = idiv + 1
if (idiv <= 10) igoto loop

endin
```

```
</CsInstruments>
<CsScore>

i 1 0 0
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
instr 1: inumber = 9.000 idiv = 1.000 ifl = 9.000
instr 1: inumber = 9.000 idiv = 2.000 ifl = 5.000
instr 1: inumber = 9.000 idiv = 3.000 ifl = 3.000
instr 1: inumber = 9.000 idiv = 4.000 ifl = 2.000
instr 1: inumber = 9.000 idiv = 5.000 ifl = 2.000
instr 1: inumber = 9.000 idiv = 6.000 ifl = 2.000
instr 1: inumber = 9.000 idiv = 7.000 ifl = 1.000
instr 1: inumber = 9.000 idiv = 8.000 ifl = 1.000
instr 1: inumber = 9.000 idiv = 9.000 ifl = 1.000
instr 1: inumber = 9.000 idiv = 10.000 ifl = 1.000
```

## Voir aussi

*abs, exp, int, log, log10, i, sqrt*

## Crédits

Auteur : Istvan Varga  
Nouveau dans Csound 5  
2005

# rspline

rspline — Génère des courbes splines aléatoires.

## Description

Génère des courbes splines aléatoires.

## Syntaxe

```
ares rspline xrangeMin, xrangeMax, kcpsMin, kcpsMax
```

```
kres rspline krangeMin, krangeMax, kcpsMin, kcpsMax
```

## Exécution

*kres, ares* -- Signal de sortie.

*xrangeMin, xrangeMax* -- Intervalle des valeurs des points générés aléatoirement.

*kcpsMin, kcpsMax* -- Intervalle de définition du taux de génération des points. Les limites minimale et maximale sont exprimées en Hz.

*rspline* (générateur de courbe spline aléatoire) est semblable à *jspline* mais l'intervalle de sortie est défini par deux valeurs limites. De plus, ici, l'intervalle de sortie réel pourra légèrement dépasser les valeurs données à cause des courbes d'interpolation entre chaque paire de points aléatoires.

Actuellement les courbes générées sont assez lisses quand *cspMin* n'est pas trop différent de *cpsMax*. Quand l'intervalle *cpsMin-cpsMax* est grand, quelques petites discontinuités peuvent se produire, mais, dans la plupart des cas, cela ne devrait pas poser de problème. L'algorithme sera peut-être amélioré dans les prochaines versions.

Ces opcodes sont souvent meilleurs que *jitter* lorsque l'on veut un rendu « naturel » ou « analogique » de sons numériques. On peut aussi les utiliser dans la composition algorithmique, pour générer des lignes mélodiques aléatoires lisses lors d'une utilisation conjointe avec l'opcode *samphold*.

Noter que le résultat est assez différent de celui que l'on obtiendrait en filtrant un bruit blanc, et que l'on peut ainsi obtenir un contrôle bien plus précis.

## Exemples

Voici un exemple de l'opcode *rspline*. Il utilise le fichier *rspline.csd* [exemples/rspline.csd].

### Exemple 864. Exemple de l'opcode *rspline*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;;realtime audio out
```

```

;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o jspline.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

krangeMin init p4
krangeMax init p5
kcpsmin init 2
kcpsmax init 3

ksp rspline krangeMin, krangeMax, kcpsmin, kcpsmax
aout pluck 1, 200+ksp, 1000, 0, 1
aout dcblock aout ;remove DC
outs aout, aout

endin
</CsInstruments>
<CsScore>

i 1 0 10 2 5 ;a bit jitter
i 1 8 10 10 20 ;some more
i 1 16 10 20 50 ;lots more
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la Version 4.15

# rtclock

rtclock — Lit l'horloge temps réel du système d'exploitation.

## Description

Lit l'horloge temps réel du système d'exploitation.

## Syntaxe

```
ires rtclock
```

```
kres rtclock
```

## Exécution

Lit l'horloge temps réel du système d'exploitation. Sous Windows, celle-ci ne change qu'une fois par seconde. Sous GNU/Linux, elle change chaque microseconde. Le comportement sous les autres systèmes varie.

## Exemples

Voici un exemple de l'opcode rtclock. Il utilise le fichier *rtclock.csd* [examples/rtclock.csd].

### Exemple 865. Exemple de l'opcode rtclock.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n          ;;no sound
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o rtclock.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

;after an example by Iain McCurdy

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

FLcolor 200, 200, 200, 0, 0, 0
; LABEL      | WIDTH | HEIGHT | X | Y
FLpanel "rtclock", 500, 130, 0, 0
;
; ON,OFF,TYPE,WIDTH, HEIGHT, X, Y, OPCODE, INS,START,IDUR
gkOnOff,ihOnOff FLbutton "On/Off", 1, 0, 22, 150, 25, 5, 5, 0, 1, 0, 3600
gkExit,ihExit FLbutton "exitnow",1, 0, 21, 150, 25, 345, 5, 0, 999, 0, 0.001
FLsetColor2 255, 0, 50, ihOnOff ;reddish color

;VALUE DISPLAY BOXES WIDTH,HEIGHT,X, Y
gidclock FLvalue "clock", 100, 25, 200, 60
```

```

FLsetVal_i 1, ihOnOff
FLpanel_end
FLrun

instr 1

if gkOnOff !=0 kgoto CONTINUE ;sense if FLTK on/off switch is not off (in which case skip the next line
turnoff          ;turn this instr. off now
CONTINUE:
ktime rtclock          ;clock continues to run even
FLprintk2 ktime, gidclock ;after the on/off button was used to stop

endin

instr 999

exitnow          ;exit Csound as fast as possible

endin
</CsInstruments>
<CsScore>

f 0 60 ;runs 60 seconds
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch

Nouveau dans la version 4.10

# S

S — Retourne l'équivalent de type-S d'un argument de taux-i ou de taux-k.

## Description

Retourne une chaîne de caractères contenant la valeur numérique de son argument.

## Syntaxe

`s(x)` (control-rate or init-rate arg)

## Voir aussi

*a, k, abs, exp, frac, int, log, log10, sqrt*

## Crédits

Victor Lazzarini, 2016. Nouveau dans la version 6.07



# s16b14

s16b14 — Crée un banc de 16 numéros différents de messages de contrôle MIDI sur 14-bit.

## Description

Crée un banc de 16 numéros différents de messages de contrôle MIDI sur 14-bit.

## Syntaxe

```
i1,...,i16 s16b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \  
    initvalue1, ifn1,..., ictlno_msb16, ictlno_lsb16, imin16, imax16, initvalue16, ifn16  
  
k1,...,k16 s16b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \  
    initvalue1, ifn1,..., ictlno_msb16, ictlno_lsb16, imin16, imax16, initvalue16, ifn16
```

## Initialisation

*i1 ... i64* -- valeurs de sortie

*ichan* -- canal MIDI (1-16)

*ictlno\_msb1 .... ictlno\_msb32* -- numéro de contrôle MIDI, octet de poids fort (0-127)

*ictlno\_lsb1 .... ictlno\_lsb32* -- numéro de contrôle MIDI, octet de poids faible (0-127)

*imin1 ... imin64* -- valeurs minimales pour chaque contrôleur

*imax1 ... imax64* -- valeurs maximales pour chaque contrôleur

*init1 ... init64* -- valeur initiale pour chaque contrôleur

*ifn1 ... ifn64* -- table de fonction de conversion pour chaque contrôleur

## Exécution

*k1 ... k64* -- valeurs de sortie

*s16b14* est un banc de contrôleurs MIDI, utile lorsque l'on utilise un mélangeur MIDI comme le Kawai MM-16 ou autres pour changer n'importe quel paramètre du son en . Les messages de contrôle MIDI arrivant sur le port d'entrée sont convertis pour entrer dans l'intervalle *iminN*, *imaxN*, et une valeur initiale peut être fixée. On peut aussi utiliser de manière facultative une table de fonction non interpolée avec une courbe de traduction personnalisée pour obtenir, par exemple, des courbes de réponse exponentielles.

Si l'on n'a pas besoin d'une table de traduction, on fixe la valeur de *ifnN* à 0, sinon, on donne à *ifnN* un numéro de table de fonction valide. Lorsque l'on utilise une table de traduction (si *ifnN* reçoit une valeur non nulle faisant référence à une table de fonction déjà allouée), la valeur de *initN* doit être égale à celle de *iminN* ou à celle de *imaxN*, sinon la valeur de sortie initiale sera différente de celle spécifiée dans l'argument *initN*.

*s16b14* fournit un banc de 16 numéros différents de messages de contrôle MIDI. Il utilise des valeurs sur 14 bit au lieu des valeurs usuelles MIDI sur 7 bit.

Comme les arguments d'entrée et de sortie sont nombreux, on peut scinder la ligne en utilisant le caractère '\' (slash inversé) (nouveau dans la version 3.47) pour améliorer la lisibilité. L'utilisation de ces opcodes est considérablement plus efficace que celle de (*ctrl7* et *tonek*) séparés, lorsque l'on a besoin de plus de contrôleurs.

Dans la version au taux-i de *s16b14*, il n'y a pas d'argument de valeur d'entrée initiale. La sortie est prise directement dans l'état courant du tableau interne de contrôleurs de Csound.

## Crédits

Auteur: Gabriel Maldonado  
Italie  
Décember 1998

Nouveau dans la version 3.50 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# s32b14

s32b14 — Crée un banc de 32 numéros différents de messages de contrôle MIDI sur 14-bit.

## Description

Crée un banc de 32 numéros différents de messages de contrôle MIDI sur 14-bit.

## Syntaxe

```
i1,...,i32 s32b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \  
    initvalue1, ifn1,..., ictlno_msb32, ictlno_lsb32, imin32, imax32, initvalue32, ifn32  
  
k1,...,k32 s32b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \  
    initvalue1, ifn1,..., ictlno_msb32, ictlno_lsb32, imin32, imax32, initvalue32, ifn32
```

## Initialisation

*i1 ... i64* -- valeurs de sortie

*ichan* -- canal MIDI (1-16)

*ictlno\_msb1 .... ictlno\_msb32* -- numéro de contrôle MIDI, octet de poids fort (0-127)

*ictlno\_lsb1 .... ictlno\_lsb32* -- numéro de contrôle MIDI, octet de poids faible (0-127)

*imin1 ... imin64* -- valeurs minimales pour chaque contrôleur

*imax1 ... imax64* -- valeurs maximales pour chaque contrôleur

*init1 ... init64* -- valeur initiale pour chaque contrôleur

*ifn1 ... ifn64* -- table de fonction de conversion pour chaque contrôleur

## Exécution

*k1 ... k64* -- valeurs de sortie

*s32b14* est un banc de contrôleurs MIDI, utile lorsque l'on utilise un mélangeur MIDI comme le Kawai MM-16 ou autres pour changer n'importe quel paramètre du son en . Les messages de contrôle MIDI arrivant sur le port d'entrée sont convertis pour entrer dans l'intervalle *iminN*, *imaxN*, et une valeur initiale peut être fixée. On peut aussi utiliser de manière facultative une table de fonction non interpolée avec une courbe de traduction personnalisée pour obtenir, par exemple, des courbes de réponse exponentielles.

Si l'on n'a pas besoin d'une table de traduction, on fixe la valeur de *ifnN* à 0, sinon, on donne à *ifnN* un numéro de table de fonction valide. Lorsque l'on utilise une table de traduction (si *ifnN* reçoit une valeur non nulle faisant référence à une table de fonction déjà allouée), la valeur de *initN* doit être égale à celle de *iminN* ou à celle de *imaxN*, sinon la valeur de sortie initiale sera différente de celle spécifiée dans l'argument *initN*.

*s32b14* fournit un banc de 32 numéros différents de messages de contrôle MIDI. Il utilise des valeurs sur 14 bit au lieu des valeurs usuelles MIDI sur 7 bit.

Comme les arguments d'entrée et de sortie sont nombreux, on peut scinder la ligne en utilisant le caractère '\' (slash inversé) (nouveau dans la version 3.47) pour améliorer la lisibilité. L'utilisation de ces opcodes est considérablement plus efficace que celle de (*ctrl7* et *tonek*) séparés, lorsque l'on a besoin de plus de contrôleurs.

Dans la version au taux-i de *s32b14*, il n'y a pas d'argument de valeur d'entrée initiale. La sortie est prise directement dans l'état courant du tableau interne de contrôleurs de Csound.

## Crédits

Auteur: Gabriel Maldonado  
Italie  
Décember 1998

Nouveau dans la version 3.50 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# samphold

samphold — Effectue une opération d'échantillonnage-bloquage sur son entrée.

## Description

Effectue une opération d'échantillonnage-bloquage sur son entrée.

## Syntaxe

```
ares samphold asig, agate [, ival] [, ivstor]
kres samphold ksig, kgate [, ival] [, ivstor]
```

## Initialisation

*ival*, *ivstor* (facultatif) -- contrôle l'état initial de l'espace mémoire interne. Si *ivstor* vaut zéro la valeur interne « bloquée » est fixée à *ival* ; sinon elle retient sa valeur précédente. Les valeurs par défaut sont 0, 0 (c'est-à-dire initialisation à zéro).

## Exécution

*kgate*, *xgate* -- Contrôle le blocage du signal.

*samphold* effectue une opération d'échantillonnage-blocage sur son entrée en fonction des valeurs de *gate*. Si *gate* != 0, les échantillons en entrée sont transmis en sortie ; si *gate* = 0, la dernière valeur de sortie est répétée. Le paramètre de contrôle *gate* peut être une constante, un signal de contrôle ou un signal audio.

## Exemples

```
asrc buzz      10000, 440, 20, 1      ; train de pulsations à bande limitée
adif diff      asrc                  ; renforcement des aigus
anew balance   adif, asrc             ; mais en conservant la puissance
agate reson    asrc, 0, 440           ; on utilise un filtrage passe-bas de l'original
asamp samphold anew, agate            ; pour laisser passer le nouveau signal audio
aout tone      asamp, 100             ; lissage des discontinuités
```

Voici un autre exemple de l'opcode *samphold*. Il utilise le fichier *samphold.csd* [examples/samphold.csd].

### Exemple 866. Exemple de l'opcode *samphold*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o samphold.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kx line -1, p3, 1 ; between -1 and +1
ktrig metro 1 ; triggers 1 time per second
kval samphold kx, ktrig ; change value whenever ktrig = 1
    printk2 kval ; will print every time kval changes
asig diskinn2 "flute.aiff", 1+kval, 0, 1
    outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 11
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*diff, downsamp, integ, interp, upsamp*

# sandpaper

sandpaper — Modèle semi-physique d'un son de papier de verre.

## Description

*sandpaper* est un modèle semi-physique d'un son de papier de verre. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntaxe

```
ares sandpaper iamp, idettack [, inum] [, idamp] [, imaxshake]
```

## Initialisation

*iamp* -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 128.

*idamp* (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

$$\text{damping\_amount} = 0,998 + (\text{idamp} * 0,002)$$

La valeur par défaut de *damping\_amount* est 0,999 ce qui signifie que la valeur par défaut de *idamp* est 0,5. Le maximum de *damping\_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 1,0.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

*imaxshake* (facultatif) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

## Exemples

Voici un exemple de l'opcode sandpaper. Il utilise le fichier *sandpaper.csd* [examples/sandpaper.csd].

### Exemple 867. Exemple de l'opcode sandpaper.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
```

```

; -o sandpaper.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

idmp = p4
a1 line 2, p3, 2 ;preset amplitude increase
a2 sandpaper 1, 0.01, 128, idmp ;sandpaper needs a little amp help at these settings
asig product a1, a2 ;increase amplitude
outs asig, asig

endin
</CsInstruments>
<CsScore>
i1 0 1 0.5
i1 + 1 0.95

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*cabasa, crunch, sekere, stix*

## Crédits

Auteur : Perry Cook, fait partie de PhOLIES (Physically-Oriented Library of Imitated Environmental Sounds)

Adapté par John ffitich

Université de Bath, Codemist Ltd.

Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.



# scale

scale — Signal de pondération arbitraire.

## Description

Met les valeurs entrantes à l'échelle d'un intervalle défini par l'utilisateur. Semblable à l'objet de pondération que l'on trouve dans les langages de flux de données les plus connus.

## Syntaxe

```
kscl scale kinput, kmax, kmin
```

## Exécution

*kin* -- Valeur d'entrée. Elle peut provenir de n'importe quelle source au taux-k pourvu que la sortie de cette dernière soit comprise entre 0 et 1.

*kmin* -- Valeur minimale de l'intervalle de pondération.

*kmax* -- Valeur maximale de l'intervalle de pondération.

## Exemples

Voici un exemple de l'opcode scale. Il utilise le fichier *scale.csd* [examples/scale.csd].

### Exemple 868. Exemple de l'opcode scale.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  Silent
-odac          -iadc    -d      ;;realtime output
</CsOptions>
<CsInstruments>

sr = 22050
ksmps = 10
nchnls = 2

/*--- */

instr 1 ; scale test

kmod ctrl17 1, 1, 0, 1
printk2 kmod

kout scale kmod, 0, -127
printk2 kout

endin
```

```
/*--- ---*/  
</CsInstruments>  
<CsScore>  
  
i1 0 8888  
  
e  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*gainslider, logcurve, expcurve*

## Crédits

Auteur : David Akbari  
Octobre  
2006

# scalearray

scalearray — Mise à l'échelle des valeurs dans une partie d'un vecteur (tableau unidimensionnel).

## Description

L'opcode *scalearray* met à l'échelle une partie d'un vecteur entre un minimum et un maximum donnés.

## Syntaxe

```
scalearray tarray, kmin, kmax[, kleft, kright]
```

## Exécution

*tarray* -- tableau pour l'opération.

*kmin*, *kmax* -- valeurs du minimum et du maximum de la cible.

*kleft*, *kright* -- partie de la table à utiliser, s'étendant par défaut de 0 à la taille du vecteur.

## Exemples

Voici un exemple de l'opcode *scalearray*. Il utilise le fichier *scalearray.csd* [examples/scalearray.csd].

### Exemple 869. Exemple de l'opcode *scalearray*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n
</CsOptions>
<CsInstruments>
;example by joachim heintz

      seed      0

instr 1
;create an array with 10 elements
kArr[]      init      10
;fill in random numbers and print them out
      printks      "kArr in maximum range 0..100:\n", 0
kIndx      =      0
      until kIndx == 10 do
kNum      random      0, 100
kArr[kIndx] =      kNum
      printf      "kArr[%d] = %10f\n", kIndx+1, kIndx, kNum
kIndx      +=      1
      od
;scale numbers 0...1 and print them out again
      scalearray kArr, 0, 1
kIndx      =      0
      printks      "kArr in range 0..1\n", 0
      until kIndx == 10 do
```

```

    kIndx      printf      "kArr[%d] = %10f\n", kIndx+1, kIndx, kArr[kIndx]
    +=         +=         1
    od
    turnoff
endin
</CsInstruments>
<CsScore>
i1 0 0.1
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*maxarray, minarray, sumarray,*

## Crédits

Auteur : John fitch  
Octobre 2011

Nouveau dans la version 5.14 de Csound.

# scanhammer

scanhammer — Copie d'une table vers une autre avec contrôle du gain.

## Description

C'est une variante de *tablecopy*, qui copie d'une table vers une autre, à partir de *ipos*, et avec un contrôle du gain. Le nombre de points copiés est déterminé par la longueur de la source. Les autres points ne sont pas changés. On peut utiliser cet opcode pour « frapper » une corde dans le code de synthèse par balayage.

## Syntaxe

```
scanhammer isrc, idst, ipos, imult
```

## Initialisation

*isrc* -- table de fonction source.

*idst* -- table de fonction destination.

*ipos* -- position de départ (en points).

*imult* -- multiplicateur du gain. S'il vaut 0, les valeurs ne seront pas modifiées.

## Voir aussi

*scantable*

## Crédits

Auteur : Matt Gilliard  
Avril 2002

Nouveau dans la version 4.20

# scans

scans — Génère une sortie audio au moyen de la synthèse par balayage.

## Description

Opcode du greffon scansyn.

Génère une sortie audio au moyen de la synthèse par balayage.

## Syntaxe

```
ares scans kamp, kfreq, ifn, id [, iorder]
```

## Initialisation

*ifn* -- ftable contenant la trajectoire du balayage. C'est une série de nombres qui contiennent les adresses des masses. L'ordre de ces adresses est utilisé comme chemin de balayage. Ne doit pas contenir de valeurs supérieures au nombre de masses, ou des nombres négatifs. Voir l'*introduction à la section sur la synthèse par balayage*.

*id* -- numéro d'ID de la forme d'onde de l'opcode *scanu* à utiliser.

*iorder* (facultatif, 0 par défaut) -- ordre de l'interpolation utilisée en interne. Peut prendre n'importe quelle valeur comprise entre 1 et 4, et vaut 4 par défaut si la valeur donnée est en dehors de cet interval. 4 est l'interpolation quartique, 3 est l'interpolation cubique, 2 est l'interpolation quadratique et 1 l'interpolation linéaire. Les nombres les plus élevés donnent un traitement plus lent, mais pas nécessairement meilleur.

## Exécution

*kamp* -- amplitude de la sortie. Noter que l'amplitude résultante dépend aussi des valeurs instantanées de la table d'onde. Ce nombre est en fait la facteur de pondération de la table d'onde.

*kfreq* -- fréquence de balayage

## Exemples

Voici un exemple de synthèse par balayage. Il utilise les fichiers *scans.csd* [examples/scans.csd], et *string-128.matrix* [examples/string-128.matrix].

### Exemple 870. Exemple de l'opcode scans.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in    No messages
```

```

-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o scans.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

    sr =      44100
    ksmpr =    128
    nchnls =    1

    instr 1
a0 = 0
; scanu init, irate, ifnvel, ifnmass, ifnstif, ifncentr, ifndamp, kmass, kstif, kcentr, kdamp, ileft,
; scanu 1, .01, 6, 2, 3, 4, 5, 2, .1, .1, -.01, .1,
;ar scans kamp, kfreq, ifntraj, id
a1 scans ampd(p4), cpspch(p5), 7, 2
    out a1
    endin

</CsInstruments>
<CsScore>

; Initial condition
f1 0 128 7 0 64 1 64 0

; Masses
f2 0 128 -7 1 128 1

; Spring matrices
f3 0 16384 -23 "string-128.matrix"

; Centering force
f4 0 128 -7 0 128 2

; Damping
f5 0 128 -7 1 128 1

; Initial velocity
f6 0 128 -7 0 128 0

; Trajectories
f7 0 128 -5 .001 128 128

; Note list
i1 0 10 86 6.00
i1 11 14 86 7.00
i1 15 20 86 5.00
e

</CsScore>
</CsSoundSynthesizer>

```

Voici un autre exemple de synthèse par balayage qui utilise des échantillons comme signal d'excitation. Il utilise le fichier *scans-2.csd* [examples/scans-2.csd].

### Exemple 871. Second exemple de l'opcode scans.

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:

```

```
; -o scans-2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

strset 1, "mary.wav"
strset 2, "fox.wav"

instr 2 ;show 2 different trajectories, with samples as excitation signal

ismp = p6
iamp = p7
itrj = p8
aout soundin p6 ;choose wave file
      scanu ismp, .01, 6, 2, 33, 44, 5, 2, .01, .05, -.05, .1, .5, 0, 0, aout, 1, 0
asig scans iamp, cpspch(p5), itrj, 0
      outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 128 7 0 64 1 64 0 ; Initial condition
f2 0 128 -7 1 128 0.3 ; Masses
f33 0 16384 -23 "cylinder-128,8" ; Spring matrices
f44 0 128 -7 2 4 0 124 2 ; Centering force
f5 0 128 -7 1 128 0 ; Damping
f6 0 128 -7 -.0 128 0 ; Initial velocity
f7 0 128 -5 .001 128 128 ; Trajectories
f77 0 128 -23 "128-spiral-8,16,128,2,lover2"

s
i2 0 5 63 6.00 1 .9 7 ;"mary.wav" &
i2 6 5 60 7.00 ;trajectory table 7
i2 10 5 60 8.00

s
i2 0 5 63 6.00 2 .08 7 ;"fox.wav", at much lower volume
i2 6 5 60 7.00
i2 10 5 60 8.00

s
i2 0 5 63 6.00 1 .9 77 ;"mary.wav" &
i2 6 5 60 7.00 ;trajectory table 77
i2 10 5 60 8.00

s
i2 0 5 63 6.00 2 .08 77 ;"fox.wav", at much lower volume
i2 6 5 60 7.00
i2 10 5 60 8.00
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

Le fichier de la matrice « string-128.matrix », ainsi que d'autres matrices, sont aussi disponibles dans un *fichier zippé* [<http://www.csounds.com/scanned/zip/scanmatrices.zip>] depuis la *page Scanned Synthesis* [<http://www.csounds.com/scanned/>] à cSounds.com.

Plus d'information sur cet opcode : [http://www.csounds.com/stevenyi/scanned/yi\\_scannedSynthesis.html](http://www.csounds.com/stevenyi/scanned/yi_scannedSynthesis.html) [[http://www.csounds.com/stevenyi/scanned/yi\\_scannedSynthesis.html](http://www.csounds.com/stevenyi/scanned/yi_scannedSynthesis.html)], écrit par Steven Yi.



## Crédits

Auteur : Paris Smaragdis  
MIT Media Lab  
Boston, Massachussetts USA

Nouveau dans la version 4.05 de Csound

# scantable

scantable — Une implémentation simplifiée de la synthèse par balayage.

## Description

Une implémentation simplifiée de la synthèse par balayage. C'est l'implémentation d'une corde circulaire parcourue au moyen de tables externes. Cet opcode permet la modification directe et la lecture des valeurs avec les opcodes de table.

## Syntaxe

```
aout scantable kamp, kpch, ipos, imass, istiff, idamp, ivel
```

## Initialisation

*ipos* -- table contenant le tableau de position.

*imass* -- table contenant la masse de la corde.

*istiff* -- table contenant la raideur de la corde.

*idamp* -- table contenant les facteurs d'atténuation de la corde.

*ivel* -- table contenant les vitesses.

## Exécution

*kamp* -- amplitude (gain) de la corde.

*kpch* -- la fréquence de balayage de la corde.

## Exemples

Voici un exemple de l'opcode scantable. Il utilise le fichier *scantable.csd* [examples/scantable.csd].

### Exemple 872. Exemple de l'opcode scantable.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0 ;;realtime audio out and midi in
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o scantable.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
```

```

nchnls = 2
odbfs = 1

gi1 ftgen 1, 0, 128, 7, 0, 64, 1, 64, 0 ; initial position
gi2 ftgen 2, 0, 128, -7, 1, 128, 1 ; masses
gi3 ftgen 3, 0, 128, -7, 0, 64, 100, 64, 0 ; stiffness
gi4 ftgen 4, 0, 128, -7, 1, 128, 1 ; damping
gi5 ftgen 5, 0, 128, -7, 0, 128, 0.5 ; initial velocity

instr 1

iamp ampmidi .5
ipch cpsmidi
kenv madsr .1, .1, .8, .3

asig scantable iamp, ipch, 1, 2, 3, 4, 5
asig dcblock asig
      outs asig*kenv, asig*kenv

endin
</CsInstruments>
<CsScore>

f0 60 ; play for 60 seconds
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*scanhammer*

Plus d'information sur cet opcode : [http://www.csounds.com/stevenyi/scanned/yi\\_scannedSynthesis.html](http://www.csounds.com/stevenyi/scanned/yi_scannedSynthesis.html)  
 [http://www.csounds.com/stevenyi/scanned/yi\_scannedSynthesis.html], écrit par Steven Yi.

## Crédits

Auteur : Matt Gilliard  
 Avril 2002

Nouveau dans la version 4.20

# scanu

scanu — Calcule la forme d'onde et la table d'onde à utiliser dans la synthèse par balayage.

## Description

Opcode du greffon scansyn.

Calcule la forme d'onde et la table d'onde à utiliser dans la synthèse par balayage.

## Syntaxe

```
scanu init, irate, ifnvel, ifnmass, ifnstif, ifncentr, ifndamp, kmass, \  
      kstif, kcentr, kdamp, ileft,  iright, kpos, kstrngth, ain, idisp, id
```

## Initialisation

*init* -- la position initiale des masses. Si c'est un nombre négatif, alors la valeur absolue de *init* indique la table à utiliser pour la forme du marteau. Si *init* > 0, il représente le nombre de masses attendu.

*ifnvel* -- ftable contenant la vitesse initiale de chaque masse. Sa taille est le nombre de masses attendu.

*ifnmass* -- ftable contenant la valeur de chaque masse. Sa taille est le nombre de masses attendu.

*ifnstif* -- ftable contenant la raideur du ressort de chaque connexion. Sa taille est le carré du nombre de masses attendu. Ses données sont ordonnées selon la succession des lignes de la matrice de connexion du système.

*ifncentr* -- ftable contenant la force de centrage de chaque masse. Sa taille est le nombre de masses attendu.

*ifndamp* -- ftable contenant le facteur d'amortissement de chaque masse. Sa taille est le nombre de masses attendu.

*ileft* -- si *init* < 0, position du marteau de gauche (*ileft* = 0 frappe complètement à gauche, *ileft* = 1 frappe complètement à droite).

*iright* -- si *init* < 0, position du marteau de droite (*iright* = 0 frappe complètement à gauche, *iright* = 1 frappe complètement à droite).

*idisp* -- s'il vaut 0, il n'y a pas d'affichage des masses.

*id* -- s'il est positif, c'est l'ID de l'opcode. Il est utilisé pour relier l'opcode de balayage au bon générateur de forme d'onde. S'il est négatif, sa valeur absolue indique la table d'onde dans laquelle sera écrite la forme d'onde. Cette forme d'onde peut être utilisée par la suite par un autre opcode pour générer du son. Le contenu initial de cette table sera écrasé.

## Exécution

*kmass* -- pondère les masses

*kstif* -- pondère la raideur des ressorts

*kcentr* -- pondère la force de centrage

*kdamp* -- pondère l'amortissement

*kpos* -- position d'un marteau actif le long de la corde (*kpos* = 0 est complètement à gauche, *kpos* = 1 est complètement à droite). La forme du marteau est déterminée par *init* et sa puissance de percussion est *kstrngth*.

*kstrngth* -- puissance utilisée par le marteau actif

*ain* -- entrée audio qui s'ajoute à la vitesse des masses. L'amplitude ne doit pas être trop grande.

## Exemples

Pour un exemple, voir la documentation de *scans*.

## Voir aussi

Plus d'information sur cet opcode : [http://www.csounds.com/stevenyi/scanned/yi\\_scannedSynthesis.html](http://www.csounds.com/stevenyi/scanned/yi_scannedSynthesis.html)  
[[http://www.csounds.com/stevenyi/scanned/yi\\_scannedSynthesis.html](http://www.csounds.com/stevenyi/scanned/yi_scannedSynthesis.html)], écrit par Steven Yi.

## Crédits

Auteur : Paris Smaragdis  
MIT Media Lab  
Boston, Massachusetts USA  
Mars 2000

Nouveau dans la version 4.05 de Csound

# schedkwhen

`schedkwhen` — Ajoute un nouvel évènement de partition généré par un signal de déclenchement de taux-k.

## Description

Ajoute un nouvel évènement de partition généré par un signal de déclenchement de taux-k.

## Syntaxe

```
schedkwhen ktrigger, kmintim, kmaxnum, kinsnum, kwhen, kdur \  
[, ip4] [, ip5] [...]  
  
schedkwhen ktrigger, kmintim, kmaxnum, "insname", kwhen, kdur \  
[, ip4] [, ip5] [...]
```

## Initialisation

« *insname* » -- Une chaîne de caractères (entre guillemets) représentant un instrument nommé.

*ip4*, *ip5*, ... -- Equivalent à *p4*, *p5*, etc., dans une *instruction i* de partition.

## Exécution

*ktrigger* -- déclenche un nouvel évènement de partition. Si *ktrigger* = 0, aucun nouvel évènement n'est déclenché.

*kmintim* -- intervalle de temps minimum entre les évènements générés, en secondes. Si *kmintim* <= 0, il n'y a aucune limite de temps. Si *kinsnum* est négatif (pour arrêter un instrument), ce test est ignoré.

*kmaxnum* -- nombre maximum d'instances simultanées de l'instrument *kinsnum* autorisées. Si le nombre d'instances existantes de *kinsnum* est >= *kmaxnum*, aucun nouvel évènement n'est généré. Si *kmaxnum* est <= 0, il n'est pas utilisé pour limiter la génération d'évènement. Si *kinsnum* est négatif (pour arrêter un instrument), ce test est ignoré.

*kinsnum* -- numéro d'un instrument. Equivalent à *p1* dans une *instruction i* de partition.

*kwhen* -- date de début du nouvel évènement. Equivalent à *p2* dans une *instruction i* de partition. Mesurée à partir de l'instant de l'évènement déclencheur. *kwhen* doit être >= 0. Si *kwhen* > 0, l'instrument ne sera pas initialisé jusqu'à ce que cette date soit atteinte.

*kdur* -- durée de l'évènement. Equivalent à *p3* dans une *instruction i* de partition. Si *kdur* = 0, l'instrument ne fera qu'une phase d'initialisation, sans exécution. Si *kdur* est négatif, une note tenue est démarrée. (Voir *ihold* et *instruction i*.)



### Note

Dans l'attente d'évènements à déclencher par *schedkwhen*, l'exécution doit continuer, ou Csound pourrait se terminer si aucun évènement de partition n'est attendu. Pour garantir une exécution continue, on peut utiliser une *instruction f0* dans la partition.



## Note

Noter que l'opcode *schedkwhen* ne peut pas accepter de p-champs chaîne de caractère. Si vous devez passer des chaînes de caractère à l'instanciation d'un instrument, utilisez l'opcode *scoreline* ou *scoreline\_i*.

## Exemples

Voici une exemple de l'opcode *schedkwhen*. Il utilise le fichier *schedkwhen.csd* [examples/schedkwhen.csd].

### Exemple 873. Exemple de l'opcode *schedkwhen*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
-odac             -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o schedkwhen.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1 - oscillator with a high note.
instr 1
; Use the fourth p-field as the trigger.
ktrigger = p4
kmintim = 0
kmaxnum = 2
kinsnum = 2
kwhen = 0
kdur = 0.5

; Play Instrument #2 at the same time, if the trigger is set.
schedkwhen ktrigger, kmintim, kmaxnum, kinsnum, kwhen, kdur

; Play a high note.
a1 oscils 10000, 880, 1
out a1
endin

; Instrument #2 - oscillator with a low note.
instr 2
; Play a low note.
a1 oscils 10000, 220, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
```

```
f 1 0 16384 10 1

; p4 = trigger for Instrument #2 (when p4 > 0).
; Play Instrument #1 for half a second, no trigger.
i 1 0 0.5 0
; Play Instrument #1 for half a second, trigger Instrument #2.
i 1 1 0.5 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*event, event\_i, schedule, schedwhen, schedkwhennamed, scoreline, scoreline\_i*

## Crédits

Auteur : Rasmus Ekman  
EMS, Stockholm, Suède

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.59 de Csound



# schedkwhennamed

`schedkwhennamed` — Semblable à `schedkwhen` mais avec un instrument nommé dans la phase d'initialisation.

## Description

Semblable à `schedkwhen` mais avec un instrument nommé dans la phase d'initialisation.

## Syntaxe

```
schedkwhennamed ktrigger, kmintim, kmaxnum, "name", kwhen, kdur \
    [, ip4] [, ip5] [...]
```

## Initialisation

*ip4*, *ip5*, ... -- Equivalent à *p4*, *p5*, etc., dans une *instruction i* de partition.

## Exécution

*ktrigger* -- déclenche un nouvel évènement de partition. Si *ktrigger* = 0, aucun nouvel évènement n'est déclenché.

*kmintim* -- intervalle de temps minimum entre les évènements générés, en secondes. Si *kmintim* est inférieur ou égal à 0, il n'y a aucune limite de temps.

*kmaxnum* -- nombre maximum d'instances simultanées de l'instrument nommé autorisées. Si le nombre d'instances existantes de l'instrument nommé est supérieur ou égal à *kmaxnum*, aucun nouvel évènement n'est généré. Si *kmaxnum* est inférieur ou égal à 0, il n'est pas utilisé pour limiter la génération d'évènement.

"*name*" -- le nom de l'instrument.

*kwhen* -- date de début du nouvel évènement. Equivalent à *p2* dans une *instruction i* de partition. Mesurée à partir de l'instant de l'évènement déclencheur. *kwhen* doit être supérieure ou égale à 0. Si *kwhen* est supérieure à 0, l'instrument ne sera pas initialisé jusqu'à ce que cette date soit atteinte.

*kdur* -- durée de l'évènement. Equivalent à *p3* dans une *instruction i* de partition. Si *kdur* vaut 0, l'instrument ne fera qu'une phase d'initialisation, sans exécution. Si *kdur* est négatif, une note tenue est démarrée. (Voir *ihold* et *instruction i*.)



### Note

Dans l'attente d'évènements à déclencher par `schedkwhennamed`, l'exécution doit continuer, ou Csound pourrait se terminer si aucun évènement de partition n'est attendu. Pour garantir une exécution continue, on peut utiliser une *instruction f0* dans la partition.



### Note

Noter que l'opcode `schedkwhennamed` ne peut pas accepter de p-champs chaîne de caractère. Si vous devez passer des chaînes de caractère à l'instanciation d'un instrument, utilisez l'opcode `scoreline` ou `scoreline_i`.

## Exemples

Voici un exemple de l'opcode `schedkwhennamed`. Il utilise le fichier `schedkwhennamed.csd` [examples/schedkwhennamed.csd].

### Exemple 874. Exemple de l'opcode `schedkwhennamed`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc    -d
; For Non-realtime ouput leave only the line below:
; -o schedkwhennamed.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

    sr      = 48000
    ksmpts  = 16
    nchnls  = 2
    odbfs   = 1

; Example by Jonathan Murphy 2007

    gSinstr2 = "printer"

    instr 1

        ktrig    metro    1
        if (ktrig == 1) then
            ;Call instrument "printer" once per second
            schedkwhennamed ktrig, 0, 1, gSinstr2, 0, 1
        endif

    endin

    instr printer

        ktime    timeinsts
        printk2   ktime

    endin

</CsInstruments>
<CsScore>
i1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*event, event\_i, schedule, schedwhen, schedkwhen, scoreline, scoreline\_i*

## Crédits

Auteur : Rasmus Ekman

EMS, Stockholm, Suède

Nouveau dans la version 4.23 de Csound

# schedule

schedule — Ajoute un nouvel évènement de partition.

## Description

Ajoute un nouvel évènement de partition.

## Syntaxe

```
schedule insnum, iwhen, idur [, ip4] [, ip5] [...]  
schedule "insname", iwhen, idur [, ip4] [, ip5] [...]
```

## Initialisation

*insnum* -- numéro d'un instrument. Equivalent à p1 dans une *instruction i* de partition. *insnum* doit être un numéro supérieur au numéro de l'instrument appelant.

« *insname* » -- une chaîne de caractères (entre guillemets) représentant un instrument nommé.

*iwhen* -- date de début du nouvel évènement. Equivalent à p2 dans une *instruction i* de partition. *iwhen* ne doit pas être négatif. Si *iwhen* vaut zéro, *insnum* doit être supérieur ou égal au p1 de l'instrument courant.

*idur* -- durée de l'évènement. Equivalent à p3 dans une *instruction i* de partition.

*ip4*, *ip5*, ... -- Equivalent à p4, p5, etc., dans une *instruction i* de partition. L'opcode accepte aussi des chaînes de caractères en p4-pN

## Exécution

*schedule* ajoute un nouvel évènement de partition. Les arguments, options incluses, sont les mêmes que dans une partition. Le temps *iwhen* (p2) est mesuré à partir de l'instant de cet évènement.

Si la durée est nulle ou négative, le nouvel évènement est de type MIDI, et il hérite le sous-évènement de relachement (release) de l'instruction *schedule*.

## Exemples

Voici un exemple de l'opcode *schedule*. Il utilise le fichier *schedule.csd* [examples/schedule.csd].

### Exemple 875. Exemple de l'opcode *schedule*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
; Audio out  Audio in  
-odac      -iadc      ;;RT audio I/O  
; For Non-realtime ouput leave only the line below:  
; -o schedule.wav -W ;;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - oscillator with a high note.
instr 1
; Play Instrument #2 at the same time.
schedule 2, 0, p3

; Play a high note.
a1 oscils 10000, 880, 1
out a1
endin

; Instrument #2 - oscillator with a low note.
instr 2
; Play a low note.
a1 oscils 10000, 220, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for half a second.
i 1 0 0.5
; Play Instrument #1 for half a second.
i 1 1 0.5
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*event, event\_i, schedulek, schedwhen, schedkwhen, schedkwhennamed, scoreline, scoreline\_i*

Pour plus d'information sur cet opcode : [http://www.csoundjournal.com/issue15/phrase\\_loops.html](http://www.csoundjournal.com/issue15/phrase_loops.html)  
[\[http://www.csoundjournal.com/issue15/phrase\\_loops.html\]](http://www.csoundjournal.com/issue15/phrase_loops.html), écrit par Jim Aikin.

## Crédits

Auteur : John ffitch  
 Université de Bath/Codemist Ltd.  
 Bath, UK  
 Novembre 1998

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.491 de Csound

Basé sur un travail de Gabriel Maldonado

Merci à David Gladstein, pour avoir clarifié le paramètre *iwhen*.

# schedulek

schedulek — Ajoute un nouvel évènement de partition.

## Description

Ajoute un nouvel évènement de partition.

## Syntaxe

```
schedulek knsnum, kwhen, kdur [, kp4] [, kp5] [...]
```

```
schedulek "insname", kwhen, kdur [, kp4] [, kp5] [...]
```

## Exécution

*knsnum* -- numéro d'un instrument. Equivalent à p1 dans une *instruction i* de partition. *knsnum* doit être un numéro supérieur au numéro de l'instrument appelant.

« *insname* » -- une chaîne de caractères (entre guillemets) représentant un instrument nommé.

*kwhen* -- date de début du nouvel évènement. Equivalent à p2 dans une *instruction i* de partition. *kwhen* ne doit pas être négatif. Si *kwhen* vaut zéro, *knsnum* doit être supérieur ou égal au p1 de l'instrument courant.

*kdur* -- durée de l'évènement. Equivalent à p3 dans une *instruction i* de partition.

*kp4*, *kp5*, ... -- Equivalent à p4, p5, etc., dans une *instruction i* de partition. L'opcode accepte aussi des chaînes de caractères en p4-pN

*schedulek* ajoute un nouvel évènement de partition. Les arguments, options incluses, sont les mêmes que dans une partition. Le temps *kwhen* (p2) est mesuré à partir de l'instant de cet évènement.

Si la durée est nulle ou négative, le nouvel évènement est de type MIDI, et il hérite le sous-évènement de relachement (release) de l'instruction *schedule*.

## See also

*event*, *event\_i*, *schedule*, *schedwhen*, *schedkwhen*, *schedkwhennamed*, *scoreline*, *scoreline\_i*

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Janvier 2020

# schedwhen

schedwhen — Ajoute un nouvel évènement de partition.

## Description

Ajoute un nouvel évènement de partition.

## Syntaxe

```
schedwhen ktrigger, kinsnum, kwhen, kdur [, ip4] [, ip5] [...]
```

```
schedwhen ktrigger, "insname", kwhen, kdur [, ip4] [, ip5] [...]
```

## Initialisation

*ip4, ip5, ...* -- Equivalent à p4, p5, etc., dans une *instruction i* de partition.

## Exécution

*kinsnum* -- numéro d'un instrument. Equivalent à p1 dans une *instruction i* de partition.

« *insname* » -- une chaîne de caractères (entre guillemets) représentant un instrument nommé.

*ktrigger* -- valeur de déclenchement pour le nouvel évènement.

*kwhen* -- date de début du nouvel évènement. Equivalent à p2 dans une *instruction i* de partition.

*kdur* -- durée de l'évènement. Equivalent à p3 dans une *instruction i* de partition.

*schedwhen* ajoute un nouvel évènement de partition. L'évènement n'est programmé que lorsque la valeur de taux-k *ktrigger* prend une valeur non nulle. Les arguments, options incluses, sont les mêmes que dans une partition. Le temps *kwhen* (p2) est mesuré à partir de l'instant de cet évènement.

Si la durée est nulle ou négative, le nouvel évènement est de type MIDI, et il hérite le sous-évènement de relachement (release) de l'instruction *schedwhen*.



### Note

Noter que l'opcode *schedwhen* ne peut pas accepter de p-champs chaîne de caractère. Si vous devez passer des chaînes de caractère à l'instanciation d'un instrument, utilisez l'opcode *scoreline* ou *scoreline\_i*.

## Exemples

Voici une exemple de l'opcode *schedwhen*. Il utilise le fichier *schedwhen.csd* [examples/schedwhen.csd].

### Exemple 876. Exemple de l'opcode *schedwhen*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.



```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o schedwhen.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kmtr metro 100      ;produce 100 triggers per second
    schedwhen kmtr, 2, 1, .3 ;but schedwhen plays instr. 2 only once

endin

instr 2

aenv linseg 0, p3*.1, 1, p3*.3, 1, p3*.6, 0 ;envelope
a1 poscil .3*aenv, 1000, 1
    outs a1, a1

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine

i 1 0 3
i 1 3 5
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*event, event\_i, schedule, schedkwhen, schedkwhennamed, scoreline, scoreline\_i*

## Crédits

Auteur : John ffitch  
 Université de Bath/Codemist Ltd.  
 Bath, UK  
 Novembre 1998

Nouveau dans la version 3.491 de Csound

Basé sur un travail de Gabriel Maldonado

# scoreline

scoreline — Délivre un ou plusieurs évènements de ligne de partition depuis un instrument.

## Description

*scoreline* délivre un ou plusieurs évènements de partition, si *ktrig* vaut 1, à chaque période *k*. Il peut gérer les chaînes de caractères dans les mêmes conditions que dans la partition standard. Les chaînes de caractères sur plusieurs lignes sont acceptées, en utilisant `{ { }` pour encadrer la chaîne de caractères.

## Syntaxe

```
scoreline Sin, ktrig
```

## Initialisation

« *Sin* » -- une chaîne de caractères (entre guillemets ou encadrée par `{ { }`), contenant un ou plusieurs évènements de partition.

## Exécution

« *ktrig* » -- déclencheur d'évènement, 1 délivre l'évènement de partition, 0 l'ignore.

## Exemples

Voici un exemple de l'opcode *scoreline*. Il utilise le fichier *scoreline.csd* [examples/scoreline.csd].

### Exemple 877. Exemple de l'opcode *scoreline*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;;realtime audio out
-iadc      ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o scoreline.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ktrig metro 3      ;trigger 3 times a second
scoreline {{      ;so it sounds like an echo
  i 2 0 3 "flute.aiff"
  i 2 1 3 "beats.wav"
}}, ktrig
```

```
ktrig = 0

endin

instr 2

asig soundin p4
    outs asig*.3, asig*.3

endin
</CsInstruments>
<CsScore>

i1 0 2 ;play for 2 seconds, so the samples are played 6 times
e
</CsScore>
</CsoundSynthesizer>
```

On peut utiliser des opcodes de chaîne de caractères comme *sprintfk* pour produire les chaînes de caractères à passer à *scoreline* comme ceci :

```
Sfil = "/Volumes/Bla/file.aif"
String sprintfk {{i 2 0 %f "%s" %f %f %f %f}}, idur, Sfil, p5, p6, knorm, iskip
scoreline String, ktrig
```

## Voir aussi

*event*, *event\_i*, *schedule*, *schedwhen*, *schedkwhen*, *schedkwhennamed*, *scoreline\_i*

Pour plus d'information sur cet opcode : [http://www.csoundjournal.com/issue15/phrase\\_loops.html](http://www.csoundjournal.com/issue15/phrase_loops.html) [[http://www.csoundjournal.com/issue15/phrase\\_loops.html](http://www.csoundjournal.com/issue15/phrase_loops.html)], écrit par Jim Aikin.

Egalement dans les Floss Manuals : <http://write.flossmanuals.net/csound/f-live-events/> [<http://write.flossmanuals.net/csound/f-live-events/>].

## Crédits

Auteur : Victor Lazzarini, 2007

# scoreline\_i

scoreline\_i — Délivre un ou plusieurs événements de ligne de partition depuis un instrument pendant la phase d'initialisation.

## Description

*scoreline\_i* délivre des événements de partition au cours de la phase d'initialisation. Il peut gérer les chaînes de caractères dans les mêmes conditions que dans la partition standard. Les chaînes de caractères sur plusieurs lignes sont acceptées, en utilisant `{ { }` pour encadrer la chaîne de caractères.

## Syntaxe

```
scoreline_i Sin
```

## Initialisation

« *Sin* » -- une chaîne de caractères (entre guillemets ou encadrée par `{ { }`), contenant un ou plusieurs événements de partition.

## Exemples

Voici en exemple de l'opcode *scoreline\_i*. Il utilise le fichier *scoreline\_i.csd* [examples/scoreline\_i.csd].

### Exemple 878. Exemple de l'opcode *scoreline\_i*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac --old-parser    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o scoreline.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

scoreline_i {{
    i 2 0 3 "flute.aiff"
    i 2 1 3 "beats.wav"
}}

endin

instr 2
```

```
asig soundin p4
outs asig*.8, asig*.8

endin
</CsInstruments>
<CsScore>

i1 0 1
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*event, event\_i, schedule, schedwhen, schedkwhen, schedkwhennamed, scoreline*

Pour plus d'information sur cet opcode : [http://www.csoundjournal.com/issue15/phrase\\_loops.html](http://www.csoundjournal.com/issue15/phrase_loops.html) [[http://www.csoundjournal.com/issue15/phrase\\_loops.html](http://www.csoundjournal.com/issue15/phrase_loops.html)], écrit par Jim Aikin.

Egalement dans les Floss Manuals : <http://write.flossmanuals.net/csound/f-live-events/> [<http://write.flossmanuals.net/csound/f-live-events/>].

## Crédits

Auteur : Victor Lazzarini, 2007

# sc\_lag

sc\_lag — Lag exponentiel.

## Description

Opcode du greffon scugens.

Lag exponentiel avec durée de latence à 60 dB. Adapté du Lag de Supercollider.

## Syntaxe

```
aout sc_lag ain, klagtime [, initialvalue=0]
kout sc_lag kin, klagtime [, initialvalue=0]
```

## Initialisation

*initialvalue* -- Si elle est fournie, fixe l'état interne. 0 par défaut.

## Exécution

*ain* -- Signal en entrée.

*klagtime* -- Durée de latence à 60 dB en secondes.

*kladown* -- Durée de latence à 60 dB en secondes pour le signal atténué.

C'est essentiellement un filtre à un pôle sauf qu'au lieu de fournir le coefficient directement, celui-ci est calculé en fonction d'un temps de latence à 60 dB. C'est le temps nécessaire pour que le filtre converge à 0.01% d'une valeur. Utile pour lisser un signal de contrôle.

## Exemples

Voici un exemple de l'opcode sc\_lag. Il utilise le fichier *sc\_lag.csd* [examples/sc\_lag.csd].

### Exemple 879. Exemple de l'opcode sc\_lag.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2
0dbfs = 1.0

instr 1
  ; smooth a krate signal
```

```
kx = floor(line(0, p3, 10))
kx2 = sc_lag(kx, 0.1)
printk2 kx2
endin

instr 2
; smooth an audio signal
kmidi = floor(line(60, p3, 72)/2)*2
afreq = upsamp(mtof(kmidi))
afreqsmooth = sc_lag(afreq, 1)
a1 = oscili(1, afreq)
a2 = oscili(1, afreqsmooth)
outch 1, a1
outch 2, a2
endin

</CsInstruments>
<CsScore>
i 1 0 5
i 2 0 10

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*sc\_lagud, port, portk*

## Crédits

Par : Eduardo Moguillansky 2017

# sc\_lagud

sc\_lagud — Lag exponentiel.

## Description

Opcode du greffon scugens.

Lag exponentiel avec différentes durées de lissage pour le signal montant et atténué. Adapté du LagUD de Supercollider.

## Syntaxe

```
aout sc_lagud ain, klagup, klagdown
```

```
kout sc_lagud kin, klagup, klagdown
```

## Initialisation

*initialvalue* -- Si elle est fournie, fixe l'état interne. 0 par défaut.

## Exécution

*ain* -- Signal en entrée.

*klagup* -- Durée de latence à 60 dB en secondes pour le signal montant.

*klagdown* -- Durée de latence à 60 dB en secondes pour le signal atténué.

C'est semblable à `sc_lag` sauf que l'on peut donner des durées de latence à 60 dB différentes pour le signal montant et pour le signal atténué. Utile pour lisser des signaux de contrôle pour lesquels le fondu doit être différent de l'atténuation.

## Exemples

Voici un exemple de l'opcode `sc_lagud`. Il utilise le fichier `sc_lagud.csd` [examples/sc\_lagud.csd].

### Exemple 880. Exemple de l'opcode `sc_lagud`.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2
0dbfs = 1.0

instr 1
```



```

    kx = randh(2, 1)
    kx2 = sc_lagud(kx, 1.0, 0.1, 1)
    printks "x %f x2 %f\n", 0.1, kx, kx2
endin

instr 2
  kmidis[] fillarray 60, 65, 60, 65, 60
  ilen = lenarray(kmidis)
  kidx = int(linseg(0, ilen*2, ilen-0.00000001))
  kmidi = mtof(kmidis[kidx])
  afreq = sc_lagud(a(kmidi), 1, 0.1)
  a0 = oscili(0.7, afreq)
  kfreq = sc_lagud(kmidi, 1, 0.1)
  a1 = oscili(0.7, kfreq)
  outch 1, a0
  outch 2, a1
endin

</CsInstruments>
<CsScore>
; i 1 0 10
i 2 0 12

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*sc\_lag, port, portk*

## Crédits

Par : Eduardo Moguillansky 2017

# sc\_phasor

sc\_phasor — Une rampe linéaire réinitialisable entre deux niveaux.

## Description

Opcode du greffon scugens.

Une rampe linéaire réinitialisable entre deux niveaux. Adapté du Phasor de Supercollider.

## Syntaxe

```
aindex sc_phasor xtrig, xrate, kstart, kend [, kresetPos]
```

```
kindex sc_phasor xtrig, xrate, kstart, kend [, kresetPos]
```

## Exécution

sc\_phasor produit une rampe linéaire entre les valeurs de *kstart* et de *kend*. Quand son entrée de déclenchement passe de valeurs non positives à des valeurs positives, la sortie de sc\_phasor saute à sa position de reset (ou à *kstart* si aucune valeur de reset n'a été donnée). Lorsqu'il atteint la fin de sa rampe, sc\_phasor recommence au début.

Si *kresetPos* est spécifié, cette valeur sera utilisée après un déclenchement. Sinon, un déclenchement ramènera à la position de *kstart*.

Si l'on veut que sc\_phasor produise un signal de fréquence *freq* oscillant entre *start* et *end*, la valeur de *rate* doit être :

$$(end - start) * freq / sr$$

## Exemples

Voici un exemple de l'opcode sc\_phasor. Il utilise le fichier *sc\_phasor.csd* [examples/sc\_phasor.csd].

### Exemple 881. Exemple de l'opcode sc\_phasor.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2
0dbfs = 1.0

instr 1
  krate linseg 1, p3, 40
  ktrig metro krate
  kx sc_phasor ktrig, krate/kr, 0, 1
```

```
asine oscili 0.2, kx*500+500
outch 1, asine
endin

instr 2
krate linseg 1, p3, 40
atrig = mpulse(1, 1/krate)
ax sc_phasor atrig, krate/sr, 0, 1
asine oscili 0.2, ax*500+500
outch 2, asine
endin

</CsInstruments>
<CsScore>
i1 0 20
i2 0 20
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*phasor*

## Crédits

Par : Eduardo Moguillansky 2017

# sc\_trig

sc\_trig — Déclencheur mesuré.

## Description

Opcodes du greffon scugens.

Déclencheur mesuré. Adapté de Trig de Supercollider.

## Syntaxe

```
aout sc_trig ain, kdur
kout sc_trig kin, kdur
```

## Exécution

*ain* -- Déclencheur, pouvant être n'importe quel signal. Un déclenchement a lieu quand le signal passe d'une valeur non-positive à une valeur positive.

*kdur* -- Durée du déclenchement en secondes.

Quand une transition d'une valeur non-positive à une valeur positive a lieu en entrée, sc\_trig restitue le niveau du signal déclencheur pendant la durée spécifiée, sinon la sortie vaut zéro.

## Exemples

Voici un exemple de l'opcode sc\_trig. Il utilise le fichier *sc\_trig.csd* [examples/sc\_trig.csd].

### Exemple 882. Exemple de l'opcode sc\_trig.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2
0dbfs = 1.0

instr 1
  km = metro(1)
  kt timeints
  ktrig = sc_trig(km, 0.5)
  printks "t=%f km=%f ktrig=%f\n", 0.01, kt, km, ktrig
endin

instr 2
  am = upsamp(metro(1))
  aenv = sc_trig(am, 0.5)
```

```
    asig pinker
    outch 1, asig*aenv
    outch 2, asig
endin

</CsInstruments>
<CsScore>
i 1 0 10
i 2 0 10

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*sc\_lag, metro, mpulse trigger*

## Crédits

Par : Eduardo Moguillansky 2017

# seed

seed — Fixe la valeur globale de la graine.

## Description

Fixe la valeur globale de la graine pour tous les *générateurs de bruit de classe x*, ainsi que pour d'autres opcodes qui utilisent un appel de random, tels que *grain*.



### Noter que

*rand*, *randh*, *randi*, *rnd(x)* et *birnd(x)* ne sont pas affectés par *seed*.

## Syntaxe

```
seed ival
```

## Exécution

Avec l'utilisation de *seed* on obtiendra des résultats prévisibles d'un orchestre utilisant des générateurs de nombres aléatoires, lors de plusieurs exécutions.

Lors de la spécification d'une valeur de graine, *ival* doit être un entier compris entre 0 et  $2^{32}$ . Si *ival* = 0, la valeur de *ival* sera dérivée de l'horloge du système.

## Exemples

Voici un exemple de l'opcode seed. Il utilise le fichier *seed.csd* [examples/seed.csd].

### Exemple 883. Exemple de l'opcode seed.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o seed.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;same values every time

seed 10
krnd randomh 100, 200, 5
printk .5, krnd ; look
```

```

aout oscili 0.8, 440+krnd, 1 ; & listen
outs aout, aout

endin

instr 2 ;different values every time - value is derived from system clock

seed 0 ; seed from system clock
krnd randomh 100, 200, 5
printk .5, krnd ; look
aout oscili 0.8, 440+krnd, 1 ; & listen
outs aout, aout

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave.

i 1 0 1
i 2 2 1
e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

i 1 time 0.00067: 100.00000
i 1 time 0.50000: 175.78677
i 1 time 1.00000: 170.89579

WARNING: Seeding from current time 834128659

i 2 time 2.00067: 100.00000
i 2 time 2.50000: 197.58517
i 2 time 3.00000: 188.69525

```

# sekere

sekere — Modèle semi-physique d'un son de chekeré.

## Description

*sekere* est un modèle semi-physique d'un son de chekeré. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntaxe

```
ares sekere iamp, idettack [, inum] [, idamp] [, imaxshake]
```

## Initialisation

*iamp* -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 64.

*idamp* (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

$$\text{damping\_amount} = 0,998 + (\text{idamp} * 0,002)$$

La valeur par défaut de *damping\_amount* est 0,999 ce qui signifie que la valeur par défaut de *idamp* est 0,5. Le maximum de *damping\_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 1,0.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

*imaxshake* (facultatif) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

## Exemples

Voici un exemple de l'opcode sekere. Il utilise le fichier *sekere.csd* [examples/sekere.csd].

### Exemple 884. Exemple de l'opcode sekere.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
```



```
; -o sekere.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
odbfs = 1

instr 1

idamp = p4 ;vary damping amount
asig sekere 1, 0.01, 64, idamp
outs asig, asig

endin
</CsInstruments>
<CsScore>

i1 0 1 .1
i1 + 1 .9
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*cabasa, crunch, sandpaper, stix*

## Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)

Adapté par John ffitich

Université de Bath, Codemist Ltd.

Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

# select

select — Choisit un échantillon sur la base de comparaisons au taux audio.

## Description

Choisit un échantillon parmi trois valeurs sur la base de comparaisons au taux audio.

## Syntaxe

```
aout select a1, a2, aless, aequal, amore
```

## Exécution

*a1, a2* -- signaux audio qui sont comparés.

*aless* -- signal audio choisi si  $a1[n] < a2[n]$

*aequal* -- signal audio choisi si  $a1[n] = a2[n]$

*asig* -- signal audio choisi si  $a1[n] > a2[n]$

La comparaison est faite échantillon par échantillon.

## Exemples

Voici un exemple de l'opcode select. Il utilise le fichier *select.csd* [examples/select.csd].

### Exemple 885. Exemple de l'opcode select.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>

<CsInstruments>
nchnls = 3
0dbfs = 1

instr 1
  a1 oscil 0.8, 440
  a2 oscil 0.6, 880
  az = 0
  alow line -1, p3, -1
  ahigh line 1, p3, 1
  aout select a1,a2,alow,az,ahigh
  outch 1, a1, 2,a2, 3,aout
endin
</CsInstruments>

<CsScore>
i1 0 5
e
</CsScore>
```

`</CsoundSynthesizer>`

## Voir aussi

*deltap*

## Crédits

Par : John ffitch 2016

Nouveau dans la version 6.09.

# semitone

semitone — Calcule un facteur pour élever/abaisser une fréquence d'un certain nombre de demi-tons.

## Description

Calcule un facteur pour élever/abaisser une fréquence d'un certain nombre de demi-tons.

## Syntaxe

`semitone(x)`

Cette fonction travaille aux taux-i, -k et -a.

## Initialisation

*x* -- une valeur exprimée en demi-tons.

## Exécution

La valeur retournée par la fonction *semitone* est un facteur. On peut multiplier une fréquence par ce facteur pour l'élever/l'abaisser du nombre de demi-tons spécifié.

## Exemples

Voici un exemple de l'opcode demi-ton. Il utilise le fichier *semitone.csd* [examples/semitone.csd].

### Exemple 886. Exemple de l'opcode demi-ton.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o semitone.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

iroot = 440 ; root note is A above middle-C (440 Hz)
ksem lfo 12, .5, 5 ; generate sawtooth, go from 5 octaves higher to root
ksm = int(ksem) ; produce only whole numbers
kfactor = semitone(ksm) ; for semitones
knew = iroot * kfactor
```

```

printk2 knew
printk2 kfactor
asig pluck 1, knew, 1000, 0, 1
asig dcblock asig ;remove DC
      outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 5
e

</CsScore>
</CsoundSynthesizer>

```

La sortie contiendra des lignes comme celles-ci :

```

i1 880.00000
i1 2.00000
i1 830.65625
i1 1.88786
i1 783.94665
i1 1.78170
i1 739.98885
i1 1.68179
i1 698.49586
i1 1.58749
i1 659.21793
i1 1.49822
i1 622.25397
i1 1.41421
i1 587.36267
i1 1.33492
i1 554.33399
i1 1.25985
i1 523.25113
i1 1.18921
i1 493.91116
i1 1.12253
i1 466.13747
i1 1.05940
i1 440.00000
i1 1.00000
.....

```

## Voir aussi

*cent, db, octave*

## Crédits

Nouveau dans la version 4.16

## sense

sense — Identique à l'opcode *sensekey*.

## Description

Voir l'opcode *sensekey*.

# sensekey

sensekey — Retourne le code ASCII d'une touche enfoncée.

## Description

Retourne le code ASCII d'une touche enfoncée ou -1 si aucune touche n'a été enfoncée.

## Syntaxe

```
kres[, kkeydown] sensekey
```

## Exécution

*kres* - retourne la valeur ASCII d'une touche qui a été enfoncée ou relachée.

*kkeydown* - retourne 1 si la touche a été enfoncée, 0 si elle a été relachée ou s'il n'y a pas d'évènement de touche.

On peut utiliser *kres* pour lire les évènements clavier de stdin. Il retourne la valeur ASCII de toute touche qui a été enfoncée ou relachée, ou -1 s'il n'y a eu aucune activité clavier. La valeur de *kkeydown* est 1 si une touche a été enfoncée, 0 sinon. Ce comportement est suivi par défaut, si bien qu'un relachement de touche est généré immédiatement après chaque pression de touche. Pour une fonctionnalité complète, on peut utiliser FLTK pour capturer les évènements clavier. *FLpanel* peut être utilisé pour capturer les évènements clavier et les envoyer à l'opcode *sensekey* en ajoutant un argument supplémentaire facultatif. Voir *FLpanel* pour plus d'information.



### Note

Cet opcode peut également s'écrire *sense*.

## Exemples

Voici un exemple de l'opcode *sensekey*. Il utilise le fichier *sensekey.csd* [exemples/sensekey.csd].

### Exemple 887. Exemple de l'opcode *sensekey*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o sensekey.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
```

```
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  k1 sensekey
  printk2 k1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for thirty seconds.
i 1 0 30
e

</CsScore>
</CsoundSynthesizer>
```

Voici à quoi devrait ressembler la sortie si la touche "q" est enfoncée...

```
q i1 113.00000
```

Voici un exemple de l'opcode sensekey en conjonction avec *FLpanel*. Il utilise le fichier *FLpanel-sensekey.csd* [exemples/FLpanel-sensekey.csd].

### Exemple 888. Exemple de l'opcode sensekey utilisant la capture clavier depuis un FLpanel.

```
<CsoundSynthesizer>

<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLpanel-sensekey.wav -W ;; for file output any platform
</CsOptions>

<CsInstruments>
; Example by Johnathan Murphy

sr      = 44100
ksmps   = 128
nchnls  = 2

; ikbdcapture flag set to 1
ikey     init      1

      FLpanel  "sensekey", 740, 340, 100, 250, 2, ikey
gkasc, giasc FLbutBank 2, 16, 8, 700, 300, 20, 20, -1
      FLpanelEnd
      FLrun

instr 1

kkey     sensekey
kprint   changed  kkey
      FLsetVal  kprint, kkey, giasc
```



```

    endin

</CsInstruments>

<CsScore>
i1 0 60
e
</CsScore>

</CsoundSynthesizer>

```

Le bouton allumé dans la fenêtre *FLpanel* montre la dernière touche enfoncée.

Voici une exemple plus complexe de l'opcode *sensekey* en conjonction avec *FLpanel*. Il utilise le fichier *FLpanel-sensekey2.csd* [exemples/FLpanel-sensekey.csd].

### Exemple 889. Exemple de l'opcode *sensekey* utilisant la capture clavier depuis un *FLpanel*.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac           ; -iadc         -d          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o FLpanel-sensekey2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 48000
ksmps = 32
nchnls = 1
; Example by Istvan Varga
; if the FLTK opcodes are commented out, sensekey will read keyboard
; events from stdin
    FLpanel "", 150, 50, 100, 100, 0, 1
    FLlabel 18, 10, 1, 0, 0, 0
    FLgroup "Keyboard Input", 150, 50, 0, 0, 0
    FLgroupEnd
    FLpanelEnd

    FLrun

    instr 1

ktrig1 init 1
ktrig2 init 1
nxtKey1:
k1, k2 sensekey
    if (k1 != -1 || k2 != 0) then
        printf "Key code = %02X, state = %d\n", ktrig1, k1, k2
    ktrig1 = 3 - ktrig1
    kgoto nxtKey1
    endif
nxtKey2:
k3 sensekey
    if (k3 != -1) then
        printf "Character = '%c'\n", ktrig2, k3
    ktrig2 = 3 - ktrig2
    kgoto nxtKey2
    endif

    endin

</CsInstruments>

```

```
<CsScore>  
i 1 0 3600  
e  
</CsScore>  
</CsoundSynthesizer>
```

La sortie console ressemblera à ceci :

```
new alloc for instr 1:  
Key code = 65, state = 1  
Character = 'e'  
Key code = 65, state = 0  
Key code = 72, state = 1  
Character = 'r'  
Key code = 72, state = 0  
Key code = 61, state = 1  
Character = 'a'  
Key code = 61, state = 0
```

## Voir aussi

*FLpanel*, *FLkeyIn*

## Crédits

Auteur : John ffitch  
Université de Bath, Codemist. Ltd.  
Bath, UK  
Octobre 2000

Exemples écrits par Kevin Conder, Johnathan Murphy et Istvan Varga.

Nouveau dans la version 4.09 de Csound. Renommé dans la version 4.10 de Csound.

# serialBegin

serialBegin — Ouvre un port série.

## Description

Opcode du greffon serial.

Ouvre un port série pour arduino.

## Syntaxe

```
iPort serialBegin SPortName [, ibaudRate]
```

## Initialisation

*SPortName* -- nom du port

*ibaudrate* -- vitesse de transmission, 9600 bauds par défaut.

## Exemples

Voici un exemple de l'opcode serialBegin. Il utilise le fichier *serialBegin.csd* [examples/serialBegin.csd].

### Exemple 890. Exemple de l'opcode serialBegin.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n      ;;no output
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o serialBegin.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 500 ; the default krate can be too fast for the arduino to handle
nchnls  = 2
0dbfs   = 1

instr 1

iPort serialBegin "/dev/cu.usbmodemfa131", 9600    ;connect to the arduino with baudrate = 9600

kGain init 16      ;read our knob value
kVal serialRead iPort
if (kVal != -1) then
    kGain = kVal/128
endif

aSig in           ;get our audio input and get its rms
kRms rms aSig*kGain
```

```

kRms = kRms*kRms*255      ;scale the rms to a good value for the LED and send it out
serialWrite iPort, (kRms < 255 ? kRms : 255)    ;must be in range: 0-255

endin
</CsInstruments>
<CsScore>
f 1 0 1024 10 1 1 1 1 1 1

i 1 0 200
e
</CsScore>
</CsoundSynthesizer>

```

Le code Arduino correspondant est :

```

void setup() {
  // active la communication serie
  Serial.begin(9600);

  // declare la pin 9 comme sortie :
  pinMode(9, OUTPUT);
}

void loop() {
  // n'agir que si l'on recoit quelque chose (doit être au taux-k de csound)
  if (Serial.available()) {
    // fixe la brillance de la LED (connectée à la pin 9) a notre valeur d'entree
    int brightness = Serial.read();
    analogWrite(9, brightness);

    // recupere la valeur du bouton rotatif et l'envoyer à csound
    int sensorValue = analogRead(A0);
    Serial.write(sensorValue/4); // ramene dans l'intervalle d'un octet (0-255)
  }
}
.....

```

## Voir aussi

serialEnd, serialWrite\_i, serialWrite, serialRead, serialPrint, serialFlush.

## Crédits

Auteur : Matt Ingalls  
2011

Nouveau dans la version 5.14

# serialEnd

serialEnd — Ferme un port série.

## Description

Opcodes du greffon serial.

Ferme un port série pour arduino.

## Syntaxe

```
serialEnd iPort
```

## Initialisation

*iPort* -- numéro du port obtenu d'un opcode *serialBegin*.

## Voir aussi

serialBegin, serialWrite\_i, serialWrite, serialRead, serialPrint, serialFlush.

## Crédits

Auteur : Matt Ingalls  
2011

Nouveau dans la version 5.14

# serialFlush

serialFlush — Vide les données d'un port série.

## Description

Opcodes du greffon serial.

Vide vers l'écran tous les octets (jusqu'à un maximum de 32K) contenus dans le tampon d'entrée. Noter que ces octets sont effacés du tampon. Utiliser cet opcode principalement pour des messages de débogage. Si l'on veut mélanger des messages de débogage avec d'autres messages de communication sur le même port, il faudra parcourir manuellement les données avec l'opcode *serialRead*.

## Syntaxe

```
serialFlush iPort
```

## Exécution

*iPort* -- numéro du port obtenu d'un opcode *serialBegin*.

## Voir aussi

serialBegin, serialEnd, serialWrite\_i, serialWrite, serialRead, serialPrint.

## Crédits

Auteur : Matt Ingalls  
2011

Nouveau dans la version 5.14

# serialPrint

serialPrint — Affiche les données d'un port série.

## Description

Opcodes du greffon serial.

Affiche à l'écran tous les octets (jusqu'à un maximum de 32K) contenus dans le tampon d'entrée. Noter que ces octets sont effacés du tampon. Utiliser cet opcode principalement pour des messages de débogage. Si l'on veut mélanger des messages de débogage avec d'autres messages de communication sur le même port, il faudra parcourir manuellement les données avec l'opcode *serialRead*.

## Syntaxe

```
serialPrint iPort
```

## Exécution

*iPort* -- numéro du port obtenu d'un opcode *serialBegin*.

## Voir aussi

serialBegin, serialEnd, serialWrite\_i, serialWrite, serialRead, serialFlush.

## Crédits

Auteur : Matt Ingalls  
2011

Nouveau dans la version 5.14

# serialRead

serialRead — Lit des donnée depuis un port série.

## Description

Opcode du greffon serial.

Lit des donnée depuis un port série pour arduino.

## Syntaxe

```
kByte serialRead iPort
```

## Exécution

*iPort* -- numéro du port obtenu d'un opcode *serialBegin*.

*kByte* -- octet de donnée lu.

## Exemples

Voici un exemple de l'opcode serialRead. Il utilise le fichier *serialRead.csd* [examples/serialRead.csd].

### Exemple 891. Exemple de l'opcode serialRead.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n      ;;no output
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o serialRead.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 500 ; the default krate can be too fast for the arduino to handle
nchnls  = 2
0dbfs   = 1

instr 1

iPort serialBegin "/dev/cu.usbmodemfa131", 9600    ;connect to the arduino with baudrate = 9600

kGain init 16      ;read our knob value
kVal serialRead iPort
if (kVal != -1) then
    kGain = kVal/128
endif

aSig in           ;get our audio input and get its rms
kRms rms aSig*kGain
```



```
kRms = kRms*kRms*255      ;scale the rms to a good value for the LED and send it out
serialWrite iPort, (kRms < 255 ? kRms : 255)    ;must be in range: 0-255

endin
</CsInstruments>
<CsScore>
f 1 0 1024 10 1 1 1 1 1 1

i 1 0 200
e
</CsScore>
</CsoundSynthesizer>
```

Le code Arduino correspondant est :

```
void setup() {
  // active la communication serie
  Serial.begin(9600);

  // declare la pin 9 comme sortie :
  pinMode(9, OUTPUT);
}

void loop() {
  // n'agir que si l'on recoit quelque chose (doit être au taux-k de csound)
  if (Serial.available()) {
    // fixe la brillance de la LED (connectée à la pin 9) a notre valeur d'entree
    int brightness = Serial.read();
    analogWrite(9, brightness);

    // recupere la valeur du bouton rotatif et l'envoyer à csound
    int sensorValue = analogRead(A0);
    Serial.write(sensorValue/4); // ramene dans l'intervalle d'un octet (0-255)
  }
}
.....
```

## Voir aussi

serialBegin, serialEnd, serialWrite\_i, serialWrite, serialPrint, serialFlush.

## Crédits

Auteur : Matt Ingalls  
2011

Nouveau dans la version 5.14

# serialWrite\_i

`serialWrite_i` — Ecrit des données sur un port série.

## Description

Opcodes du greffon serial.

Ecrit des données sur un port série pour arduino.

## Syntaxe

```
serialWrite_i iPort, iByte  
serialWrite_i iPort, SBytes
```

## Initialisation

*iPort* -- numéro du port obtenu d'un opcode *serialBegin*.

*iByte* -- octet à écrire.

## Voir aussi

`serialBegin`, `serialEnd`, `serialWrite`, `serialRead`, `serialPrint`, `serialFlush`.

## Crédits

Auteur : Matt Ingalls  
2011

Nouveau dans la version 5.14

# serialWrite

serialWrite — Ecrit des données sur un port série.

## Description

Opcodes du greffon serial.

Ecrit des données sur un port série pour arduino.

## Syntaxe

```
serialWrite iPort, iByte  
serialWrite iPort, kByte  
serialWrite iPort, SBytes
```

## Exécution

*iPort* -- numéro du port obtenu d'un opcode *serialBegin*.

*iByte* -- octet à écrire.

## Exemples

Voici un exemple de l'opcode serialWrite. Il utilise le fichier *serialWrite.csd* [examples/serialWrite.csd].

### Exemple 892. Exemple de l'opcode serialWrite.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-n   ;;no output  
;-iadc    ;;uncomment -iadc if realtime audio input is needed too  
; For Non-realtime ouput leave only the line below:  
; -o serialWrite.wav -W   ;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
sr      = 44100  
ksmps   = 500 ; the default krate can be too fast for the arduino to handle  
nchnls  = 2  
0dbfs   = 1  
  
instr 1  
  
iPort   serialBegin "/dev/cu.usbmodemfa131", 9600    ;connect to the arduino with baudrate = 9600  
  
kGain   init 16           ;read our knob value  
kVal    serialRead iPort  
if (kVal != -1) then  
    kGain = kVal/128
```

```
endif

aSig in          ;get our audio input and get its rms
kRms rms aSig*kGain

kRms = kRms*kRms*255      ;scale the rms to a good value for the LED and send it out
serialWrite iPort, (kRms < 255 ? kRms : 255) ;must be in range: 0-255

endin
</CsInstruments>
<CsScore>
f 1 0 1024 10 1 1 1 1 1 1

i 1 0 200
e
</CsScore>
</CsoundSynthesizer>
```

Le code Arduino correspondant est :

```
void setup() {
  // active la communication serie
  Serial.begin(9600);

  // declare la pin 9 comme sortie :
  pinMode(9, OUTPUT);
}

void loop() {
  // n'agir que si l'on recoit quelque chose (doit être au taux-k de csound)
  if (Serial.available()) {
    // fixe la brillance de la LED (connectée à la pin 9) a notre valeur d'entree
    int brightness = Serial.read();
    analogWrite(9, brightness);

    // recupere la valeur du bouton rotatif et l'envoyer à csound
    int sensorValue = analogRead(A0);
    Serial.write(sensorValue/4); // ramene dans l'intervalle d'un octet (0-255)
  }
}
.....
```

## Voir aussi

serialBegin, serialEnd, serialWrite\_i, serialRead, serialPrint, serialFlush.

## Crédits

Auteur : Matt Ingalls  
2011

Nouveau dans la version 5.14

# seqtime2

seqtime2 — Génère un signal de déclenchement suivant les valeurs stockées dans une table.

## Description

Génère un signal de déclenchement suivant les valeurs stockées dans une table.

## Syntaxe

```
ktrig_out seqtime2 ktrig_in, ktime_unit, kstart, kloop, kinitndx, kfn_times
```

## Exécution

*ktrig\_out* -- signal de déclenchement en sortie.

*ktime\_unit* -- unité de mesure du temps, par rapport aux secondes.

*ktrig\_in* -- signal de déclenchement en entrée.

*kstart* -- indice du début de la section en boucle.

*kloop* -- indice de la fin de la section en boucle.

*kinitndx* -- indice initial.



### Note

Bien que *kinitndx* soit renseigné au taux-k, l'accès ne s'y fait qu'au taux d'initialisation. Ainsi, si l'on utilise un argument de taux-k, son affectation doit se faire avec *init*.

*kfn\_times* -- numéro de la table contenant une suite de dates.

Cet opcode traite des suites temporelles de groupes de valeurs stockées dans une table.

*seqtime2* génère un signal déclencheur (une suite d'impulsions, voir aussi l'opcode *trigger*), en fonction des valeurs stockées dans la table *kfn\_times*. Cette table doit contenir une suite d'intervalles de temps (c-à-d des durées entre des événements adjacents). Les unités temporelles stockées dans la table sont exprimées en secondes, mais peuvent être changées d'échelle grâce à l'argument *ktime\_unit*. La table peut être remplie avec *GEN02* ou au moyen d'un fichier texte externe contenant des nombres, avec *GEN23*.

Il est possible de démarrer la séquence depuis une valeur différente de la première, en affectant à *kinitndx* un indice différent de zéro (qui correspond à la première valeur de la table). Normalement la séquence est bouclée, et le début et la fin de la boucle peuvent être ajustés en modifiant les arguments *kstart* et *kloop*. L'utilisateur doit s'assurer que les valeurs de ces arguments (ainsi que celle de *kinitndx*) correspondent à des indices de table valides, sinon Csound plantera (car il n'y a aucun test sur ces indices).

Il est possible de désactiver la boucle (mode à une passe) en affectant la même valeur aux arguments *kstart* et *kloop*. Dans ce cas, le dernier élément lu sera celui correspondant à la valeur de ces arguments. La table peut être lue à l'envers en affectant une valeur négative à *kloop*. Il est possible de déclencher deux événements presque en même temps (actuellement séparés d'un k-cycle) en donnant la valeur zéro à l'intervalle de temps correspondant. Si l'utilisateur désire envoyer une impulsion de déclenchement, le premier élément

*seqtime2* est semblable à *seqtime*, la différence étant que lorsque *ktrig\_in* contient une valeur différente de zéro, l'indice courant est réinitialisé à la valeur de *kinitndx*. *kinitndx* peut varier pendant l'exécution.

### Exemple 893. Exemple de l'opcode seqtime2.

```
;f0 3600  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*GEN02, GEN23, seqtime, trigseq, timedseq*

## Crédits

Auteur : Gabriel Maldonado

# seqtime

seqtime — Génère un signal de déclenchement suivant les valeurs stockées dans une table.

## Description

Génère un signal de déclenchement suivant les valeurs stockées dans une table.

## Syntaxe

```
ktrig_out seqtime ktime_unit, kstart, kloop, kinitndx, kfn_times
```

## Exécution

*ktrig\_out* -- signal de déclenchement en sortie.

*ktime\_unit* -- unité de mesure du temps, par rapport aux secondes.

*kstart* -- indice du début de la section en boucle.

*kloop* -- indice de la fin de la section en boucle.

*kinitndx* -- indice initial.



### Note

Bien que *kinitndx* soit renseigné au taux-k, l'accès ne s'y fait qu'au taux d'initialisation. Ainsi, si l'on utilise un argument de taux-k, son affectation doit se faire avec *init*.

*kfn\_times* -- numéro de la table contenant une suite de dates.

Cet opcode traite des suites temporelles de groupes de valeurs stockées dans une table.

*seqtime* génère un signal déclencheur (une suite d'impulsions, voir aussi l'opcode *trigger*), en fonction des valeurs stockées dans la table *kfn\_times*. Cette table doit contenir une suite d'intervalles de temps (c-à-d des durées entre des événements adjacents). Les unités temporelles stockées dans la table sont exprimées en secondes, mais peuvent être changées d'échelle grâce à l'argument *ktime\_unit*. La table peut être remplie avec *GEN02* ou au moyen d'un fichier texte externe contenant des nombres, avec *GEN23*.



### Note

Noter que l'indice *kloop* marque la limite de la boucle et n'est PAS inclus dans les éléments de la boucle. Si l'on veut boucler sur les quatre premiers éléments, il faut fixer *kstart* à 0 et *kloop* à 4.

Il est possible de démarrer la séquence depuis une valeur différente de la première, en affectant à *kinitndx* un indice différent de zéro (qui correspond à la première valeur de la table). Normalement la séquence est bouclée, et le début et la fin de la boucle peuvent être ajustés en modifiant les arguments *kstart* et *kloop*. L'utilisateur doit s'assurer que les valeurs de ces arguments (ainsi que celle de *kinitndx*) correspondent à des indices de table valides, sinon Csound plantera (car il n'y a aucun test sur ces indices).

Il est possible de désactiver la boucle (mode à une passe) en affectant la même valeur aux arguments *kstart* et *kloop*. Dans ce cas, le dernier élément lu sera celui correspondant à la valeur de ces arguments. La table



peut être lue à l'envers en affectant une valeur négative à *kloop*. Il est possible de déclencher deux événements presque en même temps (actuellement séparés d'un k-cycle) en donnant la valeur zéro à l'intervalle de temps correspondant. Si l'utilisateur désire envoyer une impulsion de déclenchement, le premier élément de la table doit valoir zéro. L'évènement doit être déclenché sur un instrument de l'orchestre venant immédiatement après l'instrument contenant l'opcode *seqtime*.

## Exemples

Voici un exemple de l'opcode *seqtime*. Il utilise le fichier *seqtime.csd* [examples/seqtime.csd].

### Exemple 894. Exemple de l'opcode *seqtime*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>

<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o seqtime.wav -W ;; for file output any platform
</CsOptions>

<CsInstruments>

sr = 44100
ksmps = 64
nchnls = 1

; By Tim Mortimer and Andres Cabrera 2007

0dbfs = 1

gisine      ftgen      0, 0, 8192, 10,      1
;;; table defining an integer pitch set
gipset      ftgen      0, 0, 4, -2, 8.00, 8.04, 8.07, 8.10
;;;DELTA times for seqtime
gidelta     ftgen      0, 0, 4, -2, .5, 1, .25, 1.25

instr 1

kndx init 0
ktrigger init 0

ktime_unit init 1
kstart init p4
kloop init p5
kinitndx init 0
kfn_times init gidelta

ktrigger seqtime ktime_unit, kstart, kloop, kinitndx, kfn_times

printk2 ktrigger

if (ktrigger > 0) then
  kpitch table kndx, gipset
  event "i", 2, 0, 1, kpitch
  kndx = kndx + 1
  kndx = kndx % kloop
```

```

endif

    endin

    instr 2
    icps = cspch (p4)
    a1 buzz 1, icps, 7, gisine
    aamp expseg 0.00003, .02, 1, p3-.02, 0.00003

    a1 = a1 * aamp * 0.5

    out a1
    endin

</CsInstruments>

<CsScore>
;      start      dur      kstart      kloop
i 1 0 7 0 4
i 1 8 10 0 3
i 1 19 10 4 4

</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

*GEN02, GEN23, trigseq seqtime2*

## Crédits

Auteur : Gabriel Maldonado

Novembre 2002. Note sur le paramètre *kinitndx* ajoutée grâce à Rasmus Ekman.

Nouveau dans la version 4.06

Exemple par Tim Mortimer et Andrés Cabrera, 2007

# setcol

setcol — Remplit une colonne donnée d'un tableau bidimensionnel à partir d'un vecteur.

## Description

Remplit une colonne donnée d'un tableau bidimensionnel. La sortie est un tableau bidimensionnel avec le contenu de la colonne concernée égal aux valeurs du tableau d'entrée (unidimensionnel ; si ce dernier est bidimensionnel, sa première ligne est utilisée).

## Syntaxe

```
i/kout[] setcol i/kin[],i/kcol
```

## Initialisation

*iout[]* -- tableau de sortie contenant la colonne fixée. Créé s'il n'existe pas.

*iin[]* -- tableau d'entrée.

*icol* -- colonne à fixer.

## Exécution

*kout[]* -- tableau de sortie contenant la colonne fixée. Créé s'il n'existe pas.

*kin[]* -- tableau d'entrée.

*kcol* -- colonne à fixer.

## Exemples

Voici un exemple de l'opcode setcol. Il utilise le fichier *setcol.csd* [examples/setcol.csd].

### Exemple 895. Exemple de l'opcode setcol.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
instr 1
  kcnt init 0
  kArr[] init 3,3
  kVec[] fillarray 0,1,2
  while kcnt < 3 do
    kArr setcol kVec,kcnt
    printf "column %d: %d %d %d\n",kcnt+1,kcnt,kArr[0][kcnt],kArr[1][kcnt],kArr[2][kcnt]
    kcnt += 1
  od
endin
```

```
</CsInstruments>  
<CsScore>  
i1 0 0.1  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux.

## Crédits

Auteur : Victor Lazzarini  
NUI Maynooth  
2014

Nouveau dans la version 6.04

# setctrl

setctrl — Contrôleurs réglettes configurables pour une utilisation en temps réel.

## Description

Opcode du greffon control.

Contrôleurs réglettes configurables pour une utilisation en temps réel. Nécessite Winsound ou TCL/TK. *setctrl* donne à une réglette une valeur spécifique ou bien fixe sa valeur minimale ou maximale.

## Syntaxe

```
setctrl inum, ival, itype
```

## Initialisation

Noter que cet opcode n'est pas disponible sous Windows à cause de l'implémentation des tuyaux sur ce système.

*inum* -- numéro de la réglette à changer

*ival* -- valeur à envoyer à la réglette

*itype* -- type de la valeur envoyée à la réglette, comme suit :

- 1 -- fixe la valeur courante. La valeur initiale est 0.
- 2 -- fixe la valeur minimale. 0 par défaut.
- 3 -- fixe la valeur maximale. 127 par défaut.
- 4 -- fixe l'étiquette. (Nouveau dans la version 4.09 de Csound)

## Exécution

L'appel de *setctrl* va créer une nouvelle réglette à l'écran. Il n'y a pas de limite théorique au nombre de réglettes. Winsound et TCL/TK n'utilisent que des entiers pour les valeurs de réglette, si bien qu'il peut être nécessaire de re-échelonner les valeurs. Parce que les interfaces graphiques passent habituellement leurs valeurs à une fréquence assez lente, il peut être sage de traiter la sortie du contrôleur avec *port*.

## Exemples

Voici un exemple de l'opcode setctrl. Il utilise le fichier *setctrl.csd* [examples/setctrl.csd].

### Exemple 896. Exemple de l'opcode setctrl.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
```

```

<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o setctrl.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Display the label "Volume" on Slider #1.
setctrl 1, "Volume", 4
; Set Slider #1's initial value to 20.
setctrl 1, 20, 1

; Capture and display the values for Slider #1.
k1 control 1
printk2 k1

; Play a simple oscillator.
; Use the values from Slider #1 for amplitude.
kamp = k1 * 128
a1 oscil kamp, 440, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for thirty seconds.
i 1 0 30
e

</CsScore>
</CsoundSynthesizer>

```

La sortie contiendra des lignes comme celles-ci :

```

i1      38.00000
i1      40.00000
i1      43.00000

```

## Voir aussi

*control*

## Crédits

Auteur : John ffitch  
 Université de Bath, Codemist. Ltd.  
 Bath, UK

Mai 2000

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.06 de Csound.

# setksmps

setksmps — Fixe la valeur locale de ksmpls dans un bloc d'opcode défini par l'utilisateur.

## Description

Fixe la valeur locale de *ksmps* dans un bloc d'opcode défini par l'utilisateur.

## Syntaxe

```
setksmps iksmps
```

## Initialisation

*iksmps* -- fixe la valeur locale de *ksmps*.

Si *iksmps* vaut zéro, le *ksmps* de l'instrument ou de l'opcode appelant est utilisé (c'est le comportement par défaut).



### Note

Le *ksmps* local est implémenté en divisant une période de contrôle en sous-périodes-k plus petites et en modifiant temporairement les variables globales internes de Csound. Ceci nécessite également de convertir le taux des entrées de taux-k et des arguments de sortie (les variables d'entrée reçoivent la même valeur durant toutes les sous-périodes-k, tandis que les sorties ne sont écrites que dans la dernière). Cela signifie aussi que l'on ne peut pas utiliser un *ksmps* local supérieur au *ksmps* global.



### Avertissement au sujet du *ksmps* local

Lorsque le *ksmps* local est différent de celui de l'orchestre (défini dans l'en-tête de l'orchestre), il ne faut pas utiliser d'opérations globales de taux-a dans le bloc d'opcode défini par l'utilisateur.

Ça comprend :

- tout accès aux variables « ga »
- les opcodes zak de taux-a (*zar*, *zaw*, etc.)
- *tablera* et *tablewa* (en fait, ces deux opcodes peuvent fonctionner, mais il faut prendre des précautions)
- La famille d'opcodes *in* et *out* (ceux-ci lisent et écrivent dans des tampons globaux de taux-a)

En général, le *ksmps* local doit être utilisé avec précaution car c'est un dispositif expérimental. Bien qu'il fonctionne dans la plupart des cas.

On peut utiliser l'instruction *setksmps* pour fixer la valeur locale de *ksmps* dans un bloc d'opcode défini par l'utilisateur. Il a un paramètre de taux-i définissant la nouvelle valeur de *ksmps* (qui reste inchangée si l'on utilise zéro). *setksmps* doit être utilisé avant tout autre opcode (mais on peut le mettre après *xin*), sinon il y aura des résultats imprévisibles.



## Exécution

La syntaxe d'un bloc d'opcode défini par l'utilisateur est la suivante :

```
opcode name, outtypes, intypes
xinarg1 [, xinarg2] [, xinarg3] ... [xinargN] xin
[setksmps iksmps]
... the rest of the instrument's code.
xout xoutarg1 [, xoutarg2] [, xoutarg3] ... [xoutargN]
endop
```

On peut alors utiliser le nouvel opcode avec la syntaxe usuelle :

```
[xinarg1] [, xinarg2] ... [xinargN] name [xoutarg1] [, xoutarg2] ... [xoutargN] [, iksmps]
```

## Exemples

Voir l'exemple de l'opcode *opcode*.

## Voir aussi

*endop, opcode, xin, xout*

## Crédits

Auteur : Istvan Varga, 2002 ; basé sur du code par Matt J. Ingalls

Nouveau dans la version 4.22

# setrow

setrow — Remplit une ligne donnée d'un tableau bidimensionnel à partir d'un vecteur.

## Description

Remplit une ligne donnée d'un tableau bidimensionnel. La sortie est un tableau bidimensionnel avec le contenu de la ligne concernée égal aux valeurs du tableau d'entrée (unidimensionnel ; si ce dernier est bidimensionnel, sa première ligne est utilisée).

## Syntaxe

```
i/kout[] setrowi/kin[],i/krow
```

## Initialisation

*iout[]* -- tableau de sortie contenant la ligne fixée. Créé s'il n'existe pas.

*iin[]* -- tableau d'entrée.

*irow* -- ligne à fixer.

## Exécution

*kout[]* -- tableau de sortie contenant la ligne fixée. Créé s'il n'existe pas.

*kin[]* -- tableau d'entrée.

*krow* -- ligne à fixer.

## Exemples

Voici un exemple de l'opcode setrow. Il utilise le fichier *rfft.csd* [examples/rfft.csd].

### Exemple 897. Exemple de l'opcode setrow.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>

<CsOptions>
-d -o dac
</CsOptions>

<CsInstruments>
;ksmps needs to be an integer div of hopsize
ksmps = 64
Odbfs=1

instr 1

    ihopsize = 256    ; hopsize
    ifftsize = 1024   ; FFT size
```

```

iolaps = ifftsize/ihopsize ; overlaps
ibw = sr/ifftsize ; bin bandwidth
kcnt init 0 ; counting vars
krow init 0

kOla[] init ifftsize ; overlap-add buffer
kIn[] init ifftsize ; input buffer
kSw[] init ifftsize
kOut[][] init iolaps, ifftsize ; output buffers

a1 diskin2 "fox.wav",1,0,1 ; audio input
ks expon 100, p3, 1000
asw vco2 0.15, ks

/* every hopsize samples */
if kcmt == ihopsize then
  /* window and take FFT */
  kWin[] window kIn,krow*ihopsize
  kSpec[] rfft kWin
  kWin window kSw,krow*ihopsize
  kSpec2[] rfft kWin
  kProd[] cmplxprod kSpec, kSpec2

  /* IFFT + window */
  kRow[] rfft kProd + kSpec
  kWin window kRow, krow*ihopsize
  /* place it on out buffer */
  kOut setrow kWin, krow

  /* zero the ola buffer */
  kOla = 0
  /* overlap-add */
  ki = 0
  until ki == iolaps do
    kRow getrow kOut, ki
    kOla = kOla + kRow
    ki += 1
  od

  /* update counters */
  krow = (krow+1)%iolaps
  kcmt = 0
endif

/* shift audio in/out of buffers */
kIn shiftin a1
kSw shiftin asw
a2 shiftout kOla
out a2/iolaps

/* increment counter */
kcmt += ksmpts

endin

</CsInstruments>

<CsScore>
i1 0 10
</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux.

## Crédits

Auteur : Victor Lazzarini  
NUI Maynooth  
2014

Nouveau dans la version 6.04

# setscorepos

setscorepos — Modifie la position de lecture de l'exécution courante de la partition.

## Description

Modifie la position de lecture de l'exécution courante de la partition.

## Syntaxe

```
setscorepos ipos
```

## Initialisation

*ipos* -- position de lecture en secondes.

## Exemples

Voici un exemple de l'opcode setscorepos. Il utilise le fichier *setscorepos.csd* [examples/setscorepos.csd].

### Exemple 898. Exemple de l'opcode setscorepos.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o setscorepos.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 8192, 10, 1

instr 1

asig poscil 0.5, p4, giSine ;play something
outs asig, asig
endin

instr 11

setscorepos 8.5
endin

</CsInstruments>
<CsScore>
```

```
i1 0 2 220 ;this one will be played
i11 2.5 1 ;start setscorepos now
i1 3 2 330 ;skip this note
i1 6 2 440 ;and this one
i1 9 2 550 ;play this one

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*rewindscore*

## Crédits

Auteur : Victor Lazzarini  
2008

Nouveau dans la version 5.09 de Csound.

# sfilist

`sfilist` — Imprime une liste de tous les instruments d'un fichier SoundFont2 (SF2) préalablement chargé.

## Description

Imprime une liste de tous les instruments d'un fichier SoundFont2 (SF2) de sons échantillonnés préalablement chargé. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## Syntaxe

```
sfilist ifilhandle
```

## Initialisation

`ifilhandle` -- nombre unique généré par l'opcode `sload` à utiliser comme identificateur pour un fichier SF2. On peut charger et activer plusieurs fichiers SF2 en même temps.

## Exécution

`sfilist` imprime sur la console une liste de tous les instruments d'un fichier SoundFont2 (SF2) préalablement chargé.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Exemples

Voici un exemple de l'opcode `sfilist`. Il utilise le fichier `sfilist.csd` [examples/sfilist.csd].

### Exemple 899. Exemple de l'opcode `sfilist`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0   ;;realtime audio out, virtual midi in
;-iadc   ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sfilist.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
```

```

0dbfs = 1

gisf sfload "sf_GMbank.sf2"
sfilist gisf ;list all instruments

instr 1 ; play from score and midi keyboard

mididefault 60, p3
midinoteonkey p4, p5
inum init p4
ivel init p5
ivel init ivel/127 ;make velocity dependent
kamp linsegr 1, 1, 1, .1, 0
kamp = kamp/3000 ;scale amplitude
kfreq init 1 ;do not change freq from sf
a1, a2 sfinstr3 ivel, inum, kamp*ivel, kfreq, 100, gisf ;choose Halo Pad
outs a1, a2

endin

</CsInstruments>
<CsScore>
f0 60 ; stay active for 1 minute

i1 0 1 60 127
i1 + 1 62 <
i1 + 1 65 <
i1 + 1 69 10

e
</CsScore>
</CsoundSynthesizer>

```

La sortie comprendra des lignes comme celles-ci :

```

Instrument list of "sf_GMbank.sf2"
0) Piano 1
1) Piano 2
2) CP 70
3) EP 1 layer 1
4) EP 1 layer 2
5) E.Piano 2
6) Harpsichord
.....
100) Halo Pad
.....

```

## Voir aussi

*sfinstr, sfinstrm, sfload, sfpassign, sfplay, sfplaym, sfplist, sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
 Italie  
 Mai 2000

Nouveau dans la version 4.07 de Csound



# **sfinstr3**

**sfinstr3** — Joue un instrument échantillonné SoundFont2 (SF2), produisant un son stéréo avec interpolation cubique.

## **Description**

Joue un instrument échantillonné SoundFont2 (SF2), produisant un son stéréo avec interpolation cubique. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## **Syntaxe**

```
ar1, ar2 sfinstr3 ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \  
[, iflag] [, ioffset]
```

## **Initialisation**

*ivel* -- vitesse.

*inotenum* -- numéro de note MIDI.

*instrnum* -- numéro d'un instrument d'un fichier SF2.

*ifilhandle* -- nombre unique généré par l'opcode *sload* à utiliser comme identificateur pour un fichier SF2. On peut charger et activer plusieurs fichiers SF2 en même temps.

*iflag* (facultatif) -- drapeau concernant le comportement de *xfreq* et de *inotenum*.

*ioffset* (facultatif) -- endroit où commence la lecture, en échantillons.

## **Exécution**

*xamp* -- facteur de correction de l'amplitude.

*xfreq* -- valeur de fréquence ou multiplicateur de fréquence, selon la valeur de *iflag*. Quand *iflag* = 0, *xfreq* est un multiplicateur de la fréquence par défaut, fixée par le preset SF2 à la valeur *inotenum*. Quand *iflag* = 1, *xfreq* est la fréquence absolue du son produit, en Hz. La valeur par défaut est 0.

Lorsque *iflag* = 0, *inotenum* fixe la fréquence de la sortie en fonction du numéro de note MIDI utilisé, et *xfreq* est utilisé comme un multiplicateur. Lorsque *iflag* = 1, la fréquence de la sortie est fixée directement par *xfreq*. Cela permet l'utilisation de n'importe quelle échelle micro-tonale. Cependant, cette méthode n'est conçue pour travailler correctement qu'avec des presets accordés selon le classique tempérament égal. L'utilisation de cette méthode avec un preset ayant déjà un accordage non standard ou bien avec des presets de drum-kit donnera des résultats imprévisibles.

L'amplitude peut être ajustée en variant l'argument *xamp* qui agit comme un multiplicateur.

Le paramètre *ioffset* permet de commencer la lecture depuis un autre échantillon que le premier. L'utilisateur doit s'assurer que sa valeur est inférieure à la longueur du son. Sinon, il y a un risque de plantage de Csound.

*sfinstr3* est une version avec interpolation cubique de *sfinstr*. La différence de qualité sonore est notable, particulièrement avec les échantillons transposés dans le grave. Pour les échantillons transposés dans l'aigu, la différence est moins appréciable et je suggère d'utiliser les versions avec interpolation linéaire, car elles sont plus rapides.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Exemples

Voici un exemple de l'opcode *sfinstr3*. Il utilise le fichier *sfinstr3.csd* [examples/sfinstr3.csd].

### Exemple 900. Exemple de l'opcode *sfinstr3*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0 ;;realtime audio out
;-iadc ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sfinstr3.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gi24 ftgen 1, 0, 32, -2, 24, 2, 261.626, 60, 1, 1.0293022, 1.059463, 1.0905076, 1.1224619, 1.1553525,
1.2240532, 1.2599207, 1.2968391, 1.33483924, 1.3739531, 1.414213, 1.4556525, 1.4983063, 1.
1.6339145, 1.6817917, 1.73107, 1.7817962, 1.8340067, 1.8877471, 1.9430623, 2 ;table for m

giSF sload "sf_GMbank.sf2"
sfilist giSF

instr 1

mididefault 60, p3
midinoteonkey p4, p5
ikey = p4
ivel = p5
aenv linsegr 1, 1, 1, 1, 0 ;envelope
icps cpstuni ikey, 1 ;24 tones per octave
iamp = 0.0002 ;scale amplitude
iamp = iamp * ivel * 1/128 ;make velocity-dependent
aL, aR sfinstr3 ivel, ikey, iamp, icps, 180, giSF, 1 ;= Slap Bass 3
aL = aL * aenv
aR = aR * aenv
outs aL, aR

endin
</CsInstruments>
<CsScore>
f0 60 ;play for 60 seconds

i1 0 1 60 100 1 ;using ftable 1
```

```
i1 + 1 62 < .  
i1 + 1 65 < .  
i1 + 1 69 40 .  
  
e  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*sfilist, sfinstr3m, sfinstrm, sfinstr, sfload, sfpassign, sfplay3, sfplay3m, sfplay, sfplaym, sfplist, sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

# **sfinstr3m**

**sfinstr3m** — Joue un instrument échantillonné SoundFont2 (SF2), produisant un son mono avec interpolation cubique.

## **Description**

Joue un instrument échantillonné SoundFont2 (SF2), produisant un son mono avec interpolation cubique. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## **Syntaxe**

```
ares sfinstr3m ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \  
    [, iflag] [, ioffset]
```

## **Initialisation**

*ivel* -- vitesse.

*inotenum* -- numéro de note MIDI.

*instrnum* -- numéro d'un instrument d'un fichier SF2.

*ifilhandle* -- nombre unique généré par l'opcode *sload* à utiliser comme identificateur pour un fichier SF2. On peut charger et activer plusieurs fichiers SF2 en même temps.

*iflag* (facultatif) -- drapeau concernant le comportement de *xfreq* et de *inotenum*.

*ioffset* (facultatif) -- endroit où commence la lecture, en échantillons.

## **Exécution**

*xamp* -- facteur de correction de l'amplitude.

*xfreq* -- valeur de fréquence ou multiplicateur de fréquence, selon la valeur de *iflag*. Quand *iflag* = 0, *xfreq* est un multiplicateur de la fréquence par défaut, fixée par le preset SF2 à la valeur *inotenum*. Quand *iflag* = 1, *xfreq* est la fréquence absolue du son produit, en Hz. La valeur par défaut est 0.

Lorsque *iflag* = 0, *inotenum* fixe la fréquence de la sortie en fonction du numéro de note MIDI utilisé, et *xfreq* est utilisé comme un multiplicateur. Lorsque *iflag* = 1, la fréquence de la sortie est fixée directement par *xfreq*. Cela permet l'utilisation de n'importe quelle échelle micro-tonale. Cependant, cette méthode n'est conçue pour travailler correctement qu'avec des presets accordés selon le classique tempérament égal. L'utilisation de cette méthode avec un preset ayant déjà un accordage non standard ou bien avec des presets de drum-kit donnera des résultats imprévisibles.

L'amplitude peut être ajustée en variant l'argument *xamp* qui agit comme un multiplicateur.

Le paramètre *ioffset* permet de commencer la lecture depuis un autre échantillon que le premier. L'utilisateur doit s'assurer que sa valeur est inférieure à la longueur du son. Sinon, il y a un risque de plantage de Csound.

*sfinstr3m* est une version avec interpolation cubique de *sfinstrm*. La différence de qualité sonore est notable, particulièrement avec les échantillons transposés dans le grave. Pour les échantillons transposés dans l'aigu, la différence est moins appréciable et je suggère d'utiliser les versions avec interpolation linéaire, car elles sont plus rapides.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Exemples

Voici un exemple de l'opcode *sfinstr3m*. Il utilise le fichier *sfinstr3m.csd* [examples/sfinstr3m.csd].

### Exemple 901. Exemple de l'opcode *sfinstr3m*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0 ;;realtime audio out
;-iadc ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sfinstr3m.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisf sfloat "07AcousticGuitar.sf2"
sfilist isf

instr 1 ; play from score and midi keyboard

mididefault 60, p3
midinoteonkey p4, p5
inum init p4
ivel init p5
ivel init ivel/127 ;make velocity dependent
kamp linsegr 1, 1, 1, .1, 0
kamp = kamp/7000 ;scale amplitude
kfreq init 1 ;do not change freq from sf
aout sfinstr3m ivel, inum, kamp*ivel, kfreq, 0, gisf
outs aout, aout

endin

</CsInstruments>
<CsScore>
f0 60 ; stay active for 1 minute

i1 0 1 60 127
i1 + 1 62 <
i1 + 1 65 <
i1 + 1 69 10
```

```
e  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*sfilist, sfinstr3, sfinstr, sfinstrm, sfload, sfpassign, sfplay3, sfplay3m, sfplay, sfplaym, sfplist, sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

# sfinstr

sfinstr — Joue un instrument échantillonné SoundFont2 (SF2), produisant un son stéréo.

## Description

Joue un instrument échantillonné SoundFont2 (SF2), produisant un son stéréo. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## Syntaxe

```
ar1, ar2 sfinstr ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \  
[, iflag] [, ioffset]
```

## Initialisation

*ivel* -- vitesse.

*inotenum* -- numéro de note MIDI.

*instrnum* -- numéro d'un instrument d'un fichier SF2.

*ifilhandle* -- nombre unique généré par l'opcode *sload* à utiliser comme identificateur pour un fichier SF2. On peut charger et activer plusieurs fichiers SF2 en même temps.

*iflag* (facultatif) -- drapeau concernant le comportement de *xfreq* et de *inotenum*.

*ioffset* (facultatif) -- endroit où commence la lecture, en échantillons.

## Exécution

*xamp* -- facteur de correction de l'amplitude.

*xfreq* -- valeur de fréquence ou multiplicateur de fréquence, selon la valeur de *iflag*. Quand *iflag* = 0, *xfreq* est un multiplicateur de la fréquence par défaut, fixée par le preset SF2 à la valeur *inotenum*. Quand *iflag* = 1, *xfreq* est la fréquence absolue du son produit, en Hz. La valeur par défaut est 0.

Lorsque *iflag* = 0, *inotenum* fixe la fréquence de la sortie en fonction du numéro de note MIDI utilisé, et *xfreq* est utilisé comme un multiplicateur. Lorsque *iflag* = 1, la fréquence de la sortie est fixée directement par *xfreq*. Cela permet l'utilisation de n'importe quelle échelle micro-tonale. Cependant, cette méthode n'est conçue pour travailler correctement qu'avec des presets accordés selon le classique tempérament égal. L'utilisation de cette méthode avec un preset ayant déjà un accordage non standard ou bien avec des presets de drum-kit donnera des résultats imprévisibles.

L'amplitude peut être ajustée en variant l'argument *xamp* qui agit comme un multiplicateur.

Le paramètre *ioffset* permet de commencer la lecture depuis un autre échantillon que le premier. L'utilisateur doit s'assurer que sa valeur est inférieure à la longueur du son. Sinon, il y a un risque de plantage de Csound.

*sfinstr* joue un instrument SF2 plutôt qu'un preset (un instrument SF2 est la base d'une couche de preset). *instrnum* indique le numéro de l'instrument, et l'utilisateur doit s'assurer que le numéro spécifié est celui d'un instrument existant d'une banque soundfont déterminée. Noter que *xamp* et *xfreq* peuvent opérer aussi bien au taux-k qu'au taux-a, mais les deux arguments doivent travailler au même taux.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Exemples

Voici un exemple de l'opcode *sfinstr*. Il utilise le fichier *sfinstr.csd* [exemples/sfinstr.csd].

### Exemple 902. Exemple de l'opcode *sfinstr*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0 ;;realtime audio out
;-iadc ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sfinstr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisf sfloat "sf_GMbank.sf2"
sfilist isf

instr 1 ; play from score and midi keyboard

mididefault 60, p3
midinoteonkey p4, p5
inum init p4
ivel init p5
ivel init ivel/127 ;make velocity dependent
kamp linsegr 1, 1, 1, .1, 0
kamp = kamp/5000 ;scale amplitude
kfreq init 1 ;do not change freq from sf
a1,a2 sfinstr ivel, inum, kamp*ivel, kfreq, 194, gisf ;= Strings 2 tighter
outs a1, a2

endin

</CsInstruments>
<CsScore>
f0 60 ; stay active for 1 minute

i1 0 1 60 127
i1 + 1 62 <
i1 + 1 65 <
i1 + 1 69 10
```



```
e  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*sfilist, sfinstrm, sfload, sfpassign, sfplay, sfplaym, sfplist, sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

# **sfinstrm**

**sfinstrm** — Joue un instrument échantillonné SoundFont2 (SF2), produisant un son mono.

## **Description**

Joue un instrument échantillonné SoundFont2 (SF2), produisant un son mono. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## **Syntaxe**

```
ares sfinstrm ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \  
    [, iflag] [, ioffset]
```

## **Initialisation**

*ivel* -- vitesse.

*inotenum* -- numéro de note MIDI.

*instrnum* -- numéro d'un instrument d'un fichier SF2.

*ifilhandle* -- nombre unique généré par l'opcode *sfload* à utiliser comme identificateur pour un fichier SF2. On peut charger et activer plusieurs fichiers SF2 en même temps.

*iflag* (facultatif) -- drapeau concernant le comportement de *xfreq* et de *inotenum*.

*ioffset* (facultatif) -- endroit où commence la lecture, en échantillons.

## **Exécution**

*xamp* -- facteur de correction de l'amplitude.

*xfreq* -- valeur de fréquence ou multiplicateur de fréquence, selon la valeur de *iflag*. Quand *iflag* = 0, *xfreq* est un multiplicateur de la fréquence par défaut, fixée par le preset SF2 à la valeur *inotenum*. Quand *iflag* = 1, *xfreq* est la fréquence absolue du son produit, en Hz. La valeur par défaut est 0.

Lorsque *iflag* = 0, *inotenum* fixe la fréquence de la sortie en fonction du numéro de note MIDI utilisé, et *xfreq* est utilisé comme un multiplicateur. Lorsque *iflag* = 1, la fréquence de la sortie est fixée directement par *xfreq*. Cela permet l'utilisation de n'importe quelle échelle micro-tonale. Cependant, cette méthode n'est conçue pour travailler correctement qu'avec des presets accordés selon le classique tempérament égal. L'utilisation de cette méthode avec un preset ayant déjà un accordage non standard ou bien avec des presets de drum-kit donnera des résultats imprévisibles.

L'amplitude peut être ajustée en variant l'argument *xamp* qui agit comme un multiplicateur.

Le paramètre *ioffset* permet de commencer la lecture depuis un autre échantillon que le premier. L'utilisateur doit s'assurer que sa valeur est inférieure à la longueur du son. Sinon, il y a un risque de plantage de Csound.

*sfinstrm* est une version mono de *sfinstr*. C'est l'opcode le plus rapide de la famille SF2.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Voir aussi

*sfilist, sfinstr, sfload, sfpassign, sfplay, sfplaym, sfplist, sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

# sfload

sfload — Charge en mémoire un fichier d'échantillons SoundFont2 (SF2) en entier.

## Description

Charge en mémoire un fichier d'échantillons SoundFont2 (SF2) en entier. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

*sfload* doit être placé dans la section d'en-tête d'un orchestre de Csound.

## Syntaxe

```
ir sfload "filename"
```

## Initialisation

*ir* -- valeur de retour à utiliser par les autres opcodes SF2. Pour *sfload*, *ir* est le *ifilhandle*.

« *filename* » -- nom du fichier SF2, avec son chemin complet. C'est une chaîne de caractères entre guillemets avec le « / » comme séparateur de répertoires (ceci s'applique également à DOS et à Windows, où l'utilisation d'un antislash générera une erreur), ou bien un entier qui a été lié à une chaîne de caractères par *strset*.

## Exécution

*sfload* charge en mémoire un fichier SF2 en entier. Il retourne un identificateur de fichier à utiliser par les autres opcodes. On peut placer plusieurs instances de *sfload* dans la section d'en-tête d'un orchestre, ce qui permet d'utiliser plusieurs fichiers SF2 dans un seul orchestre.

Si l'on tente de charger deux fois le même fichier, le fichier déjà chargé est utilisé accompagné d'un message d'avertissement (à partir de la version 6.14).

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

Il faut noter qu'avant la version 5.12 on pouvait charger au maximum dix soundfonts. Cette restriction est maintenant levée.

## Exemples

Voici un exemple de l'opcode *sfload*. Il utilise le fichier *sfload.csd* [examples/sfload.csd].

### Exemple 903. Exemple de l'opcode *sfload*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0    ;;realtime audio out, virtual midi in
;-iadc    ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sfload.wav -W    ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

;load two soundfonts
isf sfload "07AcousticGuitar.sf2"
ir sfload "01hpschd.sf2"
sfplist isf
sfplist ir
sfpassign 0, isf
sfpassign 1, ir

instr 1 ; play guitar from score and midi keyboard - preset index = 0

mididefault 60, p3
midinoteonkey p4, p5
inum init p4
ivel init p5
ivel init ivel/127    ;make velocity dependent
kamp linsegr 1, 1, 1, .1, 0
kamp = kamp/3000    ;scale amplitude
kfreq init 1    ;do not change freq from sf
a1,a2 sfplay3 ivel, inum, kamp*ivel, kfreq, 0    ;preset index = 0
outs a1, a2

endin

instr 2 ; play harpsichord from score and midi keyboard - preset index = 1

mididefault 60, p3
midinoteonkey p4, p5
inum init p4
ivel init p5
ivel init ivel/127    ;make velocity dependent
kamp linsegr 1, 1, 1, .1, 0
kamp = kamp/1000    ;scale amplitude
kfreq init 1    ;do not change freq from sf
a1,a2 sfplay3 ivel, inum, kamp*ivel, kfreq, 1    ;preset index = 1
outs a1, a2

endin

</CsInstruments>
<CsScore>
f0 60    ; stay active for 1 minute

i1 0 1 60 100
i1 + 1 62 <
i1 + 1 65 <
i1 + 1 69 10

i2 5 1 60 100
i2 + 1 62 <
i2 7 1 65 <

```

```
i2 7 1 69 10  
e  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*sfilist, sfinstr, sfinstrm, sfpassign, sfplay, sfplaym, sfplist, sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

# sflooper

**sflooper** — Joue un preset d'échantillons SoundFont2 (SF2), générant un son stéréo, avec une boucle en fondu-enchaîné à durée variable, définie par l'utilisateur.

## Description

Joue un preset d'échantillons SoundFont2 (SF2), générant un son stéréo, comme *sfplay*. Mais à l'inverse de ce dernier, il ignore les points de boucle fixés dans le fichier SF2 et les remplace par une boucle en fondu-enchaîné définie par l'utilisateur. C'est un mélange de *sfplay* et de *flooper2*.

## Syntaxe

```
ar1, ar2 sflooper ivel, inotenum, kamp, kpitch, ipreindex, kloopstart, kloopend, kcrossfade \
[, istart, imode, ifenv, iskip]
```

## Initialisation

*ivel* -- vitesse.

*inotenum* -- numéro de note MIDI.

*ipreindex* -- indice de preset.

*istart* -- début de la lecture en secondes.

*imode* -- modes de boucle : 0 à l'endroit, 1 à l'envers, 2 à l'envers et à l'endroit [0 par défaut].

*ifenv* -- s'il est différent de zéro, numéro de la table de l'enveloppe de fondu-enchaîné. La valeur par défaut de 0 définit un fondu-enchaîné linéaire.

*iskip* -- s'il vaut 1, l'initialisation de l'opcode est ignorée, pour les notes liées, l'exécution continuant depuis la position dans la boucle où la note précédente s'est terminée. Avec la valeur par défaut de 0, l'initialisation a lieu.

## Exécution

*kamp* -- contrôle de l'amplitude

*kpitch* -- contrôle de la hauteur (rapport de transposition) ; les valeurs négatives sont interdites.

*kloopstart* -- début de la boucle (en secondes). Noter que bien qu'étant de taux-k, les paramètres de boucle comme celui-ci ne sont mis à jour qu'une fois par itération de la boucle. Si le début de la boucle est fixé au-delà de la fin des échantillons, il n'y aura pas de boucle.

*kloopend* -- fin de la boucle (en secondes), mis à jour une seule fois par itération de la boucle.

*kcrossfade* -- longueur du fondu enchaîné (en secondes), mis à jour une seule fois par itération de la boucle et limité par la longueur de la boucle.

*sflooper* joue un preset, générant un son stéréo.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données

échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

Note : les points de boucle sont fixés par rapport à la touche de base de chaque son échantillonné faisant partie du preset du soundfont. Par exemple, un soundfont peut avoir un seul son échantillonné pour tout le clavier. Dans ce cas, *sflooper* fonctionnera comme *flooper* et *flooper2*, car lorsque le son échantillonné est transposé (joué à différentes vitesses), la boucle se raccourcit ou s'allonge. Au contraire, si le soundfont possède un son échantillonné pour chaque touche, il n'y aura pas de transposition et la boucle gardera la même longueur (sauf si *kpitch* est modifié).

## Exemples

Voici un exemple de l'opcode *sflooper*. Il utilise le fichier *sflooper.csd* [examples/sflooper.csd].

### Exemple 904. Exemple de l'opcode *sflooper*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac -+rtmidi=virtual -M0      ;;realtime audio in, midi in
; For Non-realtime ouput leave only the line below:
; -o sflooper.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

isf    sfload "07AcousticGuitar.sf2"
       sfpassign 0, isf

instr 1 ; play from score and midi keyboard

       mididefault 60, p3
       midinoteonkey p4, p5
inum   init p4
ivel   init p5
print  ivel

ivel   init    ivel/127 ;velocity dependent
kamp   linsegr 1,1,1,.1,0 ;envelope
kamp   = kamp * .0002 ;scale amplitude (= kamp/5000)
kfreq  init 1 ;do not change freq from sf
; "07AcousticGuitar.sf2" contains 2 samples, on notes E1 and C#4
; start loop from beginning, loop .2 seconds - on the root key of these samples
aL,aR  sflooper ivel, inum, kamp*ivel, kfreq, 0, 0, .2, .05
       outs aL, aR

endin
</CsInstruments>
<CsScore>
f0 60 ; stay active for 1 minute

i1 0 1 60 100
i1 + 1 62 <
```



```
i1 + 1 65 <  
i1 + 1 69 10  
e  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*sfilist, sfinstr, sfinstrm, sfload, sfpassign, sfplaym, sfplist, sfpreset*

## Crédits

Auteur : Victor Lazzarini  
Août 2007

Nouveau dans la version 5.07 de Csound

# sfpassign

`sfpassign` — Associe tous les presets d'un fichier d'échantillons SoundFont2 (SF2) à une suite croissante d'indices numériques.

## Description

Associe tous les presets d'un fichier d'échantillons SoundFont2 (SF2) préalablement chargé en mémoire à une suite croissante d'indices numériques. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

`sfpassign` doit être placé dans la section d'en-tête d'un orchestre de Csound.

## Syntaxe

```
sfpassign istartindex, ifilhandle[, imsgs]
```

## Initialisation

`istartindex` -- indice de départ de la séquence à associer à l'ensemble des presets.

`ifilhandle` -- nombre unique généré par l'opcode `sfload` à utiliser comme identificateur pour un fichier SF2. On peut charger et activer plusieurs fichiers SF2 en même temps.

`imsgs` -- s'il est différent de zéro, les messages sont supprimés.

## Exécution

`sfpassign` associe tous les presets d'un fichier d'échantillons SoundFont2 (SF2) préalablement chargé en mémoire à une suite croissante d'indices numériques, à utiliser ensuite avec les opcodes `sfplay` et `sfplaym`. `istartindex` indique la valeur du premier indice. On peut placer plusieurs instances de `sfpassign` dans la section d'en-tête d'un orchestre, chacune associant une séquence d'indices aux presets d'un fichier SF2 différent. L'utilisateur doit veiller à ce que les valeurs d'indice des preset de fichiers SF2 différents soient disjointes.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Exemples

Voici un exemple de l'opcode `sfpassign`. Il utilise le fichier `sfpassign.csd` [exemples/sfpassign.csd].

### Exemple 905. Exemple de l'opcode `sfpassign`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0    ;;realtime audio out, virtual midi in
;-iadc    ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sfpassign.wav -W    ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

;load two soundfonts
gisf sfload "07AcousticGuitar.sf2"
gir sfload "01hpschd.sf2"
sfplist gisf
sfplist gir
sfpassign 0, gisf
sfpassign 1, gir

instr 1 ; play guitar from score and midi keyboard - preset index = 0

mididefault 60, p3
midinoteonkey p4, p5
inum init p4
ivel init p5
ivel init ivel/127    ;make velocity dependent
kamp linsegr 1, 1, 1, .1, 0
kamp = kamp/5000    ;scale amplitude
kfreq init 1    ;do not change freq from sf
a1,a2 sfplay3 ivel, inum, kamp*ivel, kfreq, 0    ;preset index = 0
outs a1, a2

endin

instr 2 ; play harpsichord from score and midi keyboard - preset index = 1

mididefault 60, p3
midinoteonkey p4, p5
inum init p4
ivel init p5
ivel init ivel/127    ;make velocity dependent
kamp linsegr 1, 1, 1, .1, 0
kamp = kamp/1000    ;scale amplitude
kfreq init 1    ;do not change freq from sf
a1,a2 sfplay3 ivel, inum, kamp*ivel, kfreq, 1    ;preset index = 1
outs a1, a2

endin

</CsInstruments>
<CsScore>
f0 60    ; stay active for 1 minute

i1 0 1 60 100
i1 + 1 62 <
i1 + 1 65 <
i1 + 1 69 10

i2 5 1 60 100
i2 + 1 62 <
i2 7 1 65 <

```

```
i2 7 1 69 10
```

```
e  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*sfilist, sfinstr, sfinstrm, sfload, sfplay, sfplaym, sfplist, sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

# sfplay3

`sfplay3` — Joue un preset d'échantillons SoundFont2 (SF2), générant un son stéréo avec interpolation cubique.

## Description

Joue un preset d'échantillons SoundFont2 (SF2), générant un son stéréo avec interpolation cubique. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## Syntaxe

```
ar1, ar2 sfplay3 ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]
```

## Initialisation

*ivel* -- vitesse.

*inotenum* -- numéro de note MIDI.

*ipreindex* -- indice du preset.

*iflag* -- (facultatif) -- drapeau concernant le comportement de *xfreq* et de *inotenum*.

*ioffset* (facultatif) -- endroit où commence la lecture, en échantillons.

*ienv* (facultatif) -- active et détermine l'enveloppe d'amplitude. 0 = pas d'enveloppe, 1 = attaque et chute linéaires, 2 = attaque linéaire, chute exponentielle (voir ci-dessous). La valeur par défaut est 0.

## Exécution

*xamp* -- facteur de correction de l'amplitude.

*xfreq* -- valeur de fréquence ou multiplicateur de fréquence, selon la valeur de *iflag*. Quand *iflag* = 0, *xfreq* est un multiplicateur de la fréquence par défaut, fixée par le preset SF2 à la valeur *inotenum*. Quand *iflag* = 1, *xfreq* est la fréquence absolue du son produit, en Hz. La valeur par défaut est 0.

Lorsque *iflag* = 0, *inotenum* fixe la fréquence de la sortie en fonction du numéro de note MIDI utilisé, et *xfreq* est utilisé comme un multiplicateur. Lorsque *iflag* = 1, la fréquence de la sortie est fixée directement par *xfreq*. Cela permet l'utilisation de n'importe quelle échelle micro-tonale. Cependant, cette méthode n'est conçue pour travailler correctement qu'avec des presets accordés selon le classique tempérament égal. L'utilisation de cette méthode avec un preset ayant déjà un accordage non standard ou bien avec des presets de drum-kit donnera des résultats imprévisibles.

L'amplitude peut être ajustée en variant l'argument *xamp* qui agit comme un multiplicateur.

Noter que *xamp* et *xfreq* peuvent opérer aussi bien au taux-k qu'au taux-a. Les deux arguments doivent utiliser des variables de même taux, sinon *sfplay3* ne fonctionnera pas correctement. *ipreindex* doit contenir un numéro associé préalablement à un preset, ou Csound se plantera.

Le paramètre *ioffset* permet de démarrer le son depuis un autre échantillon que le premier. Il faut s'assurer que sa valeur est inférieure à la longueur du son. Sinon, il y a un risque de plantage de Csound.

Le paramètre *ienv* active et détermine l'enveloppe d'amplitude utilisée. Sa valeur par défaut est 0, soit pas d'enveloppe. Si *ienv* vaut 1, les portions de l'attaque et de la chute sont linéaires. S'il vaut 2, l'attaque est linéaire et la chute est exponentielle. La portion de relâchement de l'enveloppe n'a pas encore été implémentée.

*sfplay3* joue un preset, générant un son stéréo avec interpolation cubique. *ivel* n'affecte pas directement l'amplitude de la sortie, mais indique à *sfplay* quels échantillons choisir dans les presets à sons échantillonnés multiples, séparés par la vitesse.

*sfplay3* est une version de *sfplay* avec interpolation cubique. La différence de qualité sonore est notable, particulièrement avec les échantillons transposés dans le grave. Pour les échantillons transposés dans l'aigu, la différence est moins appréciable et je suggère d'utiliser les versions avec interpolation linéaire, car elles sont plus rapides.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Exemples

Voici un exemple de l'opcode *sfplay3*. Il utilise le fichier *sfplay3.csd* [examples/sfplay3.csd].

### Exemple 906. Exemple de l'opcode *sfplay3*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0 ;;realtime audio out
;-iadc ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sfplay3.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gitwelve ftgen 1, 0, 16, -2, 12, 2, 440, 69, 1, 16/15, 9/8, 6/5, 5/4, 4/3, 7/5, 3/2, 8/5, 5/3, 9/5, 15/8
givife ftgen 2, 0, 16, -2, 5, 2, 261.659, 60, 1, 1.1486, 1.3195, 1.5157, 1.7411, 2.00

giSF sfload "01hpschd.sf2"
sfplist giSF
gipre sfpreset 0, 0, giSF, 0

instr 1

mididefault 60, p3
midinoteonkey p4, p5
ikey = p4
ivel = p5
```

```
aenv    linsegr 1, 1, 1, 1, 0    ;envelope
icps    cpstuni ikey, gitwelve    ;12 tones per octave
iamp    = 0.0004    ;scale amplitude
iamp    = iamp * ivel * 1/128    ;make velocity-dependent
aL, aR  sfplay3 ivel, ikey, iamp, icps, gipre, 1
aL      = aL * aenv
aR      = aR * aenv
        outs aL, aR

endin

instr 2

    mididefault 60, p3
    midinoteonkey p4, p5
ikey = p4
ivel = p5
aenv    linsegr 1, 1, 1, 1, 0    ;envelope
icps    cpstuni ikey, givife    ;5 tones per octave
iamp    = 0.0004    ;scale amplitude
iamp    = iamp * ivel * 1/128    ;make velocity-dependent
aL, aR  sfplay3 ivel, ikey, iamp, icps, gipre, 1
aL      = aL * aenv
aR      = aR * aenv
        outs aL, aR

endin
</CsInstruments>
<CsScore>
f0 60 ;play for 60 seconds
;instr.1 using ftable 1
i1 0 1 60 100
i1 + 1 62 <
i1 + 1 65 <
i1 + 1 69 40

;instr.2 using ftable 2
i2 5 1 60 100
i2 + 1 62 <
i2 + 1 65 <
i2 + 1 69 40
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*sfilist, sfinstr3, sfinstr3m, sfinstr, sfinstrm, sfload, sfpassign, sfplay3m, sfplaym, sfplay, sfplist, sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

Nouveau paramètre facultatif *ienv* dans la version 5.09

# sfplay3m

**sfplay3m** — Joue un preset d'échantillons SoundFont2 (SF2), générant un son mono avec interpolation cubique.

## Description

Joue un preset d'échantillons SoundFont2 (SF2), générant un son mono avec interpolation cubique. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## Syntaxe

```
ares sfplay3m ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]
```

## Initialisation

*ivel* -- vitesse.

*inotenum* -- numéro de note MIDI.

*ipreindex* -- indice du preset.

*iflag* (optional) -- (facultatif) -- drapeau concernant le comportement de *xfreq* et de *inotenum*.

*ioffset* (facultatif) -- endroit où commence la lecture, en échantillons.

*ienv* (facultatif) -- active et détermine l'enveloppe d'amplitude. 0 = pas d'enveloppe, 1 = attaque et chute linéaires, 2 = attaque linéaire, chute exponentielle (voir ci-dessous). La valeur par défaut est 0.

## Exécution

*xamp* -- facteur de correction de l'amplitude.

*xfreq* -- valeur de fréquence ou multiplicateur de fréquence, selon la valeur de *iflag*. Quand *iflag* = 0, *xfreq* est un multiplicateur de la fréquence par défaut, fixée par le preset SF2 à la valeur *inotenum*. Quand *iflag* = 1, *xfreq* est la fréquence absolue du son produit, en Hz. La valeur par défaut est 0.

Lorsque *iflag* = 0, *inotenum* fixe la fréquence de la sortie en fonction du numéro de note MIDI utilisé, et *xfreq* est utilisé comme un multiplicateur. Lorsque *iflag* = 1, la fréquence de la sortie est fixée directement par *xfreq*. Cela permet l'utilisation de n'importe quelle échelle micro-tonale. Cependant, cette méthode n'est conçue pour travailler correctement qu'avec des presets accordés selon le classique tempérament égal. L'utilisation de cette méthode avec un preset ayant déjà un accordage non standard ou bien avec des presets de drum-kit donnera des résultats imprévisibles.

L'amplitude peut être ajustée en variant l'argument *xamp* qui agit comme un multiplicateur.

Noter que *xamp* et *xfreq* peuvent opérer aussi bien au taux-k qu'au taux-a. Les deux arguments doivent utiliser des variables de même taux, sinon *sfplay3m* ne fonctionnera pas correctement. *ipreindex* doit contenir un numéro associé préalablement à un preset, ou Csound se plantera.



Le paramètre *ioffset* permet de démarrer le son depuis un autre échantillon que le premier. Il faut s'assurer que sa valeur est inférieure à la longueur du son. Sinon, il y a un risque de plantage de Csound.

Le paramètre *ienv* active et détermine l'enveloppe d'amplitude utilisée. Sa valeur par défaut est 0, soit pas d'enveloppe. Si *ienv* vaut 1, les portions de l'attaque et de la chute sont linéaires. S'il vaut 2, l'attaque est linéaire et la chute est exponentielle. La portion de relâchement de l'enveloppe n'a pas encore été implémentée.

*sfplay3m* est une version mono de *sfplay3*. Il faut l'utiliser avec un preset mono, ou avec des presets stéréo dans lesquels la sortie stéréo n'est pas requise. Il est plus rapide que *sfplay3*.

*sfplay3m* est aussi une version avec interpolation cubique de *sfplaym*. La différence de qualité sonore est notable, particulièrement avec les échantillons transposés dans le grave. Pour les échantillons transposés dans l'aigu, la différence est moins appréciable et je suggère d'utiliser les versions avec interpolation linéaire, car elles sont plus rapides.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Exemples

Voici un exemple de l'opcode *sfplay3m*. Il utilise le fichier *sfplay3m.csd* [examples/sfplay3m.csd].

### Exemple 907. Exemple de l'opcode *sfplay3m*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0 ;;realtime audio out
;-iadc ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sfplay3m.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisf sfload "07AcousticGuitar.sf2"
sfplist gisf
sfpassign 10, gisf

instr 1 ; play from score and midi keyboard

mididefault 60, p3
midinoteonkey p4, p5
inum init p4
ivel init p5
ivel init ivel/127 ;make velocity dependent
kamp linsegr 1, 1, 1, .1, 0
kamp = kamp/7000 ;scale amplitude
kfreq init 1 ;do not change freq from sf
```

```
aout sfplay3m ivel, inum, kamp*ivel, kfreq, 10 ;preset index = 10
outs aout, aout

endin

</CsInstruments>
<CsScore>
f0 60 ; stay active for 1 minute

i1 0 1 60 127
i1 + 1 62 <
i1 + 1 65 <
i1 + 1 69 10

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*sflist, sfinstr3, sfinstr3m, sfinstr, sfinstrm, sfload, sfpassign, sfplay3, sfplaym, sfplay, sfplist, sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

Nouveau paramètre facultatif *ienv* dans la version 5.09

# sfplay

sfplay — Joue un preset d'échantillons SoundFont2 (SF2), générant un son stéréo.

## Description

Joue un preset d'échantillons SoundFont2 (SF2), générant un son stéréo. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier Sound-Font2*.

## Syntaxe

```
ar1, ar2 sfplay ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]
```

## Initialisation

*ivel* -- vitesse.

*inotenum* -- numéro de note MIDI.

*ipreindex* -- indice du preset.

*iflag* (facultatif) -- drapeau concernant le comportement de *xfreq* et de *inotenum*.

*ioffset* (facultatif) -- endroit où commence la lecture, en échantillons.

*ienv* (facultatif) -- active et détermine l'enveloppe d'amplitude. 0 = pas d'enveloppe, 1 = attaque et chute linéaires, 2 = attaque linéaire, chute exponentielle (voir ci-dessous). La valeur par défaut est 0.

## Exécution

*xamp* -- facteur de correction de l'amplitude.

*xfreq* -- valeur de fréquence ou multiplicateur de fréquence, selon la valeur de *iflag*. Quand *iflag* = 0, *xfreq* est un multiplicateur de la fréquence par défaut, fixée par le preset SF2 à la valeur *inotenum*. Quand *iflag* = 1, *xfreq* est la fréquence absolue du son produit, en Hz. La valeur par défaut est 0.

Lorsque *iflag* = 0, *inotenum* fixe la fréquence de la sortie en fonction du numéro de note MIDI utilisé, et *xfreq* est utilisé comme un multiplicateur. Lorsque *iflag* = 1, la fréquence de la sortie est fixée directement par *xfreq*. Cela permet l'utilisation de n'importe quelle échelle micro-tonale. Cependant, cette méthode n'est conçue pour travailler correctement qu'avec des presets accordés selon le classique tempérament égal. L'utilisation de cette méthode avec un preset ayant déjà un accordage non standard ou bien avec des presets de drum-kit donnera des résultats imprévisibles.

L'amplitude peut être ajustée en variant l'argument *xamp* qui agit comme un multiplicateur.

Noter que *xamp* et *xfreq* peuvent opérer aussi bien au taux-k qu'au taux-a. Les deux arguments doivent utiliser des variables de même taux, sinon *sfplay* ne fonctionnera pas correctement. *ipreindex* doit contenir un numéro associé préalablement à un preset, ou Csound se plantera.

Le paramètre *ioffset* permet de démarrer le son depuis un autre échantillon que le premier. Il faut s'assurer que sa valeur est inférieure à la longueur du son. Sinon, il y a un risque de plantage de Csound.

Le paramètre *ienv* active et détermine l'enveloppe d'amplitude utilisée. Sa valeur par défaut est 0, soit pas d'enveloppe. Si *ienv* vaut 1, les portions de l'attaque et de la chute sont linéaires. S'il vaut 2, l'attaque est linéaire et la chute est exponentielle. La portion de relâchement de l'enveloppe n'a pas encore été implémentée.

*sfplay* joue un preset, générant un son stéréo. *ivel* n'affecte pas directement l'amplitude de la sortie, mais indique à *sfplay* quels échantillons choisir dans les presets à sons échantillonnés multiples, séparés par la vitesse.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Voir aussi

*sfilist*, *sfinstr*, *sfinstrm*, *sfloat*, *sfpassign*, *sfplay3*, *sfplaym*, *sfplay3m*, *sfplist*, *sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

Nouveau paramètre facultatif *ienv* dans la version 5.09

# sfplaym

sfplaym — Joue un preset d'échantillons SoundFont2 (SF2), générant un son mono.

## Description

Joue un preset d'échantillons SoundFont2 (SF2), générant un son mono. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## Syntaxe

```
ares sfplaym ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]
```

## Initialisation

*ivel* -- vitesse.

*inotenum* -- numéro de note MIDI.

*ipreindex* -- indice du preset.

*iflag* (facultatif) -- drapeau concernant le comportement de *xfreq* et de *inotenum*.

*ioffset* (facultatif) -- endroit où commence la lecture, en échantillons.

*ienv* (facultatif) -- active et détermine l'enveloppe d'amplitude. 0 = pas d'enveloppe, 1 = attaque et chute linéaires, 2 = attaque linéaire, chute exponentielle (voir ci-dessous). La valeur par défaut est 0.

## Exécution

*xamp* -- facteur de correction de l'amplitude.

*xfreq* -- valeur de fréquence ou multiplicateur de fréquence, selon la valeur de *iflag*. Quand *iflag* = 0, *xfreq* est un multiplicateur de la fréquence par défaut, fixée par le preset SF2 à la valeur *inotenum*. Quand *iflag* = 1, *xfreq* est la fréquence absolue du son produit, en Hz. La valeur par défaut est 0.

Lorsque *iflag* = 0, *inotenum* fixe la fréquence de la sortie en fonction du numéro de note MIDI utilisé, et *xfreq* est utilisé comme un multiplicateur. Lorsque *iflag* = 1, la fréquence de la sortie est fixée directement par *xfreq*. Cela permet l'utilisation de n'importe quelle échelle micro-tonale. Cependant, cette méthode n'est conçue pour travailler correctement qu'avec des presets accordés selon le classique tempérament égal. L'utilisation de cette méthode avec un preset ayant déjà un accordage non standard ou bien avec des presets de drum-kit donnera des résultats imprévisibles.

L'amplitude peut être ajustée en variant l'argument *xamp* qui agit comme un multiplicateur.

Noter que *xamp* et *xfreq* peuvent opérer aussi bien au taux-k qu'au taux-a. Les deux arguments doivent utiliser des variables de même taux, sinon *sfplay* ne fonctionnera pas correctement. *ipreindex* doit contenir un numéro associé préalablement à un preset, ou Csound se plantera.

Le paramètre *ioffset* permet de démarrer le son depuis un autre échantillon que le premier. Il faut s'assurer que sa valeur est inférieure à la longueur du son. Sinon, il y a un risque de plantage de Csound.

Le paramètre *ienv* active et détermine l'enveloppe d'amplitude utilisée. Sa valeur par défaut est 0, soit pas d'enveloppe. Si *ienv* vaut 1, les portions de l'attaque et de la chute sont linéaires. S'il vaut 2, l'attaque est linéaire et la chute est exponentielle. La portion de relâchement de l'enveloppe n'a pas encore été implémentée.

*sfplaym* est une version mono de *sfplay*. Il faut l'utiliser avec un preset mono, ou avec des presets stéréo dans lesquels la sortie stéréo n'est pas requise. Il est plus rapide que *sfplay*.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Exemples

Voici un exemple de l'opcode *sfplaym*. Il utilise le fichier *sfplaym.csd* [examples/sfplaym.csd].

### Exemple 908. Exemple de l'opcode *sfplaym*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0 ;;realtime audio out
;-iadc ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sfplaym.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisf sload "07AcousticGuitar.sf2"
sfplist gisf
sfpassign 100, gisf

instr 1 ; play from score and midi keyboard

mididefault 60, p3
midinoteonkey p4, p5
inum init p4
ivel init p5
ivel init ivel/127 ;make velocity dependent
kamp linsegr 1, 1, 1, .1, 0
kamp = kamp/7000 ;scale amplitude
kfreq init 1 ;do not change freq from sf
aout sfplaym ivel, inum, kamp*ivel, kfreq, 100 ;preset index = 100
outs aout, aout

endin

</CsInstruments>
<CsScore>
f0 60 ; stay active for 1 minute
```

```
i1 0 1 60 127
i1 + 1 62 <
i1 + 1 65 <
i1 + 1 69 10

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*sfilist, sfinstr, sfinstrm, sfload, sfpassign, sfplay, sfplay3, sfplay3m, sfplist, sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

Nouveau paramètre facultatif *ienv* dans la version 5.09

# sfplist

`sfplist` — Imprime une liste de tous les presets d'un fichier d'échantillons SoundFont2 (SF2).

## Description

Imprime une liste de tous les presets d'un fichier d'échantillons SoundFont2 (SF2) préalablement chargé en mémoire. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

## Syntaxe

```
sfplist ifilhandle
```

## Initialisation

*ifilhandle* -- nombre unique généré par l'opcode *sload* à utiliser comme identificateur pour un fichier SF2. On peut charger et activer plusieurs fichiers SF2 en même temps.

## Exécution

*sfplist* imprime sur la console une liste de tous les presets d'un fichier (SF2) préalablement chargé en mémoire.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Exemples

Voici un exemple de l'opcode `sfplist`. Il utilise le fichier *sfplist.csd* [examples/sfplist.csd].

### Exemple 909. Exemple de l'opcode `sfplist`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0   ;;realtime audio out, virtual midi in
;-iadc   ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sfplist.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
```



```

Odbfs = 1

gisf sload "sf_GMbank.sf2"
sfplist gisf ;list all presets
gir sfpreset 125, 3, gisf, 0 ;preset = Car Pass

instr 1 ; play from score and midi keyboard

mididefault 60, p3
midinoteonkey p4, p5
inum init p4
ivel init p5
ivel init ivel/127 ;make velocity dependent
kamp linsegr 1, 1, 1, .1, 0
kamp = kamp/6000 ;scale amplitude
kfreq init 1 ;do not change freq from sf
a1,a2 sfplay3 ivel, inum, kamp, kfreq, gir
outs a1, a2

endin

</CsInstruments>
<CsScore>
f0 60 ; stay active for 1 minute

i1 0 1 60 127
i1 + 1 62 <
i1 + 1 65 <
i1 + 1 69 10

e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie comprendra des lignes comme celles-ci :

```

Preset list of "sf_GMbank.sf2"
0) Piano 1          prog:0   bank:0
1) Piano 2          prog:1   bank:0
2) Piano 3          prog:2   bank:0
3) Honky Tonk       prog:3   bank:0
4) E.Piano 1        prog:4   bank:0
5) E.Piano 2        prog:5   bank:0
6) Harpsichord      prog:6   bank:0
.....
146) Car-Pass       prog:125  bank:3
.....

```

## Voir aussi

*sfilist, sfinstr, sfinstrm, sload, sfpassign, sfplay, sfplaym, sfpreset*

## Crédits

Auteur : Gabriel Maldonado  
 Italie  
 Mai 2000

Nouveau dans la version 4.07 de Csound

# sfpreset

`sfpreset` — Associe un preset d'un fichier d'échantillons SoundFont2 (SF2) à un indice numérique.

## Description

Associe un preset d'un fichier d'échantillons SoundFont2 (SF2) préalablement chargé en mémoire à un indice numérique. Ces opcodes permettent la gestion de la structure d'échantillon des fichiers SF2. Afin de comprendre l'utilisation de ces opcodes, il faut connaître le format SF2 dont on peut trouver une brève description dans l'annexe *Format de Fichier SoundFont2*.

*sfpreset* doit être placé dans la section d'en-tête d'un orchestre de Csound.

## Syntaxe

```
ir sfpreset iprog, ibank, ifilhandle, ipreindex
```

## Initialisation

*ir* -- valeur de retour à utiliser par les autres opcodes SF2. Pour *sfpreset*, *ir* est le *ipreindex*.

*iprog* -- numéro de programme d'une banque de presets dans un fichier SF2.

*ibank* -- numéro d'une banque spécifique d'un fichier SF2.

*ifilhandle* -- nombre unique généré par l'opcode *sload* à utiliser comme identificateur pour un fichier SF2. On peut charger et activer plusieurs fichiers SF2 en même temps.

*ipreindex* -- indice du preset.

## Exécution

*sfpreset* associe un preset d'un fichier SF2 préalablement chargé en mémoire à un indice numérique, à utiliser ensuite avec les opcodes *sfplay* et *sfplaym*. L'utilisateur doit connaître à l'avance les numéros de banque du preset afin de remplir les arguments correspondants. On peut placer plusieurs instances de *sfpreset* dans la section d'en-tête d'un orchestre, chacune associant un preset différent appartenant au même (ou à différents) fichiers SF2 à différents indices numériques.

Ces opcodes ne supportent que la structure d'échantillon des fichiers SF2. La structure de modulateur du format SoundFormat2 n'est pas supportée dans Csound. Tout traitement ou modulation des données échantillonnées est à la charge de l'utilisateur de Csound, ce qui permet de s'affranchir de toutes les restrictions imposées par le standard SF2.

## Exemples

Voici un exemple de l'opcode *sfpreset*. Il utilise le fichier *sfpreset.csd* [exemples/sfpreset.csd].

### Exemple 910. Exemple de l'opcode *sfpreset*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0   ;;realtime audio out, virtual midi in
;-iadc   ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sfpreset.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisf1 sfload "sf_GMbank.sf2"
sfplist gisf1 ;list presets of first soundfont
gisf2 sfload "07AcousticGuitar.sf2"
sfplist gisf2 ;list presets of second soundfont
gir sfpreset 50, 0, gisf1, 0 ;assign Synth Strings to index 0
giv sfpreset 0, 0, gisf2, 1 ;assign AcousticGuitar to index 1
print gir
print giv

instr 1 ; play from score and midi keyboard

mididefault 60, p3
midinoteonkey p4, p5
inum init p4
ivel init p5
ivel init ivel/127 ;make velocity dependent
kamp linsegr 1, 1, 1, .1, 0
kamp = kamp/5000 ;scale amplitude
kfreq init 1 ;do not change freq from sf
a1,a2 sfplay3 ivel, inum, kamp*ivel, kfreq, p6
outs a1, a2

endin

</CsInstruments>
<CsScore>
f0 60 ; stay active for 1 minute

i1 0 1 60 127 0 ;= Synth Strings I from first soundfont
i1 + 1 62 < .
i1 + 1 65 < .
i1 + 1 69 10 .

i1 5 1 60 127 1 ;= AcousticGuitar from second soundfont
i1 + 1 62 < .
i1 + 1 65 < .
i1 + 1 69 10 .
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*sfilist, sfinstr, sfinstrm, sfload, sfpassign, sfplay, sfplaym, sfplist*

## Crédits

Auteur : Gabriel Maldonado

Italie  
Mai 2000

Nouveau dans la version 4.07 de Csound

# shaker

shaker — Produit un son comme si l'on secouait des maracas ou un instrument similaire de type calebasse.

## Description

La sortie audio produit un son comme si l'on secouait des maracas ou un instrument similaire de type calebasse. La méthode est inspirée d'un modèle physique développé d'après Perry Cook, mais recodé pour Csound.

## Syntaxe

```
ares shaker kamp, kfreq, kbeans, kdamp, ktimes [, idecay]
```

## Initialisation

*idecay* -- S'il est présent, indique la durée d'amortissement du shaker à la fin de la note. La valeur par défaut est zéro.

## Exécution

Une note jouée sur un instrument de type maracas, avec les arguments suivants.

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note.

*kbeans* -- Le nombre de graines dans la calebasse. Une valeur de 8 est convenable.

*kdamp* -- La valeur d'amortissement du shaker. Des valeurs comprises entre 0,98 et 1 conviennent, avec une valeur raisonnable par défaut de 0,99.

*ktimes* -- Nombre de secousses.



### Note

L'argument *knum* était redondant et a donc été supprimé dans la version 3.49.

## Exemples

Voici un exemple de l'opcode shaker. Il utilise le fichier *shaker.csd* [examples/shaker.csd].

### Exemple 911. Exemple de l'opcode shaker.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in    No messages
```

```

-odac          -iadc      -d      ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o shaker.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1
instr 1
kfreq line p4, p3, 440
a1 shaker 10000, kfreq, 8, 0.999, 100, 0
out a1
endin

</CsInstruments>
<CsScore>

i 1 0 1 440
i 1 + 1 4000

e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 3.47 de Csound

Exemple corrigé grâce à un message de Istvan Varga.

# shiftin

shiftin — Transfère le contenu d'une variable audio dans un tableau unidimensionnel.

## Description

On peut utiliser cet opcode pour transférer les données d'une variable audio dans un tableau unidimensionnel. Le tableau doit avoir une longueur d'au moins *ksmps* nombres, mais il peut être plus grand. Les données sont stockées circulairement, la position d'écriture avançant de *ksmps* à chaque cycle-k. Lorsque le tableau est plein, la position d'écriture se replace au début du tableau (écrasant les anciennes positions). Avec l'opcode *shiftout*, cet opcode peut former une file FIFO.

## Syntaxe

```
kout[] shiftin asig
```

## Exécution

*kout[]* -- tableau de sortie. Doit avoir une longueur d'au moins *ksmps* nombres.

*asig* -- entrée audio

## Exemples

Voici un exemple de l'opcode shiftin. Il utilise le fichier *shiftin.csd* [examples/shiftin.csd].

### Exemple 912. Exemple de l'opcode shiftin.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-d -odac
</CsOptions>
<CsInstruments>

instr 1
idelttime = 0.5
kDel[] init sr*0.5
a1 diskin2 "fox.wav",1,0,1
a2 shiftout kDel
kDel shiftin a1
    out a1 + a2
endin

</CsInstruments>
<CsScore>
i1 0 10
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux.

## Crédits

Auteur : Victor Lazzarini  
NUI Maynooth  
2014

Nouveau dans la version 6.04



# shiftout

shiftout — Transfère le contenu d'un tableau unidimensionnel dans une variable audio.

## Description

On peut utiliser cet opcode pour transférer les données d'un tableau unidimensionnel dans une variable audio. Le tableau doit avoir une longueur d'au moins *ksmps* nombres, mais il peut être plus grand. Les données sont lues circulairement, la position de lecture avançant de *ksmps* à chaque cycle-k. Lorsque le tableau est vide, la position de lecture se replace au début du tableau. Avec l'opcode *shiftin*, cet opcode peut former une file FIFO.

## Syntaxe

```
asig shiftoutkIn[[] , ioff]
```

## Initialisation

*ioff* -- décalage initial de la position de lecture (facultatif, 0 par défaut).

## Exécution

*kin*[] -- Tableau en entrée. Doit avoir une longueur d'au moins *ksmps* nombres.

*asig* -- sortie audio

## Exemples

Voici un exemple de l'opcode shiftout. Il utilise le fichier *shiftout.csd* [examples/shiftout.csd].

### Exemple 913. Exemple de l'opcode shiftout.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-d -odac
</CsOptions>
<CsInstruments>

instr 1
idelttime = 0.5
kDel[] init sr*0.5
a1 diskin2 "fox.wav",1,0,1
a2 shiftout kDel
kDel shiftin a1
out a1 + a2
endin

</CsInstruments>
<CsScore>
i1 0 10
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux.

## Crédits

Auteur : Victor Lazzarini  
NUI Maynooth  
2014

Nouveau dans la version 6.04

# signum

signum — Fonction signe.

## Description

Retourne -1, 0 ou 1 selon le signe de  $x$ .

## Syntaxe

**signum**( $x$ ) (aucune restriction de taux)

## Exemples

Voici un exemple de l'opcode signum. Il utilise le fichier *signum.csd* [examples/signum.csd].

### Exemple 914. Exemple de l'opcode signum.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o oscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
kr      = 4410
ksmps   = 10
nchnls  = 2
0dbfs   = 1

gaArr[]  init 2

instr 1
kEnv transeg 1, p3, -3, 0

a_pi = 4 * taninv(1.0);
a1 phasor 440;
a2 = sin(2 * a_pi * 1/ksmps * a1);
a3 dcblock2 a2
asig = signum(a3)

gaArr[0] = a2 * 0.6 * kEnv
gaArr[1] = asig * 0.6 * kEnv

outs gaArr[0], gaArr[1]
endin

</CsInstruments>
<CsScore>
i 1 0 3
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch  
Université de Bath, Codemist Ltd.  
Bath, UK  
Juillet 2013

Nouveau dans la version 6.01 de Csound

# sin

sin — Calcule une fonction sinus.

## Description

Retourne sinus de  $x$  ( $x$  en radians).

## Syntaxe

**sin**( $x$ ) (pas de restriction de taux)

**sin**( $k/i[]$ ) ( $k$ - ou  $i$ -tableau)

## Exemples

Voici un exemple de l'opcode sin. Il utilise le fichier *sin.csd* [examples/sin.csd].

### Exemple 915. Exemple de l'opcode sin.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sin.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

isin1 =      sin(0)           ;sine of 0 is 0
isin2 =      sin($M_PI_2)     ;sine of pi/2 (1.5707...) is 1
isin3 =      sin($M_PI)       ;sine of pi (3.1415...) is 0
isin4 =      sin($M_PI_2 * 3) ;sine of 3/2pi (4.7123...) is -1
isin5 =      sin($M_PI * 2)    ;sine of 2pi (6.2831...) is 0
isin6 =      sin($M_PI * 4)    ;sine of 4pi is also 0
print       isin1, isin2, isin3, isin4, isin5, isin6
endin

instr 2 ;sin used in panning, after an example from Hans Mikelson

aout      vco2      0.8, 220      ; sawtooth
kpan      linseg    p4, p3, p5 ;0 = left, 1 = right
kpan      =         kpan*$M_PI_2   ;range 0-1 becomes 0-pi/2
kpanl     =         cos(kpan)
kpanr     =         sin(kpan)
outs      aout*kpanl, aout*kpanr
endin
```

```
</CsInstruments>
<CsScore>
i 1 0 0
i 2 0 5 0 1 ;move left to right
i 2 5 5 1 0 ;move right to left
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  isin1 = 0.000  isin2 = 1.000  isin3 = 0.000  isin4 = -1.000  isin5 = -0.000  isin6 = 0.000
```

## Voir aussi

*cos, cosh, cosinv, sinh, sininv, tan, tanh, taninv*

# sinh

sinh — Calcule une fonction sinus hyperbolique.

## Description

Retourne sinus hyperbolique de  $x$  ( $x$  en radians).

## Syntaxe

`sinh(x)` (pas de restriction de taux)

`sinh(k/i[])` (k- ou i-tableau)

## Exemples

Voici un exemple de l'opcode sinh. Il utilise le fichier *sinh.csd* [examples/sinh.csd].

### Exemple 916. Exemple de l'opcode sinh.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o sinh.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 1
  i1 = sinh(irad)

  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 1.175
```

## Voir aussi

*cos*, *cosh*, *cosinv*, *sin*, *sininv*, *tan*, *tanh*, *taninv*

## Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.47



# sininv

sininv — Calcule une fonction arcsinus.

## Description

Retourne arcsinus de  $x$  ( $x$  en radians).

## Syntaxe

`sininv(x)` (pas de restriction de taux)

`sininv(k/i[])` (k- ou i-tableau)

## Exemples

Voici un exemple de l'opcode `sininv`. Il utilise le fichier `sininv.csd` [examples/sininv.csd].

### Exemple 917. Exemple de l'opcode `sininv`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o sininv.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 0.5
  i1 = sininv(irad)

  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 0.524
```

## Voir aussi

*cos, cosh, cosinv, sin, sinh, tan, tanh, taninv*

## Crédits

Auteur : John ffitch

Nouveau dans la version 3.48

Exemple écrit par Kevin Conder.

# sinsyn

sinsyn — Synthèse additive d'un flot de suivi de partiel avec interpolation cubique de la phase.

## Description

L'opcode *sinsyn* prend en entrée un flot de signal PV TRACKS (tel que généré par l'opcode *partials* par exemple). Il resynthétise le signal avec interpolation linéaire de l'amplitude et interpolation cubique de la phase pour piloter un banc d'oscillateurs interpolants avec pondération de l'amplitude. *sinsyn* tente de préserver la phase des partiels du signal original et ainsi il ne permet pas de modifier la hauteur ou l'échelle temporelle du signal.

## Syntaxe

```
asig sinsyn fin, kscal, kmaxtracks, ifn
```

## Exécution

*asig* -- signal audio de sortie

*fin* -- flot PV TRACKS d'entrée

*kscal* -- pondération de l'amplitude

*kmaxtracks* -- nombre maximum de canaux dans la resynthèse. En limitant ce dernier, on obtient un effet de filtrage non-linéaire en ignorant les canaux les plus récents et de fréquences hautes (les canaux sont ordonnés respectivement par date de début et par fréquence ascendante).

*ifn* -- table de fonction contenant une période de sinusoïde (sinus ou cosinus).

## Exemples

Voici un exemple de l'opcode *sinsyn*. Il utilise le fichier *sinsyn.csd* [examples/sinsyn.csd].

### Exemple 918. Exemple de l'opcode *sinsyn*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sinsyn.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
```

```
instr 1

kmtxtr init p4
ain disk2 "fox.wav", 1
fs1,fsi2 pvsifd ain, 2048, 512,1 ; ifd analysis
fst partials fs1, fsi2, .03, 1, 3, 500 ; partial tracking
aout sinsyn fst, .5, kmxtr, 1 ; scale amplitude down
outs aout, aout

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1

i 1 0 2.7 15 ;filtering effect by using low number of tracks
i 1 + 2.7 500 ;maximum number of tracks
e
</CsScore>
</CsoundSynthesizer>
```

L'exemple ci-dessus montre le suivi de partiel d'un signal d'analyse par distribution de fréquence instantanée et la resynthèse additive avec interpolation cubique de la phase.

## Crédits

Auteur : Victor Lazzarini  
Juin 2005

Nouveau greffon dans la version 5

Novembre 2004.

# sleighbells

sleighbells — Modèle semi-physique d'un son de cloche de traîneau.

## Description

*sleighbells* est un modèle semi-physique d'un son de cloche de traîneau. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntaxe

```
ares sleighbells kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \
    [, ifreq1] [, ifreq2]
```

## Initialisation

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 32.

*idamp* (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

$\text{damping\_amount} = 0,9994 + (\text{idamp} * 0,002)$

La valeur par défaut de *damping\_amount* est 0,9994 ce qui signifie que la valeur par défaut de *idamp* est 0. Le maximum de *damping\_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 0,03.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

*imaxshake* (facultatif, 0 par défaut) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

*ifreq* (facultatif) -- la fréquence de résonance principale. La valeur par défaut est 2500.

*ifreq1* (facultatif) -- la première fréquence de résonance. La valeur par défaut est 5300.

*ifreq2* (facultatif) -- la deuxième fréquence de résonance. La valeur par défaut est 6500.

## Exécution

*kamp* -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

## Exemples

Voici un exemple de l'opcode *sleighbells*. Il utilise le fichier *sleighbells.csd* [examples/sleighbells.csd].

### Exemple 919. Exemple de l'opcode *sleighbells*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sleighbells.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

idamp = p4
asig sleighbells .7, 0.01, 32, idamp, 0.55
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0.00 0.25 0 ;short sound
i 1 0.30 0.25
i 1 0.60 0.25
i 1 0.90 0.25
i 1 1.20 0.25
i 1 1.50 1 .3 ;longer sound
i 1 1.80 0.25 0 ;short sound again
i 1 2.10 0.25
i 1 2.40 0.25
i 1 2.70 0.25
i 1 3.00 0.25
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*bamboo, dripwater, guiro, tambourine*

## Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)

Adapté par John ffitch

Université de Bath, Codemist Ltd.

Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

# slicearray

slicearray — Prend une partie d'un vecteur.

## Description

Prend une partie d'un vecteur (tableau unidimensionnel de taux-k).

## Syntaxe

```
karray slicearray kinarray, istart, iend [,istride]
```

## Initialisation

*kinarray* -- tableau en entrée.

*istart* -- indice du premier élément à prélever.

*iend* -- indice du dernier élément à prélever.

*istride* -- incrément pour les éléments source (facultatif), 1 par défaut.

## Exécution

*karray* -- tableau retourné.

## Exemples

Voici un exemple de l'opcode slicearray. Il utilise le fichier *slicearray.csd* [examples/slicearray.csd].

### Exemple 920. Exemple de l'opcode slicearray.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

;create and fill an array
kArr[] genarray_i 1, 9

;print the content
printf "%s", 1, "kArr = whole array\n"
```

```

kndx    =      0
  until kndx == lenarray(kArr) do
    printf "kArr[%d] = %f\n", kndx+1, kndx, kArr[kndx]
    kndx  +=      1
  od

;build new arrays for the slices
kArr1[] init    5
kArr2[] init    4
kArr3[] init    3

;put in first five and last four elements
kArr1 slicearray kArr, 0, 4
kArr2 slicearray kArr, 5, 8
; and three values from 1, 1+2 and 1+2+3
kArr3 slicearray kArr, 1, 5, 2

;print the content
  printf "%s", 1, "\nkArr1 = slice from index 0 to index 4\n"
kndx    =      0
  until kndx == lenarray(kArr1) do
    printf "kArr1[%d] = %f\n", kndx+1, kndx, kArr1[kndx]
    kndx  +=      1
  od
  printf "%s", 1, "\nkArr2 = slice from index 5 to index 8\n"
kndx    =      0
  until kndx == lenarray(kArr2) do
    printf "kArr2[%d] = %f\n", kndx+1, kndx, kArr2[kndx]
    kndx  +=      1
  od
  printf "%s", 1, "\nkArr3 = slice from index 1 to index 5 inc2\n"
kndx    =      0
  until kndx == lenarray(kArr3) do
    printf "kArr3[%d] = %f\n", kndx+1, kndx, kArr3[kndx]
    kndx  +=      1
  od

  turnoff
endin

</CsInstruments>
<CsScore>
i 1 0 1
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

Opcodes vectoriels

## Crédits

Auteur : John ffitch  
 Codemist Ltd  
 2013

Nouveau dans la version 6.00

Argument istride ajouté dans la version 6.10



# slider16

slider16 — Crée un banc de 16 numéros différents de messages de contrôle MIDI.

## Description

Crée un banc de 16 numéros différents de messages de contrôle MIDI.

## Syntaxe

```
i1,...,i16 slider16 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \  
    ictlnum16, imin16, imax16, init16, ifn16  
  
k1,...,k16 slider16 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \  
    ictlnum16, imin16, imax16, init16, ifn16
```

## Initialisation

*i1 ... i16* -- valeurs de sortie

*ichan* -- canal MIDI (1-16)

*ictlnum1 ... ictlnum16* -- numéro de contrôle MIDI (0-127)

*imin1 ... imin16* -- valeurs minimales pour chaque contrôleur

*imax1 ... imax16* -- valeurs maximales pour chaque contrôleur

*init1 ... init16* -- valeur initiale pour chaque contrôleur

*ifn1 ... ifn16* -- table de fonction de conversion pour chaque contrôleur

## Exécution

*k1 ... k16* -- valeurs de sortie

*slider16* est un banc de contrôleurs MIDI, utile lorsque l'on utilise un mélangeur MIDI comme le Kawai MM-16 ou autres pour changer n'importe quel paramètre du son en . Les messages de contrôle MIDI arrivant sur le port d'entrée sont convertis pour entrer dans l'intervalle *iminN*, *imaxN*, et une valeur initiale peut être fixée. On peut aussi utiliser de manière facultative une table de fonction non interpolée avec une courbe de traduction personnalisée pour obtenir, par exemple, des courbes de réponse exponentielles.

Si l'on n'a pas besoin d'une table de traduction, on fixe la valeur de *ifnN* à 0, sinon, on donne à *ifnN* un numéro de table de fonction valide. Lorsque l'on utilise une table de traduction (si *ifnN* reçoit une valeur non nulle faisant référence à une table de fonction déjà allouée), la valeur de *initN* doit être égale à celle de *iminN* ou à celle de *imaxN*, sinon la valeur de sortie initiale sera différente de celle spécifiée dans l'argument *initN*.

*slider16* fournit un banc de 16 numéros différents de messages de contrôle MIDI.

Comme les arguments d'entrée et de sortie sont nombreux, on peut scinder la ligne en utilisant le caractère '\' (slash inversé) (nouveau dans la version 3.47) pour améliorer la lisibilité. L'utilisation de ces opcodes est considérablement plus efficace que celle de (*ctrl7* et *tonek*) séparés, lorsque l'on a besoin de plus de contrôleurs.

Dans la version au taux-i de *slider16*, il n'y a pas d'argument de valeur d'entrée initiale. La sortie est prise directement dans l'état courant du tableau interne de contrôleurs de Csound.

## Voir aussi

*s16b14*, *s32b14*, *slider16f*, *slider32*, *slider32f*, *slider64*, *slider64f*, *slider8*, *slider8f*

## Crédits

Auteur: Gabriel Maldonado  
Italie  
Décember 1998

Nouveau dans la version 3.50 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# slider16f

slider16f — Crée un banc de 16 numéros différents de messages de contrôle MIDI, filtrés avant la sortie.

## Description

Crée un banc de 16 numéros différents de messages de contrôle MIDI, filtrés avant la sortie.

## Syntaxe

```
k1,...,k16 slider16f ichan, ictlnum1, imin1, imax1, init1, ifn1, \  
    icutoff1,..., ictlnum16, imin16, imax16, init16, ifn16, icutoff16
```

## Initialisation

*ichan* -- canal MIDI (1-16)

*ictlnum1* ... *ictlnum16* -- numéro de contrôle MIDI (0-127)

*imin1* ... *imin16* -- valeurs minimales pour chaque contrôleur

*imax1* ... *imax16* -- valeurs maximales pour chaque contrôleur

*init1* ... *init16* -- valeur initiale pour chaque contrôleur

*ifn1* ... *ifn16* -- table de fonction de conversion pour chaque contrôleur

*icutoff1* ... *icutoff16* -- fréquence de coupure du filtre passe-bas pour chaque contrôleur

## Exécution

*k1* ... *k16* -- valeurs de sortie

*slider16f* est un banc de contrôleurs MIDI, utile lorsque l'on utilise un mélangeur MIDI comme le Kawai MM-16 ou autres pour changer n'importe quel paramètre du son en . Les messages de contrôle MIDI arrivant sur le port d'entrée sont convertis pour entrer dans l'intervalle *iminN*, *imaxN*, et une valeur initiale peut être fixée. On peut aussi utiliser de manière facultative une table de fonction non interpolée avec une courbe de traduction personnalisée pour obtenir, par exemple, des courbes de réponse exponentielles.

Si l'on n'a pas besoin d'une table de traduction, on fixe la valeur de *ifnN* à 0, sinon, on donne à *ifnN* un numéro de table de fonction valide. Lorsque l'on utilise une table de traduction (si *ifnN* reçoit une valeur non nulle faisant référence à une table de fonction déjà allouée), la valeur de *initN* doit être égale à celle de *iminN* ou à celle de *imaxN*, sinon la valeur de sortie initiale sera différente de celle spécifiée dans l'argument *initN*.

*slider16f* fournit un banc de 16 numéros différents de messages de contrôle MIDI. Il filtre le signal avant la sortie. Cela élimine les discontinuités dues à la basse résolution du MIDI (7 bit). La fréquence de coupure peut être réglée séparément pour chaque contrôleur (intervalle recommandé : 0.1 à 5 Hz).

Comme les arguments d'entrée et de sortie sont nombreux, on peut scinder la ligne en utilisant le caractère '\' (slash inversé) (nouveau dans la version 3.47) pour améliorer la lisibilité. L'utilisation de ces opcodes est considérablement plus efficace que celle de (*ctrl7* et *tonek*) séparés, lorsque l'on a besoin de plus de contrôleurs.



## Avertissement

Les opcodes *slider16f* ne sortent pas la valeur initiale immédiatement, mais seulement après quelques cycles-k parce que le filtre introduit un léger retard dans la sortie.

## Voir aussi

*s16b14, s32b14, slider16, slider32, slider32f, slider64, slider64f, slider8, slider8f*

## Crédits

Auteur: Gabriel Maldonado  
Italie  
Décember 1998

Nouveau dans la version 3.50 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# slider16table

slider16table — Enregistre un banc de 16 messages de contrôle MIDI différents dans une table.

## Description

Enregistre un banc de 16 messages de contrôle MIDI différents dans une table.

## Syntaxe

```
kflag slider16table ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \  
    init1, ifn1, .... , ictlnum16, imin16, imax16, init16, ifn16
```

## Initialisation

*ichan* -- canal MIDI (1-16)

*ioutTable* -- numéro de la table qui contiendra la sortie

*ioffset* -- décalage dans la table de sortie. Zéro signifie que la sortie du premier contrôleur affectera le premier élément de la table. 10 signifie que la sortie du premier contrôleur affectera le onzième élément de la table.

*ictlnum1* ... *ictlnum16* -- numéro de contrôle MIDI (0-127)

*imin1* ... *imin16* -- valeurs minimales pour chaque contrôleur

*imax1* ... *imax16* -- valeurs maximales pour chaque contrôleur

*init1* ... *init16* -- valeur initiale pour chaque contrôleur

*ifn1* ... *ifn16* -- table de fonction de conversion pour chaque contrôleur

## Exécution

*kflag* -- un indicateur qui informe si un message de changement de contrôle dans le banc a été reçu. Dans ce cas, *kflag* est fixé à 1. Sinon il est fixé à 0.

*slider16table* est un banc de contrôleurs MIDI, utile lorsque l'on utilise un mélangeur MIDI comme le Kawai MM-16 ou autres pour changer n'importe quel paramètre du son en . Les messages de contrôle MIDI arrivant sur le port d'entrée sont convertis pour entrer dans l'intervalle *iminN*, *imaxN*, et une valeur initiale peut être fixée. On peut aussi utiliser de manière facultative une table de fonction non interpolée avec une courbe de traduction personnalisée pour obtenir, par exemple, des courbes de réponse exponentielles.

Si l'on n'a pas besoin d'une table de traduction, on fixe la valeur de *ifnN* à 0, sinon, on donne à *ifnN* un numéro de table de fonction valide. Lorsque l'on utilise une table de traduction (si *ifnN* reçoit une valeur non nulle faisant référence à une table de fonction déjà allouée), la valeur de *initN* doit être égale à celle de *iminN* ou à celle de *imaxN*, sinon la valeur de sortie initiale sera différente de celle spécifiée dans l'argument *initN*.

*slider16table* fournit un banc de 16 numéros différents de messages de contrôle MIDI.

Comme les arguments d'entrée et de sortie sont nombreux, on peut scinder la ligne en utilisant le caractère '\' (slash inversé) (nouveau dans la version 3.47) pour améliorer la lisibilité. L'utilisation de ces opcodes est considérablement plus efficace que celle de (*ctrl7* et *tonek*) séparés, lorsque l'on a besoin de plus de contrôleurs.

*slider16table* ressemble beaucoup à la famille des opcodes *slider16* et *sliderN* (voir leur notice pour plus d'information), à la différence que la sortie n'est pas stockée dans des variables de taux-k, mais dans une table dénotée par l'argument *ioutTable*. Il est possible de définir un indice de base afin d'utiliser la même table pour plus d'un banc de contrôleurs (ou pour un autre usage).

Il est possible d'utiliser cet opcode conjointement avec *FLslidBnk2Setk* et avec *FLslidBnk2*, ce qui permet de synchroniser la position des valeurs MIDI à la position des widgets valeurs FLTK de *FLslidBnk2*. Noter qu'il faut spécifier les mêmes valeurs de min/max et de réponse linéaire/exponentielle dans *sliderNtable* et dans *FLslidBnk2*. Il y a une exception si l'on utilise une réponse dans une table indexée au lieu d'une réponse lin/exp. Dans ce cas, afin d'obtenir un résultat utilisable, la réponse par table indexée et les valeurs min/max ne doivent être fixées que dans *FLslidBnk2*, alors que dans *sliderNtable*, il faut fixer une réponse linéaire, un minimum de zéro et un maximum de un dans tous les contrôleurs.

## Voir aussi

*slider16tablef*, *slider32table*, *slider32tablef*, *slider64table*, *slider64tablef*, *slidertable8*, *slider8tablef*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06 de Csound.

# slider16tablef

**slider16tablef** — Enregistre un banc de 16 messages de contrôle MIDI différents dans une table, filtrés avant la sortie.

## Description

Enregistre un banc de 16 messages de contrôle MIDI différents dans une table, filtrés avant la sortie.

## Syntaxe

```
kflag slider16tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \  
init1, ifn1, icutoff1, .... , ictlnum16, imin16, imax16, init16, ifn16, icutoff16
```

## Initialisation

*ichan* -- canal MIDI (1-16)

*ioutTable* -- numéro de la table qui contiendra la sortie

*ioffset* -- décalage dans la table de sortie. Zéro signifie que la sortie du premier contrôleur affectera le premier élément de la table. 10 signifie que la sortie du premier contrôleur affectera le onzième élément de la table.

*ictlnum1* ... *ictlnum16* -- numéro de contrôle MIDI (0-127)

*imin1* ... *imin16* -- valeurs minimales pour chaque contrôleur

*imax1* ... *imax16* -- valeurs maximales pour chaque contrôleur

*init1* ... *init16* -- valeur initiale pour chaque contrôleur

*ifn1* ... *ifn16* -- table de fonction de conversion pour chaque contrôleur

*icutoff1* ... *icutoff16* -- fréquence de coupure du filtre passe-bas pour chaque contrôleur

## Exécution

*kflag* -- un indicateur qui informe si un message de changement de contrôle dans le banc a été reçu. Dans ce cas, *kflag* est fixé à 1. Sinon il est fixé à 0.

*slider16tablef* est un banc de contrôleurs MIDI, utile lorsque l'on utilise un mélangeur MIDI comme le Kawai MM-16 ou autres pour changer n'importe quel paramètre du son en . Les messages de contrôle MIDI arrivant sur le port d'entrée sont convertis pour entrer dans l'intervalle *iminN*, *imaxN*, et une valeur initiale peut être fixée. On peut aussi utiliser de manière facultative une table de fonction non interpolée avec une courbe de traduction personnalisée pour obtenir, par exemple, des courbes de réponse exponentielles.

Si l'on n'a pas besoin d'une table de traduction, on fixe la valeur de *ifnN* à 0, sinon, on donne à *ifnN* un numéro de table de fonction valide. Lorsque l'on utilise une table de traduction (si *ifnN* reçoit une valeur non nulle faisant référence à une table de fonction déjà allouée), la valeur de *initN* doit être égale à celle de *iminN* ou à celle de *imaxN*, sinon la valeur de sortie initiale sera différente de celle spécifiée dans l'argument *initN*.

*slider16tablef* fournit un banc de 16 numéros différents de messages de contrôle MIDI. Il filtre le signal avant la sortie. Cela élimine les discontinuités dues à la basse résolution du MIDI (7 bit). La fréquence de coupure peut être réglée séparément pour chaque contrôleur (intervalle recommandé : 0.1 à 5 Hz).

Comme les arguments d'entrée et de sortie sont nombreux, on peut scinder la ligne en utilisant le caractère ``` (slash inversé) (nouveau dans la version 3.47) pour améliorer la lisibilité. L'utilisation de ces opcodes est considérablement plus efficace que celle de (*ctrl7* et *tonek*) séparés, lorsque l'on a besoin de plus de contrôleurs.

*slider16tablef* ressemble beaucoup à la famille des opcodes *slider16f* et *sliderNf* (voir leur notice pour plus d'information), à la différence que la sortie n'est pas stockée dans des variables de taux-k, mais dans une table dénotée par l'argument *ioutTable*. Il est possible de définir un indice de base afin d'utiliser la même table pour plus d'un banc de contrôleurs (ou pour un autre usage).

Il est possible d'utiliser cet opcode conjointement avec *FLslidBnk2Setk* et avec *FLslidBnk2*, ce qui permet de synchroniser la position des valeurs MIDI à la position des widgets valeurs FLTK de *FLslidBnk2*. Noter qu'il faut spécifier les mêmes valeurs de min/max et de réponse linéaire/exponentielle dans *sliderNtablef* et dans *FLslidBnk2*. Il y a une exception si l'on utilise une réponse dans une table indexée au lieu d'une réponse lin/exp. Dans ce cas, afin d'obtenir un résultat utilisable, la réponse par table indexée et les valeurs min/max ne doivent être fixées que dans *FLslidBnk2*, alors que dans *sliderNtablef*, il faut fixer une réponse linéaire, un minimum de zéro et un maximum de un dans tous les contrôleurs.



### Avertissement

Les opcodes *slider16tablef* ne sortent pas la valeur initiale immédiatement, mais seulement après quelques cycles-k parce que le filtre introduit un léger retard dans la sortie.

## Voir aussi

*slider16table*, *slider32table*, *slider32tablef*, *slider64table*, *slider64tablef*, *slider8table*, *slider8tablef*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06 de Csound.



# slider32

slider32 — Crée un banc de 32 numéros différents de messages de contrôle MIDI.

## Description

Crée un banc de 32 numéros différents de messages de contrôle MIDI.

## Syntaxe

```
i1,...,i32 slider32 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \  
    ictlnum32, imin32, imax32, init32, ifn32  
  
k1,...,k32 slider32 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \  
    ictlnum32, imin32, imax32, init32, ifn32
```

## Initialisation

*i1 ... i32* -- valeurs de sortie

*ichan* -- canal MIDI (1-16)

*ictlnum1 ... ictlnum32* -- numéro de contrôle MIDI (0-127)

*imin1 ... imin32* -- valeurs minimales pour chaque contrôleur

*imax1 ... imax32* -- valeurs maximales pour chaque contrôleur

*init1 ... init32* -- valeur initiale pour chaque contrôleur

*ifn1 ... ifn32* -- table de fonction de conversion pour chaque contrôleur

## Exécution

*k1 ... k32* -- valeurs de sortie

*slider32* est un banc de contrôleurs MIDI, utile lorsque l'on utilise un mélangeur MIDI comme le Kawai MM-16 ou autres pour changer n'importe quel paramètre du son en . Les messages de contrôle MIDI arrivant sur le port d'entrée sont convertis pour entrer dans l'intervalle *iminN*, *imaxN*, et une valeur initiale peut être fixée. On peut aussi utiliser de manière facultative une table de fonction non interpolée avec une courbe de traduction personnalisée pour obtenir, par exemple, des courbes de réponse exponentielles.

Si l'on n'a pas besoin d'une table de traduction, on fixe la valeur de *ifnN* à 0, sinon, on donne à *ifnN* un numéro de table de fonction valide. Lorsque l'on utilise une table de traduction (si *ifnN* reçoit une valeur non nulle faisant référence à une table de fonction déjà allouée), la valeur de *initN* doit être égale à celle de *iminN* ou à celle de *imaxN*, sinon la valeur de sortie initiale sera différente de celle spécifiée dans l'argument *initN*.

*slider32* fournit un banc de 32 numéros différents de messages de contrôle MIDI.

Comme les arguments d'entrée et de sortie sont nombreux, on peut scinder la ligne en utilisant le caractère '\' (slash inversé) (nouveau dans la version 3.47) pour améliorer la lisibilité. L'utilisation de ces opcodes est considérablement plus efficace que celle de (*ctrl7* et *tonek*) séparés, lorsque l'on a besoin de plus de contrôleurs.

Dans la version au taux-i de *slider32*, il n'y a pas d'argument de valeur d'entrée initiale. La sortie est prise directement dans l'état courant du tableau interne de contrôleurs de Csound.

## Voir aussi

*s16b14*, *s32b14*, *slider16*, *slider16f*, *slider32f*, *slider64*, *slider64f*, *slider8*, *slider8f*

## Crédits

Auteur: Gabriel Maldonado  
Italie  
Décember 1998

Nouveau dans la version 3.50 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# slider32f

slider32f — Crée un banc de 32 numéros différents de messages de contrôle MIDI, filtrés avant la sortie.

## Description

Crée un banc de 32 numéros différents de messages de contrôle MIDI, filtrés avant la sortie.

## Syntaxe

```
k1,...,k32 slider32f ichan, ictlnum1, imin1, imax1, init1, ifn1, icutoff1, \  
..., ictlnum32, imin32, imax32, init32, ifn32, icutoff32
```

## Initialisation

*ichan* -- canal MIDI (1-16)

*ictlnum1* ... *ictlnum32* -- numéro de contrôle MIDI (0-127)

*imin1* ... *imin32* -- valeurs minimales pour chaque contrôleur

*imax1* ... *imax32* -- valeurs maximales pour chaque contrôleur

*init1* ... *init32* -- valeur initiale pour chaque contrôleur

*ifn1* ... *ifn32* -- table de fonction de conversion pour chaque contrôleur

*icutoff1* ... *icutoff32* -- fréquence de coupure du filtre passe-bas pour chaque contrôleur

## Exécution

*k1* ... *k32* -- valeurs de sortie

*slider32f* est un banc de contrôleurs MIDI, utile lorsque l'on utilise un mélangeur MIDI comme le Kawai MM-16 ou autres pour changer n'importe quel paramètre du son en . Les messages de contrôle MIDI arrivant sur le port d'entrée sont convertis pour entrer dans l'intervalle *iminN*, *imaxN*, et une valeur initiale peut être fixée. On peut aussi utiliser de manière facultative une table de fonction non interpolée avec une courbe de traduction personnalisée pour obtenir, par exemple, des courbes de réponse exponentielles.

Si l'on n'a pas besoin d'une table de traduction, on fixe la valeur de *ifnN* à 0, sinon, on donne à *ifnN* un numéro de table de fonction valide. Lorsque l'on utilise une table de traduction (si *ifnN* reçoit une valeur non nulle faisant référence à une table de fonction déjà allouée), la valeur de *initN* doit être égale à celle de *iminN* ou à celle de *imaxN*, sinon la valeur de sortie initiale sera différente de celle spécifiée dans l'argument *initN*.

*slider32f* fournit un banc de 32 numéros différents de messages de contrôle MIDI. Il filtre le signal avant la sortie. Cela élimine les discontinuités dues à la basse résolution du MIDI (7 bit). La fréquence de coupure peut être réglée séparément pour chaque contrôleur (intervalle recommandé : 0.1 à 5 Hz).

Comme les arguments d'entrée et de sortie sont nombreux, on peut scinder la ligne en utilisant le caractère '\' (slash inversé) (nouveau dans la version 3.47) pour améliorer la lisibilité. L'utilisation de ces opcodes est considérablement plus efficace que celle de (*ctrl7* et *tonek*) séparés, lorsque l'on a besoin de plus de contrôleurs.



## Avertissement

Les opcodes *slider32f* ne sortent pas la valeur initiale immédiatement, mais seulement après quelques cycles-k parce que le filtre introduit un léger retard dans la sortie.

## Voir aussi

*s16b14, s32b14, slider16, slider16f, slider32, slider64, slider64f, slider8, slider8f*

## Crédits

Auteur: Gabriel Maldonado  
Italie  
Décember 1998

Nouveau dans la version 3.50 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# slider32table

slider32table — Enregistre un banc de 32 messages de contrôle MIDI différents dans une table.

## Description

Enregistre un banc de 32 messages de contrôle MIDI différents dans une table.

## Syntaxe

```
kflag slider32table ichan, ioutTable, ioffset, ictlnum1, imin1, \  
imax1, init1, ifn1, ...., ictlnum32, imin32, imax32, init32, ifn32
```

## Initialisation

*ichan* -- canal MIDI (1-16)

*ioutTable* -- numéro de la table qui contiendra la sortie

*ioffset* -- décalage dans la table de sortie. Zéro signifie que la sortie du premier contrôleur affectera le premier élément de la table. 10 signifie que la sortie du premier contrôleur affectera le onzième élément de la table.

*ictlnum1* ... *ictlnum32* -- numéro de contrôle MIDI (0-127)

*imin1* ... *imin32* -- valeurs minimales pour chaque contrôleur

*imax1* ... *imax32* -- valeurs maximales pour chaque contrôleur

*init1* ... *init32* -- valeur initiale pour chaque contrôleur

*ifn1* ... *ifn32* -- table de fonction de conversion pour chaque contrôleur

## Exécution

*kflag* -- un indicateur qui informe si un message de changement de contrôle dans le banc a été reçu. Dans ce cas, *kflag* est fixé à 1. Sinon il est fixé à 0.

*slider32table* est un banc de contrôleurs MIDI, utile lorsque l'on utilise un mélangeur MIDI comme le Kawai MM-16 ou autres pour changer n'importe quel paramètre du son en . Les messages de contrôle MIDI arrivant sur le port d'entrée sont convertis pour entrer dans l'intervalle *iminN*, *imaxN*, et une valeur initiale peut être fixée. On peut aussi utiliser de manière facultative une table de fonction non interpolée avec une courbe de traduction personnalisée pour obtenir, par exemple, des courbes de réponse exponentielles.

Si l'on n'a pas besoin d'une table de traduction, on fixe la valeur de *ifnN* à 0, sinon, on donne à *ifnN* un numéro de table de fonction valide. Lorsque l'on utilise une table de traduction (si *ifnN* reçoit une valeur non nulle faisant référence à une table de fonction déjà allouée), la valeur de *initN* doit être égale à celle de *iminN* ou à celle de *imaxN*, sinon la valeur de sortie initiale sera différente de celle spécifiée dans l'argument *initN*.

*slider32table* fournit un banc de 32 numéros différents de messages de contrôle MIDI.

Comme les arguments d'entrée et de sortie sont nombreux, on peut scinder la ligne en utilisant le caractère '\' (slash inversé) (nouveau dans la version 3.47) pour améliorer la lisibilité. L'utilisation de ces opcodes est considérablement plus efficace que celle de (*ctrl7* et *tonek*) séparés, lorsque l'on a besoin de plus de contrôleurs.

*slider32table* ressemble beaucoup à la famille des opcodes *slider32* et *sliderN* (voir leur notice pour plus d'information), à la différence que la sortie n'est pas stockée dans des variables de taux-k, mais dans une table dénotée par l'argument *ioutTable*. Il est possible de définir un indice de base afin d'utiliser la même table pour plus d'un banc de contrôleurs (ou pour un autre usage).

Il est possible d'utiliser cet opcode conjointement avec *FLslidBnk2Setk* et avec *FLslidBnk2*, ce qui permet de synchroniser la position des valeurs MIDI à la position des widgets valeurs FLTK de *FLslidBnk2*. Noter qu'il faut spécifier les mêmes valeurs de min/max et de réponse linéaire/exponentielle dans *sliderNtable* et dans *FLslidBnk2*. Il y a une exception si l'on utilise une réponse dans une table indexée au lieu d'une réponse lin/exp. Dans ce cas, afin d'obtenir un résultat utilisable, la réponse par table indexée et les valeurs min/max ne doivent être fixées que dans *FLslidBnk2*, alors que dans *sliderNtable*, il faut fixer une réponse linéaire, un minimum de zéro et un maximum de un dans tous les contrôleurs.

## Voir aussi

*slider16table*, *slider16tablef*, *slider32tablef*, *slider64table*, *slider64tablef*, *slider8table*, *slider8tablef*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06 de Csound.

# slider32tablef

`slider32tablef` — Enregistre un banc de 32 messages de contrôle MIDI différents dans une table, filtrés avant la sortie.

## Description

Enregistre un banc de 32 messages de contrôle MIDI différents dans une table, filtrés avant la sortie.

## Syntaxe

```
kflag slider32tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \  
init1, ifn1, icutoff1, .... , ictlnum32, imin32, imax32, init32, ifn32, icutoff32
```

## Initialisation

*ichan* -- canal MIDI (1-16)

*ioutTable* -- numéro de la table qui contiendra la sortie

*ioffset* -- décalage dans la table de sortie. Zéro signifie que la sortie du premier contrôleur affectera le premier élément de la table. 10 signifie que la sortie du premier contrôleur affectera le onzième élément de la table.

*ictlnum1* ... *ictlnum32* -- numéro de contrôle MIDI (0-127)

*imin1* ... *imin32* -- valeurs minimales pour chaque contrôleur

*imax1* ... *imax32* -- valeurs maximales pour chaque contrôleur

*init1* ... *init32* -- valeur initiale pour chaque contrôleur

*ifn1* ... *ifn32* -- table de fonction de conversion pour chaque contrôleur

*icutoff1* ... *icutoff32* -- fréquence de coupure du filtre passe-bas pour chaque contrôleur

## Exécution

*kflag* -- un indicateur qui informe si un message de changement de contrôle dans le banc a été reçu. Dans ce cas, *kflag* est fixé à 1. Sinon il est fixé à 0.

*slider32tablef* est un banc de contrôleurs MIDI, utile lorsque l'on utilise un mélangeur MIDI comme le Kawai MM-16 ou autres pour changer n'importe quel paramètre du son en . Les messages de contrôle MIDI arrivant sur le port d'entrée sont convertis pour entrer dans l'intervalle *iminN*, *imaxN*, et une valeur initiale peut être fixée. On peut aussi utiliser de manière facultative une table de fonction non interpolée avec une courbe de traduction personnalisée pour obtenir, par exemple, des courbes de réponse exponentielles.

Si l'on n'a pas besoin d'une table de traduction, on fixe la valeur de *ifnN* à 0, sinon, on donne à *ifnN* un numéro de table de fonction valide. Lorsque l'on utilise une table de traduction (si *ifnN* reçoit une valeur non nulle faisant référence à une table de fonction déjà allouée), la valeur de *initN* doit être égale à celle de *iminN* ou à celle de *imaxN*, sinon la valeur de sortie initiale sera différente de celle spécifiée dans l'argument *initN*.

*slider32tablef* fournit un banc de 32 numéros différents de messages de contrôle MIDI. Il filtre le signal avant la sortie. Cela élimine les discontinuités dues à la basse résolution du MIDI (7 bit). La fréquence de coupure peut être réglée séparément pour chaque contrôleur (intervalle recommandé : 0.1 à 5 Hz).

Comme les arguments d'entrée et de sortie sont nombreux, on peut scinder la ligne en utilisant le caractère `\` (slash inversé) (nouveau dans la version 3.47) pour améliorer la lisibilité. L'utilisation de ces opcodes est considérablement plus efficace que celle de (*ctrl7* et *tonek*) séparés, lorsque l'on a besoin de plus de contrôleurs.

*slider32tablef* ressemble beaucoup à la famille des opcodes *slider32f* et *sliderNf* (voir leur notice pour plus d'information), à la différence que la sortie n'est pas stockée dans des variables de taux-k, mais dans une table dénotée par l'argument *ioutTable*. Il est possible de définir un indice de base afin d'utiliser la même table pour plus d'un banc de contrôleurs (ou pour un autre usage).

Il est possible d'utiliser cet opcode conjointement avec *FLslidBnk2Setk* et avec *FLslidBnk2*, ce qui permet de synchroniser la position des valeurs MIDI à la position des widgets valeurs FLTK de *FLslidBnk2*. Noter qu'il faut spécifier les mêmes valeurs de min/max et de réponse linéaire/exponentielle dans *sliderNtablef* et dans *FLslidBnk2*. Il y a une exception si l'on utilise une réponse dans une table indexée au lieu d'une réponse lin/exp. Dans ce cas, afin d'obtenir un résultat utilisable, la réponse par table indexée et les valeurs min/max ne doivent être fixées que dans *FLslidBnk2*, alors que dans *sliderNtablef*, il faut fixer une réponse linéaire, un minimum de zéro et un maximum de un dans tous les contrôleurs.



### Avertissement

Les opcodes *slider32tablef* ne sortent pas la valeur initiale immédiatement, mais seulement après quelques cycles-k parce que le filtre introduit un léger retard dans la sortie.

## Voir aussi

*slider16*, *slider16f*, *slider32*, *slider64*, *slider64f*, *slider8*, *slider8f*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06 de Csound.



# slider64

slider64 — Crée un banc de 64 numéros différents de messages de contrôle MIDI.

## Description

Crée un banc de 64 numéros différents de messages de contrôle MIDI.

## Syntaxe

```
i1,...,i64 slider64 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \  
    ictlnum64, imin64, imax64, init64, ifn64  
  
k1,...,k64 slider64 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \  
    ictlnum64, imin64, imax64, init64, ifn64
```

## Initialisation

*i1 ... i64* -- valeurs de sortie

*ichan* -- canal MIDI (1-16)

*ictlnum1 ... ictlnum64* -- numéro de contrôle MIDI (0-127)

*imin1 ... imin64* -- valeurs minimales pour chaque contrôleur

*imax1 ... imax64* -- maximum values for each controller

*init1 ... init64* -- valeur initiale pour chaque contrôleur

*ifn1 ... ifn64* -- table de fonction de conversion pour chaque contrôleur

## Exécution

*k1 ... k64* -- valeurs de sortie

*slider64* est un banc de contrôleurs MIDI, utile lorsque l'on utilise un mélangeur MIDI comme le Kawai MM-16 ou autres pour changer n'importe quel paramètre du son en . Les messages de contrôle MIDI arrivant sur le port d'entrée sont convertis pour entrer dans l'intervalle *iminN*, *imaxN*, et une valeur initiale peut être fixée. On peut aussi utiliser de manière facultative une table de fonction non interpolée avec une courbe de traduction personnalisée pour obtenir, par exemple, des courbes de réponse exponentielles.

Si l'on n'a pas besoin d'une table de traduction, on fixe la valeur de *ifnN* à 0, sinon, on donne à *ifnN* un numéro de table de fonction valide. Lorsque l'on utilise une table de traduction (si *ifnN* reçoit une valeur non nulle faisant référence à une table de fonction déjà allouée), la valeur de *initN* doit être égale à celle de *iminN* ou à celle de *imaxN*, sinon la valeur de sortie initiale sera différente de celle spécifiée dans l'argument *initN*.

*slider64* fournit un banc de 64 numéros différents de messages de contrôle MIDI.

Comme les arguments d'entrée et de sortie sont nombreux, on peut scinder la ligne en utilisant le caractère '\' (slash inversé) (nouveau dans la version 3.47) pour améliorer la lisibilité. L'utilisation de ces opcodes est considérablement plus efficace que celle de (*ctrl7* et *tonek*) séparés, lorsque l'on a besoin de plus de contrôleurs.

Dans la version au taux-i de *slider64*, il n'y a pas d'argument de valeur d'entrée initiale. La sortie est prise directement dans l'état courant du tableau interne de contrôleurs de Csound.

## Voir aussi

*s16b14*, *s32b14*, *slider16*, *slider16f*, *slider32*, *slider32f*, *slider64f* *slider8*, *slider8f*

## Crédits

Auteur: Gabriel Maldonado  
Italie  
Décember 1998

Nouveau dans la version 3.50 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# slider64f

slider64f — Crée un banc de 64 numéros différents de messages de contrôle MIDI, filtrés avant la sortie.

## Description

Crée un banc de 64 numéros différents de messages de contrôle MIDI, filtrés avant la sortie.

## Syntaxe

```
k1,...,k64 slider64f ichan, ictlnum1, imin1, imax1, init1, ifn1, \  
icutoff1,..., ictlnum64, imin64, imax64, init64, ifn64, icutoff64
```

## Initialisation

*ichan* -- canal MIDI (1-16)

*ictlnum1* ... *ictlnum64* -- numéro de contrôle MIDI (0-127)

*imin1* ... *imin64* -- valeurs minimales pour chaque contrôleur

*imax1* ... *imax64* -- valeurs maximales pour chaque contrôleur

*init1* ... *init64* -- valeur initiale pour chaque contrôleur

*ifn1* ... *ifn64* -- table de fonction de conversion pour chaque contrôleur

*icutoff1* ... *icutoff64* -- fréquence de coupure du filtre passe-bas pour chaque contrôleur

## Exécution

*k1* ... *k64* -- valeurs de sortie

*slider64f* est un banc de contrôleurs MIDI, utile lorsque l'on utilise un mélangeur MIDI comme le Kawai MM-16 ou autres pour changer n'importe quel paramètre du son en . Les messages de contrôle MIDI arrivant sur le port d'entrée sont convertis pour entrer dans l'intervalle *iminN*, *imaxN*, et une valeur initiale peut être fixée. On peut aussi utiliser de manière facultative une table de fonction non interpolée avec une courbe de traduction personnalisée pour obtenir, par exemple, des courbes de réponse exponentielles.

Si l'on n'a pas besoin d'une table de traduction, on fixe la valeur de *ifnN* à 0, sinon, on donne à *ifnN* un numéro de table de fonction valide. Lorsque l'on utilise une table de traduction (si *ifnN* reçoit une valeur non nulle faisant référence à une table de fonction déjà allouée), la valeur de *initN* doit être égale à celle de *iminN* ou à celle de *imaxN*, sinon la valeur de sortie initiale sera différente de celle spécifiée dans l'argument *initN*.

*slider64f* fournit un banc de 64 numéros différents de messages de contrôle MIDI. Il filtre le signal avant la sortie. Cela élimine les discontinuités dues à la basse résolution du MIDI (7 bit). La fréquence de coupure peut être réglée séparément pour chaque contrôleur (intervalle recommandé : 0.1 à 5 Hz).

Comme les arguments d'entrée et de sortie sont nombreux, on peut scinder la ligne en utilisant le caractère '\' (slash inversé) (nouveau dans la version 3.47) pour améliorer la lisibilité. L'utilisation de ces opcodes est considérablement plus efficace que celle de (*ctrl7* et *tonek*) séparés, lorsque l'on a besoin de plus de contrôleurs.



## Avertissement

Les opcodes *slider64f* ne sortent pas la valeur initiale immédiatement, mais seulement après quelques cycles-k parce que le filtre introduit un léger retard dans la sortie.

## Voir aussi

*s16b14, s32b14, slider16, slider16f, slider32, slider32f, slider64, slider8, slider8f*

## Crédits

Auteur: Gabriel Maldonado  
Italie  
Décember 1998

Nouveau dans la version 3.50 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# slider64table

slider64table — Enregistre un banc de 64 messages de contrôle MIDI différents dans une table.

## Description

Enregistre un banc de 64 messages de contrôle MIDI différents dans une table.

## Syntaxe

```
kflag slider64table ichan, ioutTable, ioffset, ictlnum1, imin1, \  
imax1, init1, ifn1, ... , ictlnum64, imin64, imax64, init64, ifn64
```

## Initialisation

*ichan* -- canal MIDI (1-16)

*ioutTable* -- numéro de la table qui contiendra la sortie

*ioffset* -- décalage dans la table de sortie. Zéro signifie que la sortie du premier contrôleur affectera le premier élément de la table. 10 signifie que la sortie du premier contrôleur affectera le onzième élément de la table.

*ictlnum1* ... *ictlnum64* -- numéro de contrôle MIDI (0-127)

*imin1* ... *imin64* -- valeurs minimales pour chaque contrôleur

*imax1* ... *imax64* -- valeurs maximales pour chaque contrôleur

*init1* ... *init64* -- valeur initiale pour chaque contrôleur

*ifn1* ... *ifn64* -- table de fonction de conversion pour chaque contrôleur

## Exécution

*kflag* -- un indicateur qui informe si un message de changement de contrôle dans le banc a été reçu. Dans ce cas, *kflag* est fixé à 1. Sinon il est fixé à 0.

*slider64table* est un banc de contrôleurs MIDI, utile lorsque l'on utilise un mélangeur MIDI comme le Kawai MM-16 ou autres pour changer n'importe quel paramètre du son en . Les messages de contrôle MIDI arrivant sur le port d'entrée sont convertis pour entrer dans l'intervalle *iminN*, *imaxN*, et une valeur initiale peut être fixée. On peut aussi utiliser de manière facultative une table de fonction non interpolée avec une courbe de traduction personnalisée pour obtenir, par exemple, des courbes de réponse exponentielles.

Si l'on n'a pas besoin d'une table de traduction, on fixe la valeur de *ifnN* à 0, sinon, on donne à *ifnN* un numéro de table de fonction valide. Lorsque l'on utilise une table de traduction (si *ifnN* reçoit une valeur non nulle faisant référence à une table de fonction déjà allouée), la valeur de *initN* doit être égale à celle de *iminN* ou à celle de *imaxN*, sinon la valeur de sortie initiale sera différente de celle spécifiée dans l'argument *initN*.

*slider64table* fournit un banc de 64 numéros différents de messages de contrôle MIDI.

Comme les arguments d'entrée et de sortie sont nombreux, on peut scinder la ligne en utilisant le caractère '\' (slash inversé) (nouveau dans la version 3.47) pour améliorer la lisibilité. L'utilisation de ces opcodes est considérablement plus efficace que celle de (*ctrl7* et *tonek*) séparés, lorsque l'on a besoin de plus de contrôleurs.

*slider64table* ressemble beaucoup à la famille des opcodes *slider64* et *sliderN* (voir leur notice pour plus d'information), à la différence que la sortie n'est pas stockée dans des variables de taux-k, mais dans une table dénotée par l'argument *ioutTable*. Il est possible de définir un indice de base afin d'utiliser la même table pour plus d'un banc de contrôleurs (ou pour un autre usage).

Il est possible d'utiliser cet opcode conjointement avec *FLslidBnk2Setk* et avec *FLslidBnk2*, ce qui permet de synchroniser la position des valeurs MIDI à la position des widgets valeurs FLTK de *FLslidBnk2*. Noter qu'il faut spécifier les mêmes valeurs de min/max et de réponse linéaire/exponentielle dans *sliderNtable* et dans *FLslidBnk2*. Il y a une exception si l'on utilise une réponse dans une table indexée au lieu d'une réponse lin/exp. Dans ce cas, afin d'obtenir un résultat utilisable, la réponse par table indexée et les valeurs min/max ne doivent être fixées que dans *FLslidBnk2*, alors que dans *sliderNtable*, il faut fixer une réponse linéaire, un minimum de zéro et un maximum de un dans tous les contrôleurs.

## Voir aussi

*slider16table*, *slider16tablef*, *slider32table*, *slider32tablef*, *slider64tablef* *slider8table*, *slider8tablef*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06 de Csound.

# slider64tablef

slider64tablef — Enregistre un banc de 64 messages de contrôle MIDI différents dans une table, filtrés avant la sortie.

## Description

Enregistre un banc de 64 messages de contrôle MIDI différents dans une table, filtrés avant la sortie.

## Syntaxe

```
kflag slider64tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \  
init1, ifn1, icutoff1, .... , ictlnum64, imin64, imax64, init64, ifn64, icutoff64
```

## Initialisation

*ichan* -- canal MIDI (1-16)

*ioutTable* -- numéro de la table qui contiendra la sortie

*ioffset* -- décalage dans la table de sortie. Zéro signifie que la sortie du premier contrôleur affectera le premier élément de la table. 10 signifie que la sortie du premier contrôleur affectera le onzième élément de la table.

*ictlnum1* ... *ictlnum64* -- numéro de contrôle MIDI (0-127)

*imin1* ... *imin64* -- valeurs minimales pour chaque contrôleur

*imax1* ... *imax64* -- valeurs maximales pour chaque contrôleur

*init1* ... *init64* -- valeur initiale pour chaque contrôleur

*ifn1* ... *ifn64* -- table de fonction de conversion pour chaque contrôleur

*icutoff1* ... *icutoff64* -- fréquence de coupure du filtre passe-bas pour chaque contrôleur

## Exécution

*kflag* -- un indicateur qui informe si un message de changement de contrôle dans le banc a été reçu. Dans ce cas, *kflag* est fixé à 1. Sinon il est fixé à 0.

*slider64tablef* est un banc de contrôleurs MIDI, utile lorsque l'on utilise un mélangeur MIDI comme le Kawai MM-16 ou autres pour changer n'importe quel paramètre du son en . Les messages de contrôle MIDI arrivant sur le port d'entrée sont convertis pour entrer dans l'intervalle *iminN*, *imaxN*, et une valeur initiale peut être fixée. On peut aussi utiliser de manière facultative une table de fonction non interpolée avec une courbe de traduction personnalisée pour obtenir, par exemple, des courbes de réponse exponentielles.

Si l'on n'a pas besoin d'une table de traduction, on fixe la valeur de *ifnN* à 0, sinon, on donne à *ifnN* un numéro de table de fonction valide. Lorsque l'on utilise une table de traduction (si *ifnN* reçoit une valeur non nulle faisant référence à une table de fonction déjà allouée), la valeur de *initN* doit être égale à celle de *iminN* ou à celle de *imaxN*, sinon la valeur de sortie initiale sera différente de celle spécifiée dans l'argument *initN*.

*slider64tablef* fournit un banc de 64 numéros différents de messages de contrôle MIDI. Il filtre le signal avant la sortie. Cela élimine les discontinuités dues à la basse résolution du MIDI (7 bit). La fréquence de coupure peut être réglée séparément pour chaque contrôleur (intervalle recommandé : 0.1 à 5 Hz).

Comme les arguments d'entrée et de sortie sont nombreux, on peut scinder la ligne en utilisant le caractère ``` (slash inversé) (nouveau dans la version 3.47) pour améliorer la lisibilité. L'utilisation de ces opcodes est considérablement plus efficace que celle de (*ctrl7* et *tonek*) séparés, lorsque l'on a besoin de plus de contrôleurs.

*slider64tablef* ressemble beaucoup à la famille des opcodes *slider64f* et *sliderNf* (voir leur notice pour plus d'information), à la différence que la sortie n'est pas stockée dans des variables de taux-k, mais dans une table dénotée par l'argument *ioutTable*. Il est possible de définir un indice de base afin d'utiliser la même table pour plus d'un banc de contrôleurs (ou pour un autre usage).

Il est possible d'utiliser cet opcode conjointement avec *FLslidBnk2Setk* et avec *FLslidBnk2*, ce qui permet de synchroniser la position des valeurs MIDI à la position des widgets valeurs FLTK de *FLslidBnk2*. Noter qu'il faut spécifier les mêmes valeurs de min/max et de réponse linéaire/exponentielle dans *sliderNtablef* et dans *FLslidBnk2*. Il y a une exception si l'on utilise une réponse dans une table indexée au lieu d'une réponse lin/exp. Dans ce cas, afin d'obtenir un résultat utilisable, la réponse par table indexée et les valeurs min/max ne doivent être fixées que dans *FLslidBnk2*, alors que dans *sliderNtablef*, il faut fixer une réponse linéaire, un minimum de zéro et un maximum de un dans tous les contrôleurs.



### Avertissement

Les opcodes *slider64tablef* ne sortent pas la valeur initiale immédiatement, mais seulement après quelques cycles-k parce que le filtre introduit un léger retard dans la sortie.

## Voir aussi

*slider16table*, *slider16tablef*, *slider32table*, *slider32tablef*, *slider64table*, *slider8table*, *slider8tablef*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06 de Csound.



# slider8

slider8 — Crée un banc de 8 numéros différents de messages de contrôle MIDI.

## Description

Crée un banc de 8 numéros différents de messages de contrôle MIDI.

## Syntaxe

```
i1,...,i8 slider8 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \  
    ictlnum8, imin8, imax8, init8, ifn8  
  
k1,...,k8 slider8 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \  
    ictlnum8, imin8, imax8, init8, ifn8
```

## Initialisation

*i1 ... i8* -- valeurs de sortie

*ichan* -- canal MIDI (1-16)

*ictlnum1 ... ictlnum8* -- numéro de contrôle MIDI (0-127)

*imin1 ... imin8* -- valeurs minimales pour chaque contrôleur

*imax1 ... imax8* -- valeurs maximales pour chaque contrôleur

*init1 ... init8* -- valeur initiale pour chaque contrôleur

*ifn1 ... ifn8* -- table de fonction de conversion pour chaque contrôleur

## Exécution

*k1 ... k8* -- valeurs de sortie

*slider8* est un banc de contrôleurs MIDI, utile lorsque l'on utilise un mélangeur MIDI comme le Kawai MM-16 ou autres pour changer n'importe quel paramètre du son en . Les messages de contrôle MIDI arrivant sur le port d'entrée sont convertis pour entrer dans l'intervalle *iminN*, *imaxN*, et une valeur initiale peut être fixée. On peut aussi utiliser de manière facultative une table de fonction non interpolée avec une courbe de traduction personnalisée pour obtenir, par exemple, des courbes de réponse exponentielles.

Si l'on n'a pas besoin d'une table de traduction, on fixe la valeur de *ifnN* à 0, sinon, on donne à *ifnN* un numéro de table de fonction valide. Lorsque l'on utilise une table de traduction (si *ifnN* reçoit une valeur non nulle faisant référence à une table de fonction déjà allouée), la valeur de *initN* doit être égale à celle de *iminN* ou à celle de *imaxN*, sinon la valeur de sortie initiale sera différente de celle spécifiée dans l'argument *initN*.

*slider8* fournit un banc de 8 numéros différents de messages de contrôle MIDI.

Comme les arguments d'entrée et de sortie sont nombreux, on peut scinder la ligne en utilisant le caractère '\' (slash inversé) (nouveau dans la version 3.47) pour améliorer la lisibilité. L'utilisation de ces opcodes est considérablement plus efficace que celle de (*ctrl7* et *tonek*) séparés, lorsque l'on a besoin de plus de contrôleurs.

Dans la version au taux-i de *slider8*, il n'y a pas d'argument de valeur d'entrée initiale. La sortie est prise directement dans l'état courant du tableau interne de contrôleurs de Csound.

## Voir aussi

*s16b14*, *s32b14*, *slider16*, *slider16f*, *slider32*, *slider32f*, *slider64*, *slider64f*, *slider8f*

## Crédits

Auteur: Gabriel Maldonado  
Italie  
Décember 1998

Nouveau dans la version 3.50 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# slider8f

slider8f — Crée un banc de 8 numéros différents de messages de contrôle MIDI, filtrés avant la sortie.

## Description

Crée un banc de 8 numéros différents de messages de contrôle MIDI, filtrés avant la sortie.

## Syntaxe

```
k1,...,k8 slider8f ichan, ictlnum1, imin1, imax1, init1, ifn1, icutoff1, \  
..., ictlnum8, imin8, imax8, init8, ifn8, icutoff8
```

## Initialisation

*ichan* -- canal MIDI (1-16)

*ictlnum1* ... *ictlnum8* -- numéro de contrôle MIDI (0-127)

*imin1* ... *imin8* -- valeurs minimales pour chaque contrôleur

*imax1* ... *imax8* -- valeurs maximales pour chaque contrôleur

*init1* ... *init8* -- valeur initiale pour chaque contrôleur

*ifn1* ... *ifn8* -- table de fonction de conversion pour chaque contrôleur

*icutoff1* ... *icutoff8* -- fréquence de coupure du filtre passe-bas pour chaque contrôleur

## Exécution

*k1* ... *k8* -- valeurs de sortie

*slider8f* est un banc de contrôleurs MIDI, utile lorsque l'on utilise un mélangeur MIDI comme le Kawai MM-16 ou autres pour changer n'importe quel paramètre du son en . Les messages de contrôle MIDI arrivant sur le port d'entrée sont convertis pour entrer dans l'intervalle *iminN*, *imaxN*, et une valeur initiale peut être fixée. On peut aussi utiliser de manière facultative une table de fonction non interpolée avec une courbe de traduction personnalisée pour obtenir, par exemple, des courbes de réponse exponentielles.

Si l'on n'a pas besoin d'une table de traduction, on fixe la valeur de *ifnN* à 0, sinon, on donne à *ifnN* un numéro de table de fonction valide. Lorsque l'on utilise une table de traduction (si *ifnN* reçoit une valeur non nulle faisant référence à une table de fonction déjà allouée), la valeur de *initN* doit être égale à celle de *iminN* ou à celle de *imaxN*, sinon la valeur de sortie initiale sera différente de celle spécifiée dans l'argument *initN*.

*slider8f* fournit un banc de 8 numéros différents de messages de contrôle MIDI. Il filtre le signal avant la sortie. Cela élimine les discontinuités dues à la basse résolution du MIDI (7 bit). La fréquence de coupure peut être réglée séparément pour chaque contrôleur (intervalle recommandé : 0.1 à 5 Hz).

Comme les arguments d'entrée et de sortie sont nombreux, on peut scinder la ligne en utilisant le caractère '\' (slash inversé) (nouveau dans la version 3.47) pour améliorer la lisibilité. L'utilisation de ces opcodes est considérablement plus efficace que celle de (*ctrl7* et *tonek*) séparés, lorsque l'on a besoin de plus de contrôleurs.



## Avertissement

Les opcodes *slider8f* ne sortent pas la valeur initiale immédiatement, mais seulement après quelques cycles-k parce que le filtre introduit un léger retard dans la sortie.

## Voir aussi

*s16b14, s32b14, slider16, slider16f, slider32, slider32f, slider64, slider64f, slider8*

## Crédits

Auteur: Gabriel Maldonado  
Italie  
Décember 1998

Nouveau dans la version 3.50 de Csound.

Merci à Rasmus Ekman pour avoir indiqué les intervalles corrects pour le canal MIDI et les numéros de contrôleur.

# slider8table

slider8table — Enregistre un banc de 8 messages de contrôle MIDI différents dans une table.

## Description

Enregistre un banc de 8 messages de contrôle MIDI différents dans une table.

## Syntaxe

```
kflag slider8table ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \  
init1, ifn1,..., ictlnum8, imin8, imax8, init8, ifn8
```

## Initialisation

*ichan* -- canal MIDI (1-16)

*ioutTable* -- numéro de la table qui contiendra la sortie

*ioffset* -- décalage dans la table de sortie. Zéro signifie que la sortie du premier contrôleur affectera le premier élément de la table. 10 signifie que la sortie du premier contrôleur affectera le onzième élément de la table.

*ictlnum1* ... *ictlnum8* -- numéro de contrôle MIDI (0-127)

*imin1* ... *imin8* -- valeurs minimales pour chaque contrôleur

*imax1* ... *imax8* -- valeurs maximales pour chaque contrôleur

*init1* ... *init8* -- valeur initiale pour chaque contrôleur

*ifn1* ... *ifn8* -- table de fonction de conversion pour chaque contrôleur

## Exécution

*kflag* -- un indicateur qui informe si un message de changement de contrôle dans le banc a été reçu. Dans ce cas, *kflag* est fixé à 1. Sinon il est fixé à 0.

*slider8table* est un banc de contrôleurs MIDI, utile lorsque l'on utilise un mélangeur MIDI comme le Kawai MM-16 ou autres pour changer n'importe quel paramètre du son en . Les messages de contrôle MIDI arrivant sur le port d'entrée sont convertis pour entrer dans l'intervalle *iminN*, *imaxN*, et une valeur initiale peut être fixée. On peut aussi utiliser de manière facultative une table de fonction non interpolée avec une courbe de traduction personnalisée pour obtenir, par exemple, des courbes de réponse exponentielles.

Si l'on n'a pas besoin d'une table de traduction, on fixe la valeur de *ifnN* à 0, sinon, on donne à *ifnN* un numéro de table de fonction valide. Lorsque l'on utilise une table de traduction (si *ifnN* reçoit une valeur non nulle faisant référence à une table de fonction déjà allouée), la valeur de *initN* doit être égale à celle de *iminN* ou à celle de *imaxN*, sinon la valeur de sortie initiale sera différente de celle spécifiée dans l'argument *initN*.

*slider8table* fournit un banc de 8 numéros différents de messages de contrôle MIDI.

Comme les arguments d'entrée et de sortie sont nombreux, on peut scinder la ligne en utilisant le caractère '\' (slash inversé) (nouveau dans la version 3.47) pour améliorer la lisibilité. L'utilisation de ces opcodes est considérablement plus efficace que celle de (*ctrl7* et *tonek*) séparés, lorsque l'on a besoin de plus de contrôleurs.

*slider8table* ressemble beaucoup à la famille des opcodes *slider8* et *sliderN* (voir leur notice pour plus d'information), à la différence que la sortie n'est pas stockée dans des variables de taux-k, mais dans une table dénotée par l'argument *ioutTable*. Il est possible de définir un indice de base afin d'utiliser la même table pour plus d'un banc de contrôleurs (ou pour un autre usage).

Il est possible d'utiliser cet opcode conjointement avec *FLslidBnk2Setk* et avec *FLslidBnk2*, ce qui permet de synchroniser la position des valeurs MIDI à la position des widgets valeurs FLTK de *FLslidBnk2*. Noter qu'il faut spécifier les mêmes valeurs de min/max et de réponse linéaire/exponentielle dans *sliderNtable* et dans *FLslidBnk2*. Il y a une exception si l'on utilise une réponse dans une table indexée au lieu d'une réponse lin/exp. Dans ce cas, afin d'obtenir un résultat utilisable, la réponse par table indexée et les valeurs min/max ne doivent être fixées que dans *FLslidBnk2*, alors que dans *sliderNtable*, il faut fixer une réponse linéaire, un minimum de zéro et un maximum de un dans tous les contrôleurs.

## Voir aussi

*slider16table*, *slider16tablef*, *slider32table*, *slider32tablef*, *slider64table*, *slider64tablef*, *slider8tabletablef*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06 de Csound.

# slider8tablef

slider8tablef — Enregistre un banc de 8 messages de contrôle MIDI différents dans une table, filtrés avant la sortie.

## Description

Enregistre un banc de 8 messages de contrôle MIDI différents dans une table, filtrés avant la sortie.

## Syntaxe

```
kflag slider8tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \  
init1, ifn1, icutoff1, .... , ictlnum8, imin8, imax8, init8, ifn8, icutoff8
```

## Initialisation

*ichan* -- canal MIDI (1-16)

*ioutTable* -- numéro de la table qui contiendra la sortie

*ioffset* -- décalage dans la table de sortie. Zéro signifie que la sortie du premier contrôleur affectera le premier élément de la table. 10 signifie que la sortie du premier contrôleur affectera le onzième élément de la table.

*ictlnum1* ... *ictlnum8* -- numéro de contrôle MIDI (0-127)

*imin1* ... *imin8* -- valeurs minimales pour chaque contrôleur

*imax1* ... *imax8* -- valeurs maximales pour chaque contrôleur

*init1* ... *init8* -- valeur initiale pour chaque contrôleur

*ifn1* ... *ifn8* -- table de fonction de conversion pour chaque contrôleur

*icutoff1* ... *icutoff8* -- fréquence de coupure du filtre passe-bas pour chaque contrôleur

## Exécution

*kflag* -- un indicateur qui informe si un message de changement de contrôle dans le banc a été reçu. Dans ce cas, *kflag* est fixé à 1. Sinon il est fixé à 0.

*slider8tablef* est un banc de contrôleurs MIDI, utile lorsque l'on utilise un mélangeur MIDI comme le Kawai MM-16 ou autres pour changer n'importe quel paramètre du son en . Les messages de contrôle MIDI arrivant sur le port d'entrée sont convertis pour entrer dans l'intervalle *iminN*, *imaxN*, et une valeur initiale peut être fixée. On peut aussi utiliser de manière facultative une table de fonction non interpolée avec une courbe de traduction personnalisée pour obtenir, par exemple, des courbes de réponse exponentielles.

Si l'on n'a pas besoin d'une table de traduction, on fixe la valeur de *ifnN* à 0, sinon, on donne à *ifnN* un numéro de table de fonction valide. Lorsque l'on utilise une table de traduction (si *ifnN* reçoit une valeur non nulle faisant référence à une table de fonction déjà allouée), la valeur de *initN* doit être égale à celle de *iminN* ou à celle de *imaxN*, sinon la valeur de sortie initiale sera différente de celle spécifiée dans l'argument *initN*.

*slider8tablef* fournit un banc de 8 numéros différents de messages de contrôle MIDI. Il filtre le signal avant la sortie. Cela élimine les discontinuités dues à la basse résolution du MIDI (7 bit). La fréquence de coupure peut être réglée séparément pour chaque contrôleur (intervalle recommandé : 0.1 à 5 Hz).

Comme les arguments d'entrée et de sortie sont nombreux, on peut scinder la ligne en utilisant le caractère `\` (slash inversé) (nouveau dans la version 3.47) pour améliorer la lisibilité. L'utilisation de ces opcodes est considérablement plus efficace que celle de (*ctrl7* et *tonek*) séparés, lorsque l'on a besoin de plus de contrôleurs.

*slider8tablef* ressemble beaucoup à la famille des opcodes *slider8f* et *sliderNf* (voir leur notice pour plus d'information), à la différence que la sortie n'est pas stockée dans des variables de taux-k, mais dans une table dénotée par l'argument *ioutTable*. Il est possible de définir un indice de base afin d'utiliser la même table pour plus d'un banc de contrôleurs (ou pour un autre usage).

Il est possible d'utiliser cet opcode conjointement avec *FLslidBnk2Setk* et avec *FLslidBnk2*, ce qui permet de synchroniser la position des valeurs MIDI à la position des widgets valeurs FLTK de *FLslidBnk2*. Noter qu'il faut spécifier les mêmes valeurs de min/max et de réponse linéaire/exponentielle dans *sliderNtablef* et dans *FLslidBnk2*. Il y a une exception si l'on utilise une réponse dans une table indexée au lieu d'une réponse lin/exp. Dans ce cas, afin d'obtenir un résultat utilisable, la réponse par table indexée et les valeurs min/max ne doivent être fixées que dans *FLslidBnk2*, alors que dans *sliderNtablef*, il faut fixer une réponse linéaire, un minimum de zéro et un maximum de un dans tous les contrôleurs.



### Avertissement

Les opcodes *slider8tablef* ne sortent pas la valeur initiale immédiatement, mais seulement après quelques cycles-k parce que le filtre introduit un léger retard dans la sortie.

## Voir aussi

*slider16table*, *slider16tablef*, *slider32table*, *slider32tablef*, *slider64table*, *slider64tablef*, *slider8table*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06 de Csound.



# sliderKawai

sliderKawai — Crée un banc de 16 numéros de messages de contrôle MIDI différents venant d'un mélangeur MIDI KAWAI MM-16.

## Description

Crée un banc de 16 numéros de messages de contrôle MIDI différents venant d'un mélangeur MIDI KAWAI MM-16.

## Syntaxe

```
k1, k2, ..., k16 sliderKawai imin1, imax1, init1, ifn1, \  
imin2, imax2, init2, ifn2, ..., imin16, imax16, init16, ifn16
```

## Initialisation

*imin1 ... imin16* -- valeurs minimales pour chaque contrôleur

*imax1 ... imax16* -- valeurs maximales pour chaque contrôleur

*init1 ... init16* -- valeur initiale pour chaque contrôleur

*ifn1 ... ifn16* -- table de fonction de conversion pour chaque contrôleur

## Exécution

*k1 ... k16* -- valeurs en sortie

L'opcode *sliderKawai* est équivalent à *slider16*, mais ses numéros de contrôleur et de canal (*ichan* et *ictlnum*) sont fixés dans le code afin d'obtenir une compatibilité rapide avec le mélangeur MIDI KAWAI MM-16. C'est l'appareil ne permet pas de changer le message MIDI associé à chaque réglette. Il ne peut fournir que le contrôle 7 pour chaque réglette sur un canal MIDI différent. Cet opcode permet d'assigner rapidement les 16 réglettes du mélangeur à 16 variables de taux-k de Csound.

## Voir aussi

*slider16, slider16f, slider32, slider32f, slider64, slider64f, slider8, slider8f*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06 de Csound.

# sndloop

sndloop — Une boucle de son avec contrôle de la hauteur.

## Description

Cet opcode enregistre l'entrée audio et la restitue dans une boucle avec une durée définie par l'utilisateur et un fondu enchainé. On peut également contrôler la hauteur de la boucle et sa lecture à l'envers.

## Syntaxe

```
asig, krec sndloop ain, kpitch, ktrig, idur, ifad
```

## Initialisation

*idur* -- durée de la boucle en secondes.

*ifad* -- durée du fondu enchainé en secondes.

## Exécution

*asig* -- signal de sortie.

*krec* -- signal d'activation de l'enregistrement, 1 lors de l'enregistrement, 0 sinon.

*kpitch* -- contrôle de la hauteur (rapport de transposition) ; avec des valeurs négatives, la boucle est jouée à l'envers.

*ktrig* -- signal de déclenchement : lorsqu'il vaut 0, le traitement est suspendu. Lorsqu'il change (*ktrig*  $\geq$  1), l'opcode commence à enregistrer jusqu'à ce que la mémoire de la boucle soit pleine. Puis il restitue ensuite le son en boucle jusqu'au prochain changement (*ktrig* = 0). Un autre enregistrement peut recommencer lorsque *ktrig*  $\geq$  1.

## Exemples

Voici un exemple de l'opcode `sndloop`. Il utilise le fichier *sndloop.csd* [examples/sndloop.csd].

### Exemple 921. Exemple de l'opcode `sndloop`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sndloop.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1

itrig = p4
asig diskin2 "beats.wav", 1, 0, 1 ;get the signal in, loop it
ktrig line 0, itrig, 1 ;when to trigger signal = p4
kpitch line 1.2, p3, .5 ;vary pitch of recorded signal
aout,krec sndloop asig, kpitch, ktrig, .4, 0.05 ;rec starts at p4 sec, for .4 secs and 0.05 crossfade
      printk2 krec ; prints the recording signal
      outs aout, aout

endin

</CsInstruments>
<CsScore>

i1 0 5 .5 ;trigger in seconds (=p4)
i1 + 5 .8
i1 + 10 1.2

e
</CsScore>
</CsoundSynthesizer>
```

L'exemple ci-dessus montre l'opération de base de *sndloop*. La hauteur peut-être contrôlée au taux-k, l'enregistrement commence dès que le signal de déclenchement est  $\geq 1$ . L'enregistrement peut recommencer en fixant la valeur du signal de déclenchement à 0 puis de nouveau à 1.

## Crédits

Auteur : Victor Lazzarini  
Avril 2005

Nouveau dans la version 5.00

# sndwarp

**sndwarp** — Lit un son mono échantillonné dans une table et lui applique une modification de durée et/ou de hauteur.

## Description

*sndwarp* lit des échantillons sonores dans une table et applique une modification de durée et/ou de hauteur. Les modifications du temps et de la fréquence sont indépendantes l'une de l'autre. Par exemple un son peut être ralenti en durée tout en étant transposé dans l'aigu !

Les arguments de taille de fenêtre et de chevauchement influent grandement sur le résultat et seront fixés par expérimentation. En général ils doivent être aussi petits que possible. Par exemple, on peut commencer avec *iwsiz*e=*sr*/10 et *ioverlap*=15. Essayer *irandw*=*iwsiz*e\*0,2. Si l'on peut arriver à ses fins avec moins de chevauchements, le programme sera plus rapide. Mais si ces dernières sont en nombre insuffisant, on peut entendre des fluctuations d'amplitude. L'algorithme réagit différemment selon le son en entrée et il n'y a pas de règle fixe adaptée à toutes les circonstances. Si l'on arrive à trouver les bons réglages, on peut obtenir d'excellents résultats.

## Syntaxe

```
ares [, ac] sndwarp xamp, xtimewarp, xresample, ifn1, ibeg, iwsiz, \  
irandw, ioverlap, ifn2, itimemode
```

## Initialisation

*ifn1* -- le numéro de la table contenant les échantillons qui seront traités par *sndwarp*. *GEN01* est le générateur de fonction approprié pour mémoriser les échantillons d'un fichier son pré-existant.

*ibeg* -- le temps en secondes à partir duquel commencera la lecture dans la table. Lorsque *itimemode* est différent de zéro, la valeur de *xtimewarp* est décalée de *ibeg*.

*iwsiz*e -- la taille en échantillons de la fenêtre utilisée dans l'algorithme de variation de la durée.

*irandw* -- la largeur de bande d'un générateur de nombres aléatoires. Les nombres aléatoires seront ajoutés à *iwsiz*e.

*ioverlap* -- détermine la densité de fenêtres se chevauchant.

*ifn2* -- une fonction qui fournit la forme de la fenêtre. On l'utilise habituellement pour créer une sorte de rampe qui part de zéro au début et qui y retourne à la fin de chaque fenêtre. Essayer d'utiliser une moitié de sinusoïde (c-à-d : f1 0 16384 9 .5 1 0) qui fonctionne plutôt bien. On peut utiliser d'autres formes.

## Exécution

*ares* -- l'unique canal de sortie du générateur unitaire *sndwarp*. *sndwarp* suppose que la table de fonction contenant le signal échantillonné est monophonique. *sndwarp* indexera la table avec un incrément d'un seul échantillon. Il faut ainsi remarquer que si l'on utilise un signal stéréo avec *sndwarp*, la durée et la hauteur seront altérées en conséquence.

*ac* (facultatif) -- une version mono-couche (pas de superpositions), et non fenêtrée du signal modifié en durée et/ou en hauteur. Elle est fournie afin de permettre de pondérer l'amplitude du signal de sortie, qui contient habituellement beaucoup de versions se chevauchant et fenêtrées du signal, avec une version

épurée du signal modifié en durée et en hauteur. Le traitement de *sndwarp* peut causer des variations notables en amplitude (en plus ou en moins), à cause de la différence de temps entre les superpositions lorsque la variation de durée est appliquée. Si on l'utilise avec une unité *balance*, *ac* permet d'améliorer grandement la qualité sonore.

*xamp* -- la valeur qui sert à pondérer l'amplitude (voir la note sur son utilisation avec *ac*).

*xtimewarp* -- détermine comment la durée du signal en entrée sera allongée ou raccourcie. Il y a deux manières d'utiliser cet argument selon la valeur donnée à *itimemode*. Si la valeur de *itimemode* est 0, *xtimewarp* changera l'échelle temporelle du son. Par exemple, une valeur de 2 doublera la durée du son. Si *itimemode* a une valeur non nulle, alors *xtimewarp* est utilisé comme un pointeur temporel de la même manière que dans *lpread* et dans *pvoc*. Un des exemples ci-dessous illustre cette possibilité. Dans les deux cas, la hauteur ne sera *pas* altérée par le traitement. La transposition de hauteur est effectuée indépendamment au moyen de *xresample*.

*xresample* -- le facteur de changement de la hauteur du son. Par exemple, une valeur de 2 produira un son une octave plus haut que l'original. La durée du son, quant à elle, ne sera *pas* modifiée.

## Exemples

## Exemples

Voici en exemple de l'opcode *sndwarp*. Il utilise le fichier *sndwarp.csd* [examples/sndwarp.csd].

### Exemple 922. Exemple de l'opcode *sndwarp*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sndwarp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1

ktimewarp line 0, p3, 2.7 ;length of "fox.wav"
kresample init 1 ;do not change pitch
ibeg = 0 ;start at beginning
iwsz = 4410 ;window size in samples with
irandw = 882 ;bandwidth of a random number generator
itimemode = 1 ;ktimewarp is "time" pointer
ioverlap = p4

asig sndwarp .5, ktimewarp, kresample, 1, ibeg, iwsz, irandw, ioverlap, 2, itimemode
outs asig, asig

endin
</CsInstruments>
```

```
<CsScore>
f 1 0 131072 1 "fox.wav" 0 0 0 ; audio file
f 2 0 1024 9 0.5 1 0 ; half of a sine wave

i 1 0 7 2 ;different overlaps
i 1 + 7 5
i 1 + 7 15
e

</CsScore>
</CsoundSynthesizer>
```

L'exemple ci-dessous montre un ralentissement du son stocké dans la table (*ifn1*). Pendant toute la durée de la note, le ralentissement s'intensifiera depuis l'original jusqu'à un son dix fois plus « lent » que l'original. Pendant ce temps, la hauteur montera progressivement d'une octave.

```
iwindfun = 1
isampfun = 2
ibeg = 0
iwindsize = 2000
iwindrand = 400
ioverlap = 10
awarp line 1, p3, 1
aresamp line 1, p3, 2
kenv line 1, p3, .1
asig sndwarp kenv, awarp, aresamp, isampfun, ibeg, iwindsize, iwindrand, ioverlap, iwindfun, 0
```

Voici maintenant un exemple utilisant *xtimewarp* comme pointeur temporel et la stéréophonie :

```
itimemode = 1
atime line 0, p3, 10
ar1, ar2 sndwarpst kenv, atime, aresamp, sampfun, ibeg, iwindsize, iwindrand, ioverlap, \
iwindfun, itimemode
```

Ci-dessus, *atime* avance le pointeur temporel utilisé dans *sndwarpst* de 0 à 10 sur toute la durée de la note. Si *p3* vaut 20 alors le son sera deux fois plus lent que l'original. Bien sûr, on peut utiliser une fonction plus complexe qu'une simple ligne droite pour contrôler le facteur temporel.

Maintenant le même exemple que ci-dessus mais en utilisant la fonction *balance* avec les sorties facultatives :

```
asig, acmp sndwarp 1, awarp, aresamp, isampfun, ibeg, iwindsize, iwindrand, ioverlap, iwindfun, itimemode
abal balance asig, acmp

asig1, asig2, acmp1, acmp2 sndwarpst 1, atime, aresamp, sampfun, ibeg, iwindsize, iwindrand, ioverlap, \
iwindfun, itimemode
abal1 balance asig1, acmp1
abal2 balance asig2, acmp2
```

Noter l'utilisation de l'unité *balance* dans les deux exemples ci-dessus. La sortie de *balance* peut ensuite être pondérée, enveloppée, envoyée à un *out* ou un *outs*, etc. Noter que les arguments d'amplitude de *sndwarp* et de *sndwarpst* valent « 1 » dans ces exemples. En pondérant le signal après son traitement par *sndwarp*, *abal*, *abal1*, et *abal2* contiendront des signaux ayant à peu près la même amplitude que le signal original traité par *sndwarp*. Il est ainsi plus facile de prédire les niveaux et d'éviter d'avoir des échantillons hors intervalle ou des valeurs d'échantillon trop petites.



## Conseil Supplémentaire

N'utilisez la version stéréo que si vous avez réellement besoin de traiter un fichier stéréo. Elle est sensiblement plus lente que la version mono et si vous utilisez la fonction *balance*, c'est

encore plus lent. Il n'y a aucun inconvénient à utiliser un *sndwarp* mono dans un orchestre stéréo puis d'envoyer le résultat à un ou aux deux canaux de la sortie stéréo.

## Voir aussi

*sndwarpst*

## Crédits

Auteur : Richard Karpen  
Seattle, WA USA  
1997

# sndwarpst

*sndwarpst* — Lit un son stéréo échantillonné dans une table et lui applique une modification de durée et/ou de hauteur.

## Description

*sndwarpst* lit des échantillons stéréo sonores dans une table et applique une modification de durée et/ou de hauteur. Les modifications du temps et de la fréquence sont indépendantes l'une de l'autre. Par exemple un son peut être ralenti en durée tout en étant transposé dans l'aigu !

Les arguments de taille de fenêtre et de chevauchement influent grandement sur le résultat et seront fixés par expérimentation. En général ils doivent être aussi petits que possible. Par exemple, on peut commencer avec *iwsz*=*sr*/10 et *ioverlap*=15. Essayer *irandw*=*iwsz*\*0,2. Si l'on peut arriver à ses fins avec moins de chevauchements, le programme sera plus rapide. Mais si ces dernières sont en nombre insuffisant, on peut entendre des fluctuations d'amplitude. L'algorithme réagit différemment selon le son en entrée et il n'y a pas de règle fixe adaptée à toutes les circonstances. Si l'on arrive à trouver les bons réglages, on peut obtenir d'excellents résultats.

## Syntaxe

```
ar1, ar2 [,ac1] [, ac2] sndwarpst xamp, xtimewarp, xresample, ifn1, \  
ibeg, iwsz, irandw, ioverlap, ifn2, itimemode
```

## Initialisation

*ifn1* -- le numéro de la table contenant les échantillons qui seront traités par *sndwarpst*. *GEN01* est le générateur de fonction approprié pour mémoriser les échantillons d'un fichier son pré-existant.

*ibeg* -- le temps en secondes à partir duquel commencera la lecture dans la table. Lorsque *itimemode* est différent de zéro, la valeur de *xtimewarp* est décalée de *ibeg*.

*iwsz* -- la taille en échantillons de la fenêtre utilisée dans l'algorithme de variation de la durée.

*irandw* -- la largeur de bande d'un générateur de nombres aléatoires. Les nombres aléatoires seront ajoutés à *iwsz*.

*ioverlap* -- détermine la densité de fenêtres se chevauchant.

*ifn2* -- une fonction qui fournit la forme de la fenêtre. On l'utilise habituellement pour créer une sorte de rampe qui part de zéro au début et qui y retourne à la fin de chaque fenêtre. Essayer d'utiliser une moitié de sinusoïde (c-à-d : f1 0 16384 9 .5 1 0) qui fonctionne plutôt bien. On peut utiliser d'autres formes.

## Exécution

*ar1, ar2* -- *ar1* et *ar2* sont les sorties stéréo (gauche et droite) de *sndwarpst*. *sndwarpst* suppose que la table de fonction contenant le signal échantillonné est stéréophonique. *sndwarpst* indexera la table avec un incrément de deux échantillons. Il faut ainsi remarquer que si l'on utilise un signal mono avec *sndwarpst*, la durée et la hauteur seront altérées en conséquence.

*ac1, ac2* -- *ac1* et *ac2* sont des versions mono-couche (pas de superpositions), et non fenêtrées du signal modifié en durée et/ou en hauteur. Elles sont fournies afin de permettre de pondérer l'amplitude du signal



de sortie, qui contient habituellement beaucoup de versions se chevauchant et fenêtrées du signal, avec une version épurée du signal modifié en durée et en hauteur. Le traitement de *sndwarpst* peut causer des variations notables en amplitude (en plus ou en moins), à cause de la différence de temps entre les superpositions lorsque la variation de durée est appliquée. Si on les utilise avec une unité *balance*, *ac1* et *ac2* permettent d'améliorer grandement la qualité sonore. Ils sont facultatifs mais il faut noter que la syntaxe exige la présence des deux arguments (utiliser les deux ou aucun). Un exemple de leur utilisation est donné ci-dessous.

*xamp* -- la valeur qui sert à pondérer l'amplitude (voir la note sur son utilisation avec *ac1* et *ac2*).

*xtimewarp* -- détermine comment la durée du signal en entrée sera allongée ou raccourcie. Il y a deux manières d'utiliser cet argument selon la valeur donnée à *itimemode*. Si la valeur de *itimemode* est 0, *xtimewarp* changera l'échelle temporelle du son. Par exemple, une valeur de 2 doublera la durée du son. Si *itimemode* a une valeur non nulle, alors *xtimewarp* est utilisé comme un pointeur temporel de la même manière que dans *lpread* et dans *pvoc*. Un des exemples ci-dessous illustre cette possibilité. Dans les deux cas, la hauteur ne sera *pas* altérée par le traitement. La transposition de hauteur est effectuée indépendamment au moyen de *xresample*.

*xresample* -- le facteur de changement de la hauteur du son. Par exemple, une valeur de 2 produira un son une octave plus haut que l'original. La durée du son, quant à elle, ne sera *pas* modifiée.

## Exemples

## Exemples

Voici un exemple de l'opcode *sndwarpst*. Il utilise le fichier *sndwarpst.csd* [examples/sndwarpst.csd].

### Exemple 923. Exemple de l'opcode *sndwarpst*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sndwarpst.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ktimewarp line 0, p3, 1 ;length of stereo file "kickroll.wav"
kresample init 1 ;playback at the normal speed
ibeg = 0
iwsiz = 4410
irandw = 441
ioverlap = p4
itimemode = 1 ; Use the ktimewarp parameter as a "time" pointer
```

```

aL, aR sndwarpst .3, ktimewarp, kresample, 1, ibeg, iwsiz, irandw, ioverlap, 2, itimemode
aL dcblock aL ;get rid of DC offsets for left channel &
aR dcblock aR ;right channel
outs aL, aR

endin
</CsInstruments>
<CsScore>
f 1 0 65536 1 "kickroll.wav" 0 0 0
f 2 0 16384 9 0.5 1 0 ;half of a sine wave

i 1 0 7 2 ;different overlaps
i 1 + 7 5
i 1 + 7 15
e
</CsScore>
</CsSoundSynthesizer>

```

L'exemple ci-dessous montre un ralentissement du son stocké dans la table (*ifn1*). Pendant toute la durée de la note, le ralentissement s'intensifiera depuis l'original jusqu'à un son dix fois plus « lent » que l'original. Pendant ce temps, la hauteur montera progressivement d'une octave.

```

iwindfun = 1
isampfun = 2
ibeg = 0
iwindsize = 2000
iwindrand = 400
ioverlap = 10
awarp line 1, p3, 1
aresamp line 1, p3, 2
kenv line 1, p3, .1
asig sndwarp kenv, awarp, aresamp, isampfun, ibeg, iwindsize, iwindrand, ioverlap, iwindfun, 0

```

Voici maintenant un exemple utilisant *xtimewarp* comme pointeur temporel et la stéréophonie :

```

itimemode = 1
atime line 0, p3, 10
ar1, ar2 sndwarpst kenv, atime, aresamp, sampfun, ibeg, iwindsize, iwindrand, ioverlap, \
iwindfun, itimemode

```

Ci-dessus, *atime* avance le pointeur temporel utilisé dans *sndwarpst* de 0 à 10 sur toute la durée de la note. Si *p3* vaut 20 alors le son sera deux fois plus lent que l'original. Bien sûr, on peut utiliser une fonction plus complexe qu'une simple ligne droite pour contrôler le facteur temporel.

Maintenant le même exemple que ci-dessus mais en utilisant la fonction *balance* avec les sorties facultatives :

```

asig,acmp sndwarp 1, awarp, aresamp, isampfun, ibeg, iwindsize, iwindrand, ioverlap, iwindfun, itimemode
abal balance asig, acmp

asig1,asig2,acmp1,acmp2 sndwarpst 1, atime, aresamp, sampfun, ibeg, iwindsize, iwindrand, ioverlap, \
iwindfun, itimemode

abal1 balance asig1, acmp1
abal2 balance asig2, acmp2

```

Noter l'utilisation de l'unité *balance* dans les deux exemples ci-dessus. La sortie de *balance* peut ensuite être pondérée, enveloppée, envoyée à un *out* ou un *outs*, etc. Noter que les arguments d'amplitude de *sndwarp* et de *sndwarpst* valent « 1 » dans ces exemples. En pondérant le signal après son traitement par *sndwarp*, *abal*, *abal1*, et *abal2* contiendront des signaux ayant à peu près la même amplitude que le signal original traité par *sndwarp*. Il est ainsi plus facile de prédire les niveaux et d'éviter d'avoir des échantillons hors intervalle ou des valeurs d'échantillon trop petites.



## Conseil Supplémentaire

N'utilisez la version stéréo que si vous avez réellement besoin de traiter un fichier stéréo. Elle est sensiblement plus lente que la version mono et si vous utilisez la fonction *balance*, c'est encore plus lent. Il n'y a aucun inconvénient à utiliser un *sndwarp* mono dans un orchestre stéréo puis d'envoyer le résultat à un ou aux deux canaux de la sortie stéréo.

## Voir aussi

*sndwarp*

## Crédits

Auteur : Richard Karpen  
Seattle, WA USA  
1997

# sockrecv

sockrecv — Reçoit des données d'autres processus en utilisant les protocoles de bas-niveau UDP et TCP.

## Description

Reçoit directement en utilisant le protocole UDP (*sockrecv* et *sockrecvs*) ou TCP (*strecv*) à travers un réseau. Les données ne sont sujettes à aucun encodage ou routage spécial. L'opcode *sockrecvs* reçoit un signal stéréo entrelacé.

## Syntaxe

```
asig sockrecv iport, ilength
ksig sockrecv iport, ilength
asigl, asigr sockrecvs iport, ilength
String sockrecv iport, ilength
asig[,kstate] strecv Sipaddr, iport
```

## Initialisation

*Sipaddr* -- une chaîne qui est l'adresse IP de l'émetteur au format standard sur 4 octets séparés par des points.

*iport* -- numéro du port utilisé pour la communication.

*ilength* -- longueur des paquets individuels dans la transmission UDP. Cette longueur doit être suffisamment petite pour entrer dans une seule MTU, dont la valeur enregistrée est 1456. Dans les transmissions UDP, l'émetteur et le récepteur doivent s'accorder sur la même valeur.

## Exécution

*asig*, *asigl*, *asigr* -- données audio à recevoir.

*ksig* -- données de contrôle à recevoir.

*String* -- données chaîne de caractères à recevoir.

*kstate* -- sortie facultative pour donner l'état de la réception. Donne le nombre d'octets transférés dans le cycle d'exécution courant ou -1 si l'émetteur a cessé d'écrire.

## Exemples

L'exemple montre un signal mono reçu sur le port 7777 en utilisant UDP.

```
sr = 44100
ksmps = 100
nchnls = 1
```

```
instr    1
al sockrecv  7777, 200
        out   al
endin
```

## Crédits

Auteur : John ffitch  
2006

La sortie facultative kstate est nouvelle dans la version 6.14

# socksend

socksend — Envoie des données à d'autres processus en utilisant les protocoles de bas-niveau UDP et TCP.

## Description

Transmet des données directement en utilisant le protocole UDP (*socksend* et *socksends*) ou TCP (*stsend*) à travers un réseau. Les données ne sont sujettes à aucun encodage ou routage spécial. L'opcode *socksends* envoie un signal stéréo entrelacé.

## Syntaxe

```
socksend asig, Sipaddr, iport, ilength
socksend ksig, Sipaddr, iport, ilength
socksends asigl, asigr, Sipaddr, iport,
    ilength
stsend asig, Sipaddr, iport
```

## Initialisation

*Sipaddr* -- une chaîne qui est l'adresse IP du récepteur au format standard sur 4 octets séparés par des points.

*iport* -- numéro du port utilisé pour la communication.

*ilength* -- longueur des paquets individuels dans la transmission UDP. Cette longueur doit être suffisamment petite pour entrer dans une seule MTU, dont la valeur enregistrée est 1456. Dans les transmissions UDP, le récepteur doit connaître cette valeur.

## Exécution

*asig*, *ksig*, *asigl*, *asigr* -- données à transmettre.

## Exemples

Cet exemple montre une simple onde sinus envoyée une seule fois à un ordinateur appelé "172.16.0.255" sur le port 7777 en utilisant UDP. Noter que .255 est souvent utilisé pour la diffusion.

```
sr = 44100
ksmps = 100
nchnls = 1

instr 1
a1 oscil      20000,441,1
  socksend    a1, "172.16.0.255",7777, 200
endin
```

## Crédits

Auteur : John ffitch

2006

# sorta

sorta — Trie un tableau en ordre ascendant.

## Description

Prend un tableau numérique (taux-k ou i) et le retourne trié en ordre ascendant.

## Syntaxe

```
k/i[]sorta k/i[] (k- or i-arrays )
```

## Exemples

Voici un exemple de l'opcode sorta. Il utilise le fichier *sorta.csd* [examples/sorta.csd].

### Exemple 924. Exemple de l'opcode sorta.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-n
</CsOptions>
<CsInstruments>

instr 1
  kArr[] fillarray 1,3,2,7,4
  kSorted[] sorta kArr
  kn = 0
  while kn < lenarray(kSorted) do
    printk2 kSorted[kn]
    kn += 1
  od
  turnoff
endin

</CsInstruments>
<CsScore>
i1 0 1
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*sortd*

## Crédits

Auteur : Victor Lazzarini

Nouveau dans la version 6.09



# sortd

sortd — Trie un tableau en ordre descendant.

## Description

Prend un tableau numérique (taux-k ou i) et le retourne trié en ordre descendant.

## Syntaxe

```
k/i[]sortd k/i[] (k- or i-arrays )
```

## Exemples

Voici un exemple de l'opcode sortd. Il utilise le fichier *sortd.csd* [examples/sortd.csd].

### Exemple 925. Exemple de l'opcode sortd.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-n
</CsOptions>
<CsInstruments>

instr 1
  kArr[] fillarray 1,3,2,7,4
  kSorted[] sortd kArr
  kn = 0
  while kn < lenarray(kSorted) do
    printk2 kSorted[kn]
    kn += 1
  od
  turnoff
endin

</CsInstruments>
<CsScore>
i1 0 1
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*sorta*

## Crédits

Auteur : Victor Lazzarini

Nouveau dans la version 6.09

# soundin

soundin — Lit des données audio mono depuis un périphérique externe ou un flot.

## Description

Lit des données audio mono depuis un périphérique externe ou un flot. On pouvait lire jusqu'à 24 canaux avant la version 5.14 de Csound, limite étendue à 40 canaux depuis lors.

## Syntaxe

```
ar1[, ar2[, ar3[, ... a24]]] soundin ifilcod [, iskptim] [, iformat] \  
[, iskipinit] [, ibufsize]
```

## Initialisation

*ifilcod* -- entier ou chaîne de caractères donnant le nom du fichier son source. Un entier indique le fichier soundin.filcod ; une chaîne de caractères (entre guillemets, espaces autorisés) donne le nom de fichier lui-même, éventuellement un nom de chemin complet. Si ce n'est pas un nom de chemin complet, le fichier nommé est d'abord cherché dans le répertoire courant, puis dans celui qui est donné par la variable d'environnement *SSDIR* (si elle est définie) puis par *SFDIR*. Voir aussi *GEN01*.

*iskptim* (facultatif, 0 par défaut) -- portion du son en entrée à ignorer, exprimée en secondes. La valeur par défaut est 0. A partir de Csound 5.00, cette valeur peut être négative ce qui ajoute un délai au lieu d'une portion à ignorer.

*iformat* (facultatif, 0 par défaut) -- spécifie le format des données audio du fichier :

- 1 = caractères signés sur 8 bit (les 8 bit de poids fort d'un entier sur 16 bit)
- 2 = octets sur 8 bit A-law
- 3 = octets sur 8 bit U-law
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit
- 7 = entiers non signés sur 8 bit (non disponible dans les versions de Csound antérieures à la 5.00)
- 8 = entiers sur 24 bit (non disponible dans les versions de Csound antérieures à la 5.00)
- 9 = doubles sur 64 bit (non disponible dans les versions de Csound antérieures à la 5.00)

*iskipinit* -- supprime toute initialisation s'il est non nul (vaut 0 par défaut). Fut introduit dans la version 4\_23f13 et dans csound5.

*ibufsize* -- taille du tampon en échantillons mono (pas en trames d'échantillons). N'est pas disponible dans les versions de Csound antérieures à la 5.00. La taille de tampon par défaut est 2048.

Si *iformat* = 0, il est déduit de l'en-tête du fichier, et s'il n'y a pas d'en-tête, de l'option de ligne de commande *-o* de Csound. La valeur par défaut est 0.

## Exécution

*soundin* est fonctionnellement un générateur audio dont le signal est dérivé d'un fichier pré-existant. Le nombre de canaux lus est contrôlé par le nombre de variables résultat, *a1*, *a2*, etc., qui doivent concorder avec ceux du fichier d'entrée. Un opcode *soundin* ouvre le fichier chaque fois que l'instrument le contenant est initialisé, puis il le ferme chaque fois que l'instrument est arrêté.

Il peut y avoir n'importe quel nombre d'opcodes *soundin* dans un instrument de l'orchestre. Plusieurs d'entre eux peuvent lire simultanément depuis le même fichier.



### Note pour les utilisateurs de Windows

Les utilisateurs de Windows utilisent normalement des anti-slash, « \ », pour spécifier les chemins de leurs fichiers. Par exemple un utilisateur de Windows pourra utiliser le chemin « c:\music\samples\loop001.wav ». Ceci pose problème car les anti-slash servent habituellement à spécifier des caractères spéciaux.

Pour spécifier correctement ce chemin dans Csound on peut utiliser :

- soit le *slash* : c:/music/samples/loop001.wav
- soit le *caractère spécial d'anti-slash*, « \\ » : c:\\music\\samples\\loop001.wav

## Exemples

Voici un exemple de l'opcode *soundin*. Il utilise les fichiers *soundin.csd* [examples/soundin.csd] *fox.wav* [examples/fox.wav] et *kickroll.wav* [examples/kickroll.wav].

### Exemple 926. Exemple de l'opcode *soundin*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
;-odac      ;;realtime audio out
-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime output leave only the line below:
;-o soundin.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; choose between mono or stereo file

ichn filenchnls p4 ;check number of channels
print ichn

if ichn == 1 then
asig soundin p4 ;mono signal
outs asig, asig
else ;stereo signal
aL, aR soundin p4
```

```
        outs      aL, aR
    endif

    endin
</CsInstruments>
<CsScore>

i 1 0 3 "fox.wav" ;mono signal
i 1 5 2 "kickroll.wav" ;stereo signal

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*diskin, in, inh, ino, inq, ins*

## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

Avertissement pour les utilisateurs de Windows ajouté par Kevin Conder, avril 2002

# space

*space* — Distribue un signal audio sur quatre canaux en utilisant des coordonnées cartésiennes.

## Description

*space* prend un signal en entrée et le distribue sur quatre canaux en utilisant les coordonnées cartésiennes *xy* pour calculer la balance des sorties. Les coordonnées *xy* peuvent être définies dans un fichier texte séparé et récupérées par le biais d'une instruction de fonction dans la partition en utilisant *Gen28*, ou bien on peut les spécifier au moyen des arguments facultatifs *kx*, *ky*. Les avantages de la première méthode sont :

1. On peut utiliser une interface graphique pour dessiner et éditer la trajectoire dans le plan cartésien.
2. Le format de fichier est de la forme temps1 X1 Y1 temps2 X2 Y2 temps3 X3 Y3 ce qui permet de définir une trajectoire paramétrée par le temps.

*space* permet ainsi de définir un pointeur temporel (du même type que ceux utilisés par *pvoc*, *lpread* et quelques autres unités) pour avoir un contrôle détaillé sur la vitesse résultante du mouvement.

## Syntaxe

*a1*, *a2*, *a3*, *a4* **space** *asig*, *ifn*, *ktime*, *kreverb**send*, *kx*, *ky*

## Initialisation

*ifn* -- numéro de la fonction créée au moyen de *Gen28*. Ce générateur de fonction lit un fichier texte qui contient des groupes de trois valeurs représentant les coordonnées *xy* et un paramètre de temps indiquant quand le signal doit être placé à cette position. Le fichier ressemblera à ceci :

```
0    -1    1
1     1    1
2     4    4
2.1  -4   -4
3    10  -10
5   -40    0
```

Avec un fichier nommé « move » l'appel à *Gen28* dans la partition s'écrira :

```
f1 0 0 28 "move"
```

*Gen28* prend pour taille 0 et alloue la mémoire automatiquement. Il crée ses valeurs avec une résolution de 10 ms. Ainsi dans ce cas, il y aura 500 valeurs créées en interpolant de X1 à X2 à X3 et ainsi de suite, et de Y1 à Y2 à Y3 et ainsi de suite, sur le nombre approprié de valeurs stockées dans la table de fonction. Dans l'exemple ci-dessus, le son démarre à l'avant-gauche, après une seconde il atteint l'avant-droite, après une autre seconde il est plus éloigné mais toujours à l'avant-droite, ensuite il bouge à l'arrière-gauche en moins d'un dixième de seconde, un peu éloigné. Enfin, durant les neuf dixièmes de seconde suivants le son bouge à l'arrière-droite, modérément distant, puis il se fixe entre les deux haut-parleurs de gauche (plein ouest !), assez éloigné. Comme les valeurs dans la table sont lues par l'unité *space* au moyen d'un pointeur temporel, le temps courant peut être réglé pour suivre exactement celui du fichier, ou bien il peut être réglé pour aller

plus vite ou plus lentement le long de la même trajectoire. Si l'on a accès à l'interface graphique permettant de dessiner et d'éditer les fichiers, il n'est pas nécessaire de créer les fichiers texte manuellement. Mais dès lors que le fichier est en ASCII dans le format ci-dessus, peu importe comment il a été créé !



### Important

Si *ifn* vaut 0, *space* obtient les valeurs des coordonnées *xy* depuis *kx* et *ky*.

## Exécution

La configuration des coordonnées *xy* dans l'espace place le signal de la manière suivante :

- *a1* est en (-1, 1)
- *a2* est en (1, 1)
- *a3* est en (-1, -1)
- *a4* est en (1, -1)

Ceci suppose une disposition des haut-parleurs où *a1* est à l'avant-gauche, *a2* à l'avant-droit, *a3* à l'arrière-gauche et *a4* à l'arrière-droit. Les valeurs supérieures à 1 donnent un son atténué, comme s'il était éloigné. *space* considère que les haut-parleurs sont à une distance de 1 ; on peut utiliser des valeurs de *xy* inférieures, mais *space* n'amplifiera pas le signal dans ce cas. Il équilibrera le signal cependant de manière à ce qu'il soit entendu comme s'il se trouvait à l'intérieur de l'espace des quatre haut-parleurs. *x=0, y=1*, place le signal entre les canaux avant gauche et droite, *x=y=0* place le signal également entre les quatre canaux, et ainsi de suite. Bien que *space* fournisse quatre signaux en sortie, on peut l'utiliser dans un orchestre à deux canaux. Si les *xy* sont tels que *y* reste  $\geq 1$ , il fonctionnera correctement pour faire des panoramiques et des localisations fixes dans un champ stéréophonique.

*asig* -- signal audio en entrée.

*ktime* -- indice dans la table contenant les coordonnées *xy*. S'il est utilisé comme ceci :

```
ktime      line 0, 5, 5
a1, a2, a3, a4 space asig, 1, ktime, ...
```

avec le fichier « move » décrit ci-dessus, la vitesse du mouvement du signal sera exactement celle qui est décrite dans ce fichier. Cependant avec :

```
ktime      line 0, 10, 5
```

le signal se déplacera deux fois moins vite qu'à la vitesse spécifiée. Ou dans le cas de :

```
ktime      line 5, 15, 0
```

le signal se déplacera dans la direction inverse et trois fois moins vite ! Enfin avec :

```
ktime      line 2, 10, 3
```

le signal ne se déplacera que depuis l'endroit spécifié à la ligne 3 du fichier texte jusqu'à l'endroit spécifié à la ligne 5 du fichier texte, et il lui faudra 10 secondes pour le faire.

*kreverb*send -- le pourcentage du signal direct qui sera combiné avec la distance déduite des coordonnées xy pour calculer la quantité de signal qui sera envoyée à des unités de réverbération comme *reverb* ou *reverb2*.

*kx*, *ky* -- lorsque *ifn* vaut 0, *space* et *spdist* utilisent ces valeurs comme coordonnées xy pour positionner le signal.

## Exemples

Voici un exemple de l'opcode *space*. Il utilise le fichier *space\_quad.csd* [examples/space\_quad.csd].

### Exemple 927. Exemple de l'opcode *space*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o space_quad.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 4

ga1 init 0
ga2 init 0
ga3 init 0
ga4 init 0

instr 1 ;uses GEN28 file "move", as found in /manual/examples

kx    init 0
ky    init 0
ktime line 0, 5, 5    ;same time as in table 1 ("move")
asig   diskin2 "beats.wav", 1, 0, 1 ;sound source is looped
a1, a2, a3, a4 space asig, 1, ktime, .1, kx, ky ;use table 1 = GEN28
ar1, ar2, ar3, ar4 spsend ;send to reverb

ga1 = ga1+ar1
ga2 = ga2+ar2
ga3 = ga3+ar3
ga4 = ga4+ar4
outq a1, a2, a3, a4

endin

instr 99 ; reverb instrument

a1 reverb2 ga1, 2.5, .5
a2 reverb2 ga2, 2.5, .5
a3 reverb2 ga3, 2.5, .5
a4 reverb2 ga4, 2.5, .5
outq a1, a2, a3, a4

ga1=0
ga2=0
```

```

ga3=0
ga4=0

endin
</CsInstruments>
<CsScore>
f1 0 0 28 "move"

i1 0 5 ;same time as ktime
i 99 0 10 ;keep reverb active
e
</CsScore>
</CsoundSynthesizer>

```

Dans l'exemple ci-dessus, le signal *asig* est déplacé selon les données dans la Fonction n°1 indexée par *ktime*. *space* envoie en interne la quantité appropriée du signal à *spsend*. Les sorties de *spsend* sont ajoutées à des accumulateurs globaux selon la manière habituelle dans Csound et les signaux globaux servent d'entrée aux unités de réverbération dans un instrument séparé.

*space* peut être utile pour une spatialisation quadro ou stéréo ainsi que pour le positionnement fixe des sons n'importe où entre deux haut-parleurs. Ci-dessous un exemple du positionnement fixe de sons dans un champ stéréo en utilisant des valeurs xy provenant de la partition au lieu d'une table de fonction. Il utilise le fichier *space\_stereo.csd* [examples/space\_stereo.csd].

### Exemple 928. Second exemple de l'opcode *space*.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o space_stereo.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

ga1 init 0
ga2 init 0

instr 1

kx = p4
ky = p5
asig diskin2 "beats.wav", 1
a1, a2, a3, a4 space asig, 0, 0, .1, kx, ky ;take position values from p4, p5
ar1, ar2, ar3, ar4 spsend ;send to reverb

ga1 = ga1+ar1
ga2 = ga2+ar2
outs a1, a2

endin

instr 99 ; reverb instrument

a1 reverb2 ga1, 2.5, .5
a2 reverb2 ga2, 2.5, .5
outs a1, a2

```



```
ga1=0
ga2=0

endin

</CsInstruments>
<CsScore>
;place the sound in the left speaker and near
i1 0 1 -1 1
;place the sound in the right speaker and far
i1 1 1 45 45
;place the sound equally between left and right and in the middle ground distance
i1 2 1 0 12

i 99 0 7 ;keep reverb active
e
</CsScore>
</CsoundSynthesizer>
```

*spdist* montre un exemple d'une simple utilisation intuitive des valeurs de distance pour simuler un effet Doppler.

## Voir aussi

*spdist*, *spsend*

## Crédits

Auteur : Richard Karpen  
Seattle, WA USA  
1998

Nouveau dans la version 3.48 de Csound.

# spat3d

spat3d — Positionne le son d'entrée dans un espace 3D et permet de déplacer le son au taux-k.

## Description

Cet opcode positionne le son d'entrée dans un espace 3D, avec simulation facultative d'un espace acoustique, dans différents formats de sortie. *spat3d* permet de déplacer le son au taux-k (ce mouvement est interpolé en interne pour éliminer le bruit de transition (« zipper noise ») si *sr* n'est pas égal à *kr*).

## Syntaxe

aW, aX, aY, aZ **spat3d** ain, kX, kY, kZ, idist, ift, imode, imdel, iovr [, istor]

## Initialisation

*idist* -- Pour les modes 0 à 3, *idist* est la distance du cercle unité en mètres. Pour le mode 4, *idist* est la distance entre les microphones.

Les formules suivantes décrivent l'amplitude et le retard comme une fonction de la distance entre la source sonore et le(s) microphone(s) :

amplitude =  $1 / (0.1 + \text{distance})$

delai = distance / 340 (en secondes)

La distance peut être calculée par :

distance =  $\sqrt{iX^2 + iY^2 + iZ^2}$

Dans le mode 4, la distance est calculée par :

distance au micro de gauche =  $\sqrt{(iX + idist/2)^2 + iY^2 + iZ^2}$

distance au micro de droite =  $\sqrt{(iX - idist/2)^2 + iY^2 + iZ^2}$

Avec *spat3d* la distance entre la source sonore et un microphone doit valoir au moins  $(340 * 18) / sr$  mètres. Les distances inférieures fonctionneront mais pourront produire des artefacts dans certains cas. Cette limitation n'existe pas pour *spat3di* et *spat3dt*.

Les changements brusques et les discontinuités dans le positionnement de la source sonore peuvent donner des pops ou des clics. Un mouvement très rapide peut aussi dégrader la qualité.

*ift* -- Table de fonction contenant les paramètres spatiaux (pour une spatialisation en milieu ouvert, mettre zéro ou une valeur négative). La taille de la table est 54. Les valeurs dans la table sont :

Paramètre Spatial	Fonction
0	Profondeur de récursion des premières réflexions (0 est la source sonore, 1 est la première réflexion, etc.) pour <i>spat3d</i> et <i>spat3di</i> . Le nombre d'échos pour quatre murs (avant, arrière, droit, gauche) est : $N =$

Paramètre Spatial	Fonction
	$(2 \cdot R + 2) \cdot R$ . Si les six murs sont pris en compte : $N = (((4 \cdot R + 6) \cdot R + 8) \cdot R) / 3$
1	Profondeur de récursion des réflexions secondaires (utilisé seulement par <i>spat3dt</i> ). <i>spat3dt</i> passe les premières réflexions et restitue des échos jusqu'à ce niveau. Si la profondeur des premières réflexions est négative, <i>spat3d</i> et <i>spat3di</i> donneront zéro en sortie, tandis que <i>spat3dt</i> commencera sa restitution depuis la source sonore.
2	<i>imdel</i> pour <i>spat3d</i> . Remplace le paramètre de l'opcode s'il est non négatif.
3	<i>irlen</i> pour <i>spat3dt</i> . Remplace le paramètre de l'opcode s'il est non négatif.
4	valeur de <i>idist</i> . Remplace le paramètre de l'opcode si elle est $\geq 0$ .
5	Graine aléatoire (0 - 65535). -1 fait prendre le temps courant comme graine.
6 - 53	paramètre de mur (w = 6 : plafond, w = 14 : plancher, w = 22 : avant, w = 30 : arrière, w = 38 : droite, w = 46 : gauche)
w + 0	Active les réflexions depuis ce mur (0 : non, 1 : oui)
w + 1	Distance entre le mur et l'auditeur (en mètres)
w + 2	Variation aléatoire de la distance du mur (0 à 1) (en unités de 1 / (distance du mur))
w + 3	Niveau de réflexion level (-1 à 1)
w + 4	Fréquence de l'égaliseur paramétrique en Hz.
w + 5	Niveau de l'égaliseur paramétrique (1.0 : pas de filtrage)
w + 6	Q de l'égaliseur paramétrique (0.7071 : pas de résonnance)
w + 7	Mode de l'égaliseur paramétrique (0 : peak EQ, 1 : low shelf, 2 : high shelf)

*imode* -- Mode de sortie.

- 0 : format B avec sortie W seulement (mono)

$aout = aW$

- 1 : format B avec sorties W et Y (stéréo)

$aleft = aW + 0.7071 \cdot aY$

$aright = aW - 0.7071 \cdot aY$

- 2 : format B avec sorties W, X et Y (2D). Peut être converti au format UHJ :

```

aWre, aWim    hilbert aW
aXre, aXim    hilbert aX
aYre, aYim    hilbert aY
aWXr  = 0.0928*aXre + 0.4699*aWre
aWXiYr = 0.2550*aXim - 0.1710*aWim + 0.3277*aYre
aleft  = aWXr + aWXiYr
aright = aWXr - aWXiYr

```

- 3 : format B avec toutes les sorties (3D)
- 4 : Simule une paire de microphones (sortie stéréo)

```

aW    butterlp aW, ifreq ; les valeurs recommandées pour ifreq
aY    butterlp aY, ifreq ; se situent autour de 1000 Hz
aleft = aW + aX
aright = aY + aZ

```

Le mode 0 est le moins couteux en capacité de calcul, tandis que le mode 4 est le plus gourmand.

Dans le mode 4, les filtres passe-bas facultatifs peuvent changer la réponse en fréquence en fonction de la direction. Par exemple, si la source sonore se situe à gauche de l'auditeur, les fréquences élevées sont atténuées dans le canal droit et légèrement augmentées dans le canal gauche. Si l'on utilise pas de filtre, cet effet n'a pas lieu. On peut expérimenter avec d'autres filtres (*tone*, etc.) pour un meilleur effet.

Noter que le mode 4 est plutôt destiné à une écoute au casque et qu'il est aussi plus coûteux en calcul que les modes du format B (0 à 3). Dans ce cas, le paramètre *idist* fixe la distance entre les microphones gauche et droit ; pour le casque, des valeurs comprises entre 0.2 et 0.25 sont recommandées, bien que l'on puisse utiliser des valeurs plus grandes, jusqu'à 0.4, pour des effets stéréo larges.

On peut trouver plus d'information sur le format B ici : [http://www.york.ac.uk/inst/mustech/3d\\_audio/am-bis2.htm](http://www.york.ac.uk/inst/mustech/3d_audio/am-bis2.htm)

*imdel* -- Retard maximum pour *spat3d* en secondes. Doit être plus long que le retard de la dernière réflexion (qui dépend des dimensions de la pièce, de la distance à la source et de la profondeur de récursion) ; la formule suivante donne une valeur sûre (bien que parfois surestimée) :

$$\text{imdel} = (R + 1) * \sqrt{W*W + H*H + D*D} / 340.0$$

où R est la profondeur de récursion, W, H et D sont respectivement la largeur, la hauteur et la profondeur de la pièce).

*iovr* -- Facteur de suréchantillonnage pour *spat3d* (1 à 8). Les valeurs supérieures augmentent la qualité au prix d'une consommation plus importante de la mémoire et du processeur. La valeur recommandée est 2.

*istor* (facultatif, 0 par défaut) -- S'il est différent de zéro, la phase d'initialisation est ignorée.

## Exécution

*aW*, *aX*, *aY*, *aZ* -- Signaux de sortie.

	mode 0	mode 1	mode 2	mode 3	mode 4
aW	sortie W	sortie W	sortie W	sortie W	canal gauche / basses fréq.
aX	0	0	sortie X	sortie X	canal gauche / hautes fréq.

	mode 0	mode 1	mode 2	mode 3	mode 4
aY	0	sortie Y	sortie Y	sortie Y	canal droit / basses fréq.
aZ	0	0	0	sortie Z	canal droit / hautes fréq.

*ain* -- Signal d'entrée.

*kX*, *kY*, *kZ* -- Coordonnées de la source sonore en (mètres).

Si l'on constate un fort ralentissement (jusqu'à 100 fois plus lent), la cause peut venir des nombres dénormalisés (nombres de magnitude trop faible). Ceci vaut aussi pour d'autres opcodes RII comme *butterlp*, *pareq*, *hilbert*, et bien d'autres. Ces déficits de capacité peuvent être évités en :

- utilisant l'opcode *denorm* sur *ain* avant *spat3d*.
- ajoutant au signal d'entrée une composante continue de faible niveau ou du bruit, par exemple  

```
atmp rnd31 1/1e24, 0, 0
```

```
aW, aX, aY, aZ spat3di ain + atmp, ...
```

ou  

```
aW, aX, aY, aZ spat3di ain + 1/1e24, ...
```
- réduisant *irlen* dans le cas de *spat3dt* (qui n'a pas de signal en entrée). Une valeur de 0.005 convient à la plupart des cas, bien que cela dépende aussi des réglages EQ. Si l'on utilise pas l'égaliseur, « *irlen* » peut rester être mis à 0.

## Exemples

Voici un exemple de l'opcode *spat3d* qui produit une sortie stéréo. Il utilise le fichier *spat3d\_stereo.csd* [examples/spat3d\_stereo.csd].

### Exemple 929. Exemple stéréo de l'opcode *spat3d*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o spat3d_stereo.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Istvan Varga */
sr      = 48000
kr      = 1000
ksmps   = 48
nchnls  = 2

/* room parameters */

idep     = 3      /* early reflection depth      */
```

```

itmp    ftgen    1, 0, 64, -2,
/* depth1, depth2, max delay, IR length, idist, seed */ \
idep, 48, -1, 0.01, 0.25, 123, \
1, 21.982, 0.05, 0.87, 4000.0, 0.6, 0.7, 2, /* ceil */ \
1, 1.753, 0.05, 0.87, 3500.0, 0.5, 0.7, 2, /* floor */ \
1, 15.220, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* front */ \
1, 9.317, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* back */ \
1, 17.545, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* right */ \
1, 12.156, 0.05, 0.87, 5000.0, 0.8, 0.7, 2 /* left */

instr 1

/* some source signal */

a1      phasor 150                ; oscillator
a1      butterbp a1, 500, 200    ; filter
a1      = taninv(a1 * 100)
a2      phasor 3                  ; envelope
a2      mirror 40*a2, -100, 5
a2      limit a2, 0, 1
a1      = a1 * a2 * 9000

kazim   line 0, 2.5, 360          ; move sound source around
kdist   line 1, 10, 4             ; distance

; convert polar coordinates
kX      = sin(kazim * 3.14159 / 180) * kdist
kY      = cos(kazim * 3.14159 / 180) * kdist
kZ      = 0

a1      = a1 + 0.000001 * 0.000001 ; avoid underflows

imode   = 1 ; change this to 3 for 8 spk in a cube,
; or 1 for simple stereo

aW, aX, aY, aZ spat3d a1, kX, kY, kZ, 1.0, 1, imode, 2, 2

aW      = aW * 1.4142

; stereo
;
aL      = aW + aY                /* left */
aR      = aW - aY                /* right */

; quad (square)
;
;aFL    = aW + aX + aY           /* front left */
;aFR    = aW + aX - aY           /* front right */
;aRL    = aW - aX + aY           /* rear left */
;aRR    = aW - aX - aY           /* rear right */

; eight channels (cube)
;
;aUFL   = aW + aX + aY + aZ      /* upper front left */
;aUFR   = aW + aX - aY + aZ      /* upper front right */
;aURL   = aW - aX + aY + aZ      /* upper rear left */
;aURR   = aW - aX - aY + aZ      /* upper rear right */
;aLFL   = aW + aX + aY - aZ      /* lower front left */
;aLFR   = aW + aX - aY - aZ      /* lower front right */
;aLRL   = aW - aX + aY - aZ      /* lower rear left */
;aLRR   = aW - aX - aY - aZ      /* lower rear right */

outs aL, aR

endin

```

```

</CsInstruments>
<CsScore>

/* Written by Istvan Varga */
i 1 0 10
e

</CsScore>
</CsoundSynthesizer>

```

Voici un exemple de l'opcode spat3d qui produit une sortie UHJ. Il utilise le fichier *spat3d\_UHJ.csd* [examples/spat3d\_UHJ.csd].

### Exemple 930. Exemple UHJ de l'opcode spat3d.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o spat3d_UHJ.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Istvan Varga */
sr = 48000
kr = 750
ksmps = 64
nchnls = 2

itmp      ftgen      1, 0, 64, -2,
/* depth1, depth2, max delay, IR length, idist, seed */ \
3, 48, -1, 0.01, 0.25, 123, \
1, 21.982, 0.05, 0.87, 4000.0, 0.6, 0.7, 2, /* ceil */ \
1, 1.753, 0.05, 0.87, 3500.0, 0.5, 0.7, 2, /* floor */ \
1, 15.220, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* front */ \
1, 9.317, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* back */ \
1, 17.545, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* right */ \
1, 12.156, 0.05, 0.87, 5000.0, 0.8, 0.7, 2 /* left */

instr 1

p3 = p3 + 1.0

kazim line 0.0, 4.0, 360.0 ; azimuth
kelev line 40, p3 - 1.0, -20 ; elevation
kdist = 2.0 ; distance
; convert coordinates
kX = kdist * cos(kelev * 0.01745329) * sin(kazim * 0.01745329)
kY = kdist * cos(kelev * 0.01745329) * cos(kazim * 0.01745329)
kZ = kdist * sin(kelev * 0.01745329)

; source signal
a1 phasor 160.0
a2 delay1 a1
a1 = a1 - a2
kffrq1 port 200.0, 0.8, 12000.0
affrq upsamp kffrq1
affrq pareq affrq, 5.0, 0.0, 1.0, 2
kffrq downsamp affrq
aenv4 phasor 3.0
aenv4 limit 2.0 - aenv4 * 8.0, 0.0, 1.0

```

```

a1 butterbp a1 * aenv4, kffrq, 160.0
aenv linseg 1.0, p3 - 1.0, 1.0, 0.04, 0.0, 1.0, 0.0
a_ = 4000000 * a1 * aenv + 0.00000001

; spatialize
a_W, a_X, a_Y, a_Z spat3d a_, kX, kY, kZ, 1.0, 1, 2, 2.0, 2

; convert to UHJ format (stereo)
aWre, aWim hilbert a_W
aXre, aXim hilbert a_X
aYre, aYim hilbert a_Y

aWXre = 0.0928*aXre + 0.4699*aWre
aWXim = 0.2550*aXim - 0.1710*aWim

aL = aWXre + aWXim + 0.3277*aYre
aR = aWXre - aWXim - 0.3277*aYre

outs aL, aR

endin

</CsInstruments>
<CsScore>

/* Written by Istvan Varga */
t 0 60

i 1 0.0 8.0
e

</CsScore>
</CsoundSynthesizer>

```

Voici un exemple de l'opcode spat3d qui produit une sortie quadraphonique. Il utilise le fichier *spat3d\_quad.csd* [examples/spat3d\_quad.csd].

### Exemple 931. Exemple quadraphonique de l'opcode spat3d.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o spat3d_quad.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

/* Written by Istvan Varga */
sr      = 48000
kr      = 1000
ksmps   = 48
nchnls  = 4

/* room parameters */

idep     = 3      /* early reflection depth      */

itmp     ftgen    1, 0, 64, -2, \
/* depth1, depth2, max delay, IR length, idist, seed */ \
idep, 48, -1, 0.01, 0.25, 123, \
1, 21.982, 0.05, 0.87, 4000.0, 0.6, 0.7, 2, /* ceil */ \

```



```

1, 1.753, 0.05, 0.87, 3500.0, 0.5, 0.7, 2, /* floor */ \
1, 15.220, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* front */ \
1, 9.317, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* back */ \
1, 17.545, 0.05, 0.87, 5000.0, 0.8, 0.7, 2, /* right */ \
1, 12.156, 0.05, 0.87, 5000.0, 0.8, 0.7, 2 /* left */

instr 1

/* some source signal */

a1      phasor 150          ; oscillator
a1      butterbp a1, 500, 200 ; filter
a1      = taninv(a1 * 100)
a2      phasor 3           ; envelope
a2      mirror 40*a2, -100, 5
a2      limit a2, 0, 1
a1      = a1 * a2 * 9000

kazim   line 0, 2.5, 360    ; move sound source around
kdist   line 1, 10, 4      ; distance

; convert polar coordinates
kX      = sin(kazim * 3.14159 / 180) * kdist
kY      = cos(kazim * 3.14159 / 180) * kdist
kZ      = 0

a1      = a1 + 0.000001 * 0.000001 ; avoid underflows

imode   = 2 ; change this to 3 for 8 spk in a cube,
         ; or 1 for simple stereo

aW, aX, aY, aZ spat3d a1, kX, kY, kZ, 1.0, 1, imode, 2, 2

aW      = aW * 1.4142

; stereo
;
;aL      = aW + aY          /* left          */
;aR      = aW - aY          /* right         */

; quad (square)
;
aFL      = aW + aX + aY      /* front left    */
aFR      = aW + aX - aY      /* front right   */
aRL      = aW - aX + aY      /* rear left     */
aRR      = aW - aX - aY      /* rear right    */

; eight channels (cube)
;
;aUFL    = aW + aX + aY + aZ /* upper front left */
;aUFR    = aW + aX - aY + aZ /* upper front right */
;aURL    = aW - aX + aY + aZ /* upper rear left  */
;aURR    = aW - aX - aY + aZ /* upper rear right  */
;aLFL    = aW + aX + aY - aZ /* lower front left  */
;aLFR    = aW + aX - aY - aZ /* lower front right */
;aLRL    = aW - aX + aY - aZ /* lower rear left   */
;aLRR    = aW - aX - aY - aZ /* lower rear right  */

outq aFL, aFR, aRL, aRR

endin

</CsInstruments>
<CsScore>

/* Written by Istvan Varga */

```

```
t 0 60  
i 1 0 10  
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*spat3di, spat3dt*

## Crédits

Auteur : Istvan Varga  
2001

Nouveau dans la version 4.12

Mis à jour en avril 2002 par Istvan Varga

# spat3di

spat3di — Positionne le son d'entrée dans un espace 3D en fixant la position de la source au temps-i.

## Description

Cet opcode positionne le son d'entrée dans un espace 3D, avec simulation facultative d'un espace acoustique, dans différents formats de sortie. Avec *spat3di*, la position de la source sonore est fixée au temps-i.

## Syntaxe

```
aW, aX, aY, aZ spat3di ain, iX, iY, iZ, idist, ift, imode [, istor]
```

## Initialisation

*iX* -- Coordonnée X de la source sonore en mètres (positive : à droite, négative : à gauche).

*iY* -- Coordonnée Y de la source sonore en mètres (positive : en avant, négative : en arrière).

*iZ* -- Coordonnée Z de la source sonore en mètres (positive : en haut, négative : en bas).

*idist* -- Pour les modes 0 à 3, *idist* est la distance du cercle unité en mètres. Pour le mode 4, *idist* est la distance entre les microphones.

Les formules suivantes décrivent l'amplitude et le retard comme une fonction de la distance entre la source sonore et le(s) microphone(s) :

$$\text{amplitude} = 1 / (0.1 + \text{distance})$$
$$\text{delay} = \text{distance} / 340 \text{ (en secondes)}$$

La distance peut être calculée par :

$$\text{distance} = \sqrt{iX^2 + iY^2 + iZ^2}$$

Dans le mode 4, la distance est calculée par :

$$\text{distance from left mic} = \sqrt{(iX + idist/2)^2 + iY^2 + iZ^2}$$
$$\text{distance from right mic} = \sqrt{(iX - idist/2)^2 + iY^2 + iZ^2}$$

Avec *spat3d* la distance entre la source sonore et un microphone doit valoir au moins  $(340 * 18) / sr$  mètres. Les distances inférieures fonctionneront mais pourront produire des artefacts dans certains cas. Cette limitation n'existe pas pour *spat3di* et *spat3dt*.

Les changements brusques et les discontinuités dans le positionnement de la source sonore peuvent donner des pops ou des clics. Un mouvement très rapide peut aussi dégrader la qualité.

*ift* -- Table de fonction contenant les paramètres spatiaux (pour une spatialisation en milieu ouvert, mettre zéro ou une valeur négative). La taille de la table est 54. Les valeurs dans la table sont :

Paramètre Spatial	Fonction
0	Profondeur de récursion des premières réflexions (0 est la source sonore, 1 est la première réflexion, etc.) pour <i>spat3d</i> et <i>spat3di</i> . Le nombre d'échos pour quatre murs (avant, arrière, droit, gauche) est : $N = (2 * R + 2) * R$ . Si les six murs sont pris en compte : $N = (((4 * R + 6) * R + 8) * R) / 3$
1	Profondeur de récursion des réflexions secondaires (utilisé seulement par <i>spat3dt</i> ). <i>spat3dt</i> passe les premières réflexions et restitue des échos jusqu'à ce niveau. Si la profondeur des premières réflexions est négative, <i>spat3d</i> et <i>spat3di</i> donneront zéro en sortie, tandis que <i>spat3dt</i> commencera sa restitution depuis la source sonore.
2	<i>imdel</i> pour <i>spat3d</i> . Remplace le paramètre de l'opcode s'il est non négatif.
3	<i>irlen</i> pour <i>spat3dt</i> . Remplace le paramètre de l'opcode s'il est non négatif.
4	valeur de <i>idist</i> . Remplace le paramètre de l'opcode si elle est $\geq 0$ .
5	Graine aléatoire (0 - 65535). -1 fait prendre le temps courant comme graine.
6 - 53	paramètre de mur (w = 6 : plafond, w = 14 : plancher, w = 22 : avant, w = 30 : arrière, w = 38 : droite, w = 46 : gauche)
w + 0	Active les réflexions depuis ce mur (0 : non, 1 : oui)
w + 1	Distance entre le mur et l'auditeur (en mètres)
w + 2	Variation aléatoire de la distance du mur (0 à 1) (en unités de 1 / (distance du mur))
w + 3	Niveau de réflexion level (-1 à 1)
w + 4	Fréquence de l'égaliseur paramétrique en Hz.
w + 5	Niveau de l'égaliseur paramétrique (1.0 : pas de filtrage)
w + 6	Q de l'égaliseur paramétrique (0.7071 : pas de résonnance)
w + 7	Mode de l'égaliseur paramétrique (0 : peak EQ, 1 : low shelf, 2 : high shelf)

*imode* -- Mode de sortie.

- 0 : format B avec sortie W seulement (mono)

aout = aW

- 1 : format B avec sorties W et Y (stéréo)

aleft = aW + 0.7071\*aY

$a_{right} = aW - 0.7071 * aY$

- 2 : format B avec sorties W, X et Y (2D). Peut être converti au format UHJ :

```

aWre, aWim    hilbert aW
aXre, aXim    hilbert aX
aYre, aYim    hilbert aY
aWXr  = 0.0928*aXre + 0.4699*aWre
aWXiYr = 0.2550*aXim - 0.1710*aWim + 0.3277*aYre
aleft  = aWXr + aWXiYr
aright = aWXr - aWXiYr

```

- 3 : format B avec toutes les sorties (3D)
- 4 : Simule une paire de microphones (sortie stéréo)

```

aW    butterlp aW, ifreq ; les valeurs recommandées pour ifreq
aY    butterlp aY, ifreq ; se situent autour de 1000 Hz
aleft = aW + aX
aright = aY + aZ

```

Le mode 0 est le moins couteux en capacité de calcul, tandis que le mode 4 est le plus gourmand.

Dans le mode 4, les filtres passe-bas facultatifs peuvent changer la réponse en fréquence en fonction de la direction. Par exemple, si la source sonore se situe à gauche de l'auditeur, les fréquences élevées sont atténuées dans le canal droit et légèrement augmentées dans le canal gauche. Si l'on utilise pas de filtre, cet effet n'a pas lieu. On peut expérimenter avec d'autres filtres (*tone*, etc.) pour un meilleur effet.

Noter que le mode 4 est plutôt destiné à une écoute au casque et qu'il est aussi plus coûteux en calcul que les modes du format B (0 à 3). Dans ce cas, le paramètre *idist* fixe la distance entre les microphones gauche et droit ; pour le casque, des valeurs comprises entre 0.2 et 0.25 sont recommandées, bien que l'on puisse utiliser des valeurs plus grandes, jusqu'à 0.4, pour des effets stéréo larges.

On peut trouver plus d'information sur le format B ici : [http://www.york.ac.uk/inst/mustech/3d\\_audio/am-bis2.htm](http://www.york.ac.uk/inst/mustech/3d_audio/am-bis2.htm)

*istor* (facultatif, 0 par défaut) -- S'il est différent de zéro, la phase d'initialisation est ignorée.

## Exécution

*ain* -- Signal d'entrée.

*aW, aX, aY, aZ* -- Signaux de sortie.

	mode 0	mode 1	mode 2	mode 3	mode 4
aW	sortie W	sortie W	sortie W	sortie W	canal gauche / basses fréq.
aX	0	0	sortie X	sortie X	canal gauche / hautes fréq.
aY	0	sortie Y	sortie Y	sortie Y	canal droit / basses fréq.

	mode 0	mode 1	mode 2	mode 3	mode 4
aZ	0	0	0	sortie Z	canal droit / hautes fréq.

Si l'on constate un fort ralentissement (jusqu'à 100 fois plus lent), la cause peut venir des nombres dénormalisés (nombres de magnitude trop faible). Ceci vaut aussi pour d'autres opcodes RII comme *butterlp*, *pareq*, *hilbert*, et bien d'autres. Ces déficits de capacité peuvent être évités en :

- utilisant l'opcode *denorm* sur *ain* avant *spat3d*.
- ajoutant au signal d'entrée une composante continue de faible niveau ou du bruit, par exemple  
  
atmp rnd31 1/1e24, 0, 0  
  
aW, aX, aY, aZ spa3di ain + atmp, ...  
  
ou  
  
aW, aX, aY, aZ spa3di ain + 1/1e24, ...
- réduisant *irlen* dans le cas de *spat3dt* (qui n'a pas de signal en entrée). Une valeur de 0.005 convient à la plupart des cas, bien que cela dépende aussi des réglages EQ. Si l'on utilise pas l'égaliseur, « irlen » peut rester être mis à 0.

## Exemples

Voir les exemples de *spat3d*.

## Voir aussi

*spat3d*, *spat3dt*

## Crédits

Auteur : Istvan Varga  
2001

Nouveau dans la version 4.12

Mis à jour en avril 2002 par Istvan Varga

# spat3dt

spat3dt — Utilisable pour obtenir une réponse impulsionnelle dans un espace 3D au temps-i.

## Description

Cet opcode positionne le son d'entrée dans un espace 3D, avec simulation facultative d'un espace acoustique, dans différents formats de sortie. *spat3dt* peut être utilisé pour obtenir la réponse impulsionnelle au temps-i, stockant sa sortie dans une table de fonction, convenant à la convolution.

## Syntaxe

```
spat3dt ioutft, iX, iY, iZ, idist, ift, imode, irlen [, iftnocl]
```

## Initialisation

*ioutft* -- Numéro de la ftable de sortie. Les sorties W, X, Y et Z sont écrites de manière entrelacée dans cette table. Si la table est trop courte, la sortie est tronquée.

*iX* -- Coordonnée X de la source sonore en mètres (positive : à droite, négative : à gauche).

*iY* -- Coordonnée Y de la source sonore en mètres (positive : en avant, négative : en arrière).

*iZ* -- Coordonnée Z de la source sonore en mètres (positive : en haut, négative : en bas).

*idist* -- Pour les modes 0 à 3, *idist* est la distance du cercle unité en mètres. Pour le mode 4, *idist* est la distance entre les microphones.

Les formules suivantes décrivent l'amplitude et le retard comme une fonction de la distance entre la source sonore et le(s) microphone(s) :

amplitude =  $1 / (0.1 + \text{distance})$

delay = distance / 340 (en secondes)

La distance peut être calculée par :

distance =  $\sqrt{iX^2 + iY^2 + iZ^2}$

Dans le mode 4, la distance est calculée par :

distance from left mic =  $\sqrt{(iX + idist/2)^2 + iY^2 + iZ^2}$

distance from right mic =  $\sqrt{(iX - idist/2)^2 + iY^2 + iZ^2}$

Avec *spat3d* la distance entre la source sonore et un microphone doit valoir au moins  $(340 * 18) / sr$  mètres. Les distances inférieures fonctionneront mais pourront produire des artefacts dans certains cas. Cette limitation n'existe pas pour *spat3di* et *spat3dt*.

Les changements brusques et les discontinuités dans le positionnement de la source sonore peuvent donner des pops ou des clics. Un mouvement très rapide peut aussi dégrader la qualité.

*ift* -- Table de fonction contenant les paramètres spatiaux (pour une spatialisation en milieu ouvert, mettre zéro ou une valeur négative). La taille de la table est 54. Les valeurs dans la table sont :

Paramètre Spatial	Fonction
0	Profondeur de récursion des premières réflexions (0 est la source sonore, 1 est la première réflexion, etc.) pour <i>spat3d</i> et <i>spat3di</i> . Le nombre d'échos pour quatre murs (avant, arrière, droit, gauche) est : $N = (2 \cdot R + 2) \cdot R$ . Si les six murs sont pris en compte : $N = (((4 \cdot R + 6) \cdot R + 8) \cdot R) / 3$
1	Profondeur de récursion des réflexions secondaires (utilisé seulement par <i>spat3dt</i> ). <i>spat3dt</i> passe les premières réflexions et restitue des échos jusqu'à ce niveau. Si la profondeur des premières réflexions est négative, <i>spat3d</i> et <i>spat3di</i> donneront zéro en sortie, tandis que <i>spat3dt</i> commencera sa restitution depuis la source sonore.
2	<i>imdel</i> pour <i>spat3d</i> . Remplace le paramètre de l'opcode s'il est non négatif.
3	<i>irlen</i> pour <i>spat3dt</i> . Remplace le paramètre de l'opcode s'il est non négatif.
4	valeur de <i>idist</i> . Remplace le paramètre de l'opcode si elle est $\geq 0$ .
5	Graine aléatoire (0 - 65535). -1 fait prendre le temps courant comme graine.
6 - 53	paramètre de mur (w = 6 : plafond, w = 14 : plancher, w = 22 : avant, w = 30 : arrière, w = 38 : droite, w = 46 : gauche)
w + 0	Active les réflexions depuis ce mur (0 : non, 1 : oui)
w + 1	Distance entre le mur et l'auditeur (en mètres)
w + 2	Variation aléatoire de la distance du mur (0 à 1) (en unités de 1 / (distance du mur))
w + 3	Niveau de réflexion level (-1 à 1)
w + 4	Fréquence de l'égaliseur paramétrique en Hz.
w + 5	Niveau de l'égaliseur paramétrique (1.0 : pas de filtrage)
w + 6	Q de l'égaliseur paramétrique (0.7071 : pas de résonance)
w + 7	Mode de l'égaliseur paramétrique (0 : peak EQ, 1 : low shelf, 2 : high shelf)

*imode* -- Mode de sortie.

- 0 : format B avec sortie W seulement (mono)



$aout = aW$

- 1 : format B avec sorties W et Y (stéréo)

$aleft = aW + 0.7071 * aY$

$aright = aW - 0.7071 * aY$

- 2 : format B avec sorties W, X et Y (2D). Peut être converti au format UHJ :

$aWre, aWim \quad \text{hilbert } aW$

$aXre, aXim \quad \text{hilbert } aX$

$aYre, aYim \quad \text{hilbert } aY$

$aWXr = 0.0928 * aXre + 0.4699 * aWre$

$aWXiYr = 0.2550 * aXim - 0.1710 * aWim + 0.3277 * aYre$

$aleft = aWXr + aWXiYr$

$aright = aWXr - aWXiYr$

- 3 : format B avec toutes les sorties (3D)
- 4 : Simule une paire de microphones (sortie stéréo)

$aW \quad \text{butterlp } aW, \text{ ifreq} \quad ; \text{ les valeurs recommandées pour ifreq}$

$aY \quad \text{butterlp } aY, \text{ ifreq} \quad ; \text{ se situent autour de 1000 Hz}$

$aleft = aW + aX$

$aright = aY + aZ$

Le mode 0 est le moins couteux en capacité de calcul, tandis que le mode 4 est le plus gourmand.

Dans le mode 4, les filtres passe-bas facultatifs peuvent changer la réponse en fréquence en fonction de la direction. Par exemple, si la source sonore se situe à gauche de l'auditeur, les fréquences élevées sont atténuées dans le canal droit et légèrement augmentées dans le canal gauche. Si l'on utilise pas de filtre, cet effet n'a pas lieu. On peut expérimenter avec d'autres filtres (*tone*, etc.) pour un meilleur effet.

Noter que le mode 4 est plutôt destiné à une écoute au casque et qu'il est aussi plus coûteux en calcul que les modes du format B (0 à 3). Dans ce cas, le paramètre *idist* fixe la distance entre les microphones gauche et droit ; pour le casque, des valeurs comprises entre 0.2 et 0.25 sont recommandées, bien que l'on puisse utiliser des valeurs plus grandes, jusqu'à 0.4, pour des effets stéréo larges.

On peut trouver plus d'information sur le format B ici : [http://www.york.ac.uk/inst/mustech/3d\\_audio/ambis2.htm](http://www.york.ac.uk/inst/mustech/3d_audio/ambis2.htm)

*irlen* -- Longueur de la réponse impulsionnelle des échos (en secondes). Dépend des paramètres du filtre, les valeurs entre 0.005 et 0.01 conviennent à la plupart des usages (les valeurs supérieures donnent une sortie plus précise, mais plus lente à calculer).

*iftnocl* (facultatif, 0 par défaut) -- S'il vaut 1, la table de sortie n'est pas effacée (mélange avec les données existantes), s'il vaut 0, la table est effacée avant l'écriture.

## Exemples

Voici un exemple de l'opcode spat3dt. Il utilise le fichier *spat3dt.csd* [examples/spat3dt.csd].

**Exemple 932. Exemple de l'opcode spat3dt.**

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o spat3dt.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

garvb    init 0
gaW      init 0
gaX      init 0
gaY      init 0

itmp ftgen 1, 0, 64, -2, 2, 40, -1, -1, -1, 123,      \
      1, 13.000, 0.05, 0.85, 20000.0, 0.0, 0.50, 2,  \
      1, 2.000, 0.05, 0.85, 20000.0, 0.0, 0.25, 2,  \
      1, 16.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2,  \
      1, 9.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2,  \
      1, 12.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2,  \
      1, 8.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2

itmp ftgen 2, 0, 262144, -2, 0
      spat3dt 2, -0.2, 1, 0, 1, 1, 2, 0.005

itmp ftgen 3, 0, 262144, -52, 3, 2, 0, 4, 2, 1, 4, 2, 2, 4

instr 1

a1 vco2 1, 440, 10
kfrq port 100, 0.008, 20000
a1 butterlp a1, kfrq
a2 linseg 0, 0.003, 1, 0.01, 0.7, 0.005, 0, 1, 0
a1 = a1 * a2 * 2
    denorm a1
    vincr garvb, a1
aw, ax, ay, az spat3di a1, p4, p5, p6, 1, 1, 2
    vincr gaW, aw
    vincr gaX, ax
    vincr gaY, ay

endin

instr 2

denorm garvb
; skip as many samples as possible without truncating the IR
arW, arX, arY ftconv garvb, 3, 2048, 2048, (65536 - 2048)
aW = gaW + arW
aX = gaX + arX
aY = gaY + arY
garvb = 0
gaW = 0
gaX = 0
gaY = 0

```

```

aWre, aWim hilbert aW
aXre, aXim hilbert aX
aYre, aYim hilbert aY
aWXr    = 0.0928*aXre + 0.4699*aWre
aWXiYr  = 0.2550*aXim - 0.1710*aWim + 0.3277*aYre
aL       = aWXr + aWXiYr
aR       = aWXr - aWXiYr
          outs aL, aR

endin

</CsInstruments>
<CsScore>

i 1 0 0.5 0.0 2.0 -0.8
i 1 1 0.5 1.4 1.4 -0.6
i 1 2 0.5 2.0 0.0 -0.4
i 1 3 0.5 1.4 -1.4 -0.2
i 1 4 0.5 0.0 -2.0 0.0
i 1 5 0.5 -1.4 -1.4 0.2
i 1 6 0.5 -2.0 0.0 0.4
i 1 7 0.5 -1.4 1.4 0.6
i 1 8 0.5 0.0 2.0 0.8
i 2 0 10
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*spat3d, spat3di*

## Crédits

Auteur : Istvan Varga  
2001

Nouveau dans la version 4.12

Mis à jour en avril 2002 par Istvan Varga

# spdist

spdist — Calcule les valeurs de distance à partir des coordonnées xy.

## Description

*spdist* utilise les mêmes données xy que *space*, depuis un fichier texte lu par *Gen28* ou depuis les arguments x et y donnés directement dans l'unité. Cet opcode a pour but de mettre à disposition les valeurs de distance calculées à partir des coordonnées xy.

Dans le cas de *space*, les valeurs xy sont utilisées pour déterminer une distance qui sert à atténuer le signal et à le préparer pour son utilisation dans *spsend*. Mais il est également utile d'avoir ces valeurs de distance pour pondérer la fréquence du signal avant de l'envoyer à l'unité *space*.

## Syntaxe

```
k1 spdist ifn, ktime, kx, ky
```

## Initialisation

*ifn* -- numéro de la fonction créée au moyen de *Gen28*. Ce générateur de fonction lit un fichier texte qui contient des groupes de trois valeurs représentant les coordonnées xy et un paramètre de temps indiquant quand le signal doit être placé à cette position. Le fichier ressemblera à ceci :

```
0   -1   1
1    1   1
2    4   4
2.1 -4  -4
3   10 -10
5  -40   0
```

Avec un fichier nommé « move » l'appel à *Gen28* dans la partition s'écrira :

```
f1 0 0 28 "move"
```

*Gen28* prend pour taille 0 et alloue la mémoire automatiquement. Il crée ses valeurs avec une résolution de 10 ms. Ainsi dans ce cas, il y aura 500 valeurs créées en interpolant de X1 à X2 à X3 et ainsi de suite, et de Y1 à Y2 à Y3 et ainsi de suite, sur le nombre approprié de valeurs stockées dans la table de fonction. Dans l'exemple ci-dessus, le son démarre à l'avant-gauche, après une seconde il atteint l'avant-droite, après une autre seconde il est plus éloigné mais toujours à l'avant-droite, ensuite il bouge à l'arrière-gauche en moins d'un dixième de seconde, un peu éloigné. Enfin, durant les neuf dixièmes de seconde suivants le son bouge à l'arrière-droite, modérément distant, puis il se fixe entre les deux haut-parleurs de gauche (plein ouest !), assez éloigné. Comme les valeurs dans la table sont lues par l'unité *space* au moyen d'un pointeur temporel, le temps courant peut être réglé pour suivre exactement celui du fichier, ou bien il peut être réglé pour aller plus vite ou plus lentement le long de la même trajectoire. Si l'on a accès à l'interface graphique permettant de dessiner et d'éditer les fichiers, il n'est pas nécessaire de créer les fichiers texte manuellement. Mais dès lors que le fichier est en ASCII dans le format ci-dessus, peu importe comment il a été créé !

IMPORTANT: Si *ifn* vaut 0, *spdist* obtient les valeurs des coordonnées xy depuis *kx* et *ky*.

## Exécution

La configuration des coordonnées xy dans l'espace place le signal de la manière suivante :

- a1 est en (-1, 1)
- a2 est en (1, 1)
- a3 est en (-1, -1)
- a4 est en (1, -1)

Ceci suppose une disposition des haut-parleurs où a1 est à l'avant-gauche, a2 à l'avant-droit, a3 à l'arrière-gauche et a4 à l'arrière-droite. Les valeurs supérieures à 1 donnent un son atténué, comme s'il était éloigné. *space* considère que les haut-parleurs sont à une distance de 1 ; on peut utiliser des valeurs de xy inférieures, mais *space* n'amplifiera pas le signal dans ce cas. Il équilibrera le signal cependant de manière à ce qu'il soit entendu comme s'il se trouvait à l'intérieur de l'espace des quatre haut-parleurs.  $x=0, y=1$ , place le signal entre les canaux avant gauche et droite,  $x=y=0$  place le signal également entre les quatre canaux, et ainsi de suite. Bien que *space* fournisse quatre signaux en sortie, on peut l'utiliser dans un orchestre à deux canaux. Si les xy sont tels que y reste  $\geq 1$ , il fonctionnera correctement pour faire des panoramiques et des localisations fixes dans un champ stéréophonique.

*ktime* -- indice dans la table contenant les coordonnées xy. S'il est utilisé comme ceci :

```
ktime      line 0, 5, 5
a1, a2, a3, a4 space asig, 1, ktime, ...
```

avec le fichier « move » décrit ci-dessus, la vitesse du mouvement du signal sera exactement celle qui est décrite dans ce fichier. Cependant avec :

```
ktime      line 0, 10, 5
```

le signal se déplacera deux fois moins vite qu'à la vitesse spécifiée. Ou dans le cas de :

```
ktime      line 5, 15, 0
```

le signal se déplacera dans la direction inverse et trois fois moins vite ! Enfin avec :

```
ktime      line 2, 10, 3
```

le signal ne se déplacera que depuis l'endroit spécifié à la ligne 3 du fichier texte jusqu'à l'endroit spécifié à la ligne 5 du fichier texte, et il lui faudra 10 secondes pour le faire.

*kx, ky* -- lorsque *ifn* vaut 0, *space* et *spdist* utilisent ces valeurs comme coordonnées xy pour positionner le signal.

## Exemples

Voici un exemple de l'opcode *spdist*. Il utilise le fichier *spdist.csd* [examples/spdist.csd].

**Exemple 933. Exemple de l'opcode spdist.**

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o spdist.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 4

ga1 init 0
ga2 init 0
ga3 init 0
ga4 init 0

instr 1 ;uses GEN28 file "move2", as found in /manual/examples

ifreq = 1
kx    init 0
ky    init 0
ktime line 0, 5.6, 5.6    ;same time as in table 1 ("move2")
kdist spdist 1, ktime, kx, ky
kfreq = (ifreq*340) / (340 + kdist) ;calculate doppler shift
printk2 kdist    ;print distance values
asig disk2 "flute.aiff", kfreq, 0, 1 ;sound source is looped
a1, a2, a3, a4 space asig, 1, ktime, .1, kx, ky ;use table 1 = GEN28
ar1, ar2, ar3, ar4 spsend    ;send to reverb

ga1 = ga1+ar1
ga2 = ga2+ar2
ga3 = ga3+ar3
ga4 = ga4+ar4
outq a1, a2, a3, a4

endin

instr 99 ; reverb instrument

a1 reverb2 ga1, 2.5, .5
a2 reverb2 ga2, 2.5, .5
a3 reverb2 ga3, 2.5, .5
a4 reverb2 ga4, 2.5, .5
outq a1, a2, a3, a4

ga1=0
ga2=0
ga3=0
ga4=0

endin
</CsInstruments>
<CsScore>
f1 0 0 28 "move2" ;from left front and left rear to the middle in front

i 1 0 5.6 ;same time as ktime
i 99 0 10 ;keep reverb active

```

```
e  
</CsScore>  
</CsoundSynthesizer>
```

Les mêmes fonction et valeurs temporelles sont utilisées pour *spdist* et pour *space*. Grâce à cela les valeurs de distance utilisées en interne par l'unité *space* seront les mêmes que celles retournées par *spdist* pour donner l'impression de l'effet Doppler !

## Voir aussi

*space*, *spsend*

## Crédits

Auteur : Richard Karpen  
Seattle, WA USA  
1998

Nouveau dans la version 3.48 de Csound.

# specaddm

specaddm — Exécute une somme pondérée de deux spectres.

## Description

Exécute une somme pondérée de deux spectres.

## Syntaxe

```
wsig specaddm wsig1, wsig2 [, imul2]
```

## Initialisation

*imul2* (facultatif, 0 par défaut) -- s'il est différent de zéro, pondère les magnitudes de *wsig2* avant la somme. La valeur par défaut est 0.

## Exécution

*wsig1* -- le premier spectre en entrée.

*wsig2* -- le second spectre en entrée.

Exécute une somme pondérée de deux spectres. Pour chaque canal des deux spectres en entrée, les deux magnitudes sont combinées et écrites en sortie selon la formule :

$$\text{magout} = \text{mag1in} + \text{mag2in} * \text{imul2}$$

L'opération est exécutée chaque fois qu'un nouveau *wsig1* est détecté en entrée. Cette unité vérifie à l'initialisation la consistance des deux spectre (même taille, même période, même type de magnitude).

## Voir aussi

*specdiff*, *specfilt*, *spechist*, *specscal*



# specdiff

specdiff — Trouve les valeurs de différence positive entre trames spectrales consécutives.

## Description

Trouve les valeurs de différence positive entre trames spectrales consécutives.

## Syntaxe

```
wsig specdiff wsignin
```

## Exécution

*wsig* -- le spectre de sortie.

*wsignin* -- le spectre d'entrée.

Trouve les valeurs de différence positive entre trames spectrales consécutives. A chaque nouvelle trame de *wsignin*, chaque valeur de magnitude est comparée avec celle qui la précède et les changements positifs sont écrits dans le spectre de sortie. Cette unité est utile comme détecteur d'attaque d'énergie.

## Exemples

```
wsig2    specdiff    wsig1      ; sense onsets
wsig3    specfilt    wsig2, 2    ; absorb slowly
          specdisp    wsig2, 0.1  ; & display both spectra
          specdisp    wsig3, 0.1
```

## Voir aussi

*specaddm*, *specfilt*, *spechist*, *specscal*

# specdisp

specdisp — Affiche les valeurs de magnitude du spectre.

## Description

Affiche les valeurs de magnitude du spectre.

## Syntaxe

```
specdisp wsig, iprd [, iwtflg]
```

## Initialisation

*iprd* -- la période, en secondes, de chaque nouvel affichage.

*iwtflg* (facultatif, 0 par défaut) -- indicateur de pause. S'il est différent de zéro, chaque affichage est maintenu jusqu'à ce que l'utilisateur le libère. La valeur par défaut est 0 (pas de pause).

## Exécution

*wsig* -- le spectre d'entrée.

Affiche les valeurs de magnitude du spectre *wsig* toutes les *iprd* secondes (arrondi à un multiple entier du *iprd* original de *wsig*).

## Exemples

```
ksum      specsum  wsig, 1          ; sum the spec bins, and ksmooth
          if ksum < 2000 kgoto zero  ; if sufficient amplitude
kocf      specptrk wsig              ; pitch-track the signal
          kgoto     contin
zero:
  kocf    =          0              ; else output zero
contin:
```

## Voir aussi

*specsum*

# specfilt

specfilt — Filtre chaque canal d'un spectre en entrée.

## Description

Filtre chaque canal d'un spectre en entrée.

## Syntaxe

```
wsig specfilt wsigin, ifhtim
```

## Initialisation

*ifhtim* -- demi-constante de temps.

## Exécution

*wsigin* -- le spectre d'entrée.

Filtre chaque canal d'un spectre en entrée. A chaque nouvelle trame de *wsigin*, chaque valeur de magnitude est injectée dans un filtre passe-bas récursif du premier ordre, dont la demi-constante de temps a été initialement fixée en échantillonnant la ftable *ifhtim* par l'espace (logarithmique) de fréquence du spectre d'entrée. Cette unité applique effectivement un facteur de *persistence* aux données apparaissant dans chaque canal spectral, et il est utile pour simuler *l'intégration d'énergie* qui survient durant la perception auditive. On peut aussi l'utiliser comme *histogramme* courant atténué dans le temps de la distribution spectrale.

## Exemples

```
wsig2    specdiff    wsig1      ; sense onsets
wsig3    specfilt    wsig2, 2    ; absorb slowly
          specdisp    wsig2, 0.1  ; & display both spectra
          specdisp    wsig3, 0.1
```

## Voir aussi

*specaddm*, *specdiff*, *spechist*, *specscal*

# spechist

spechist — Accumule les valeurs de trames spectrales successives.

## Description

Accumule les valeurs de trames spectrales successives.

## Syntaxe

```
wsig spechist wsignin
```

## Exécution

*wsignin* -- le spectre d'entrée

Accumule les valeurs de trames spectrales successives. A chaque nouvelle trame de *wsignin*, les accumulations récentes dans chaque piste de magnitude sont écrits dans le spectre de sortie. Cette unité fournit ainsi un *histogram* courant de la distribution spectrale.

## Exemples

```
wsig2    specdiff    wsig1        ; sense onsets
wsig3    specfilt    wsig2, 2      ; absorb slowly
          specdisp    wsig2, 0.1    ; & display both spectra
          specdisp    wsig3, 0.1
```

## Voir aussi

*specaddm, specdiff, specfilt, specscal*

# specptrk

specptrk — Estime la hauteur du ton complexe le plus proéminent dans le spectre.

## Description

Estime la hauteur du ton complexe le plus proéminent dans le spectre.

## Syntaxe

```
koct, kamp specptrk wsig, kvar, ilo, ihi, istr, idbthresh, inptls, \  
    irolloff [, iodd] [, iconfs] [, interp] [, ifprd] [, iwtflg]
```

## Initialisation

*ilo, ihi, istr* -- conditionneurs de l'intervalle des hauteurs (grave, aigu et valeur de départ) exprimés sous le forme octave point partie décimale.

*idbthresh* -- seuil d'énergie (en décibels) pour que le suivi de hauteur ait lieu. Une fois commencé, le suivi continue jusqu'à ce que l'énergie soit inférieure à la moitié du seuil (6 dB plus bas), et ensuite les sorties *koct* et *kamp* restent à zéro jusqu'à ce que le seuil soit à nouveau surpassé. *idbthresh* est une valeur repère. A l'initialisation il est d'abord converti au mode *idbout* du spectre source (et le point 6 dB plus bas devient 0.5, 0.25, ou 1/(racine carrée de 2) pour les modes 0, 2 et 3). Les valeurs sont ensuite pondérées pour permettre la somme pondérée des partiels utilisée durant la corrélation. Le seuillage effectif est réalisant en utilisant la valeur interne pondérée et sommée *kamp* qui apparaît comme second paramètre de sortie.

*inptls, irolloff* -- nombre d'harmoniques utilisés comme modèle pour la détection de hauteur basée sur le spectre, et pente d'amplitude pour l'ensemble exprimée sous la forme d'une fraction par octave (linéaire, donc ne pas descendre dans les valeurs négatives). Comme les harmoniques et la pente peuvent affecter le suivi de hauteur, il est utile d'expérimenter : essayer 4 ou 5 harmoniques avec une pente de 0.6 comme réglages initiaux ; monter jusqu'à 10 ou 12 harmoniques avec une pente de 0.75 pour des timbres complexes comme celui du basson (faible fondamental). Le temps de calcul dépend du nombre d'harmoniques cherchés. Le nombre maximum est 16.

*iodd* (facultatif) -- s'il est différent de zéro, seuls les harmoniques impairs sont employés dans l'ensemble ci-dessus (par exemple si *inptls* vaut 4, les harmoniques 1, 3, 5, 7 seront employés). Ceci améliore le suivi de certains instruments comme la clarinette. La valeur par défaut est 0 (employer tous les harmoniques).

*iconfs* (facultatif) -- nombre de confirmations requises pour que le suiveur de hauteur saute d'une octave, au prorata de fractions d'une octave (ainsi la valeur 12 implique qu'un changement d'un demi-ton nécessite 1 confirmation (deux succès) au *spectre* générant *iprd*). Ce paramètre limite les analyses de hauteur faussées tel que les erreurs d'octave. Une valeur de 0 signifie qu'aucune confirmation n'est requise ; la valeur par défaut est 10.

*interp* (facultatif) -- s'il est différent de zéro, chaque signal de sortie (*koct*, *kamp*) est interpolé entre les trames entrantes de *wsig*. La valeur par défaut est 0 (répéter les valeurs du signal entre les trames).

*ifprd* (facultatif) -- s'il est différent de zéro, le spectre calculé en interne des fondamentaux candidats est affiché. La valeur par défaut est 0 (pas d'affichage).

*iwtflg* (facultatif) -- indicateur de pause. S'il est différent de zéro, chaque affichage est maintenu jusqu'à ce que l'utilisateur le libère. La valeur par défaut est 0 (pas de pause).

## Exécution

A l'initialisation de la note, cette unité crée un modèle de *inptls* harmoniques (harmoniques impairs si *iodd* est différent de zéro) avec une pente d'amplitude valant *iroloff* par octave. A chaque nouvelle trame de *wsig*, le spectre est mis en corrélation croisée avec ce modèle pour fournir un spectre en interne de fondamentaux candidats (facultativement affichés). Une paire hauteur/amplitude probable (*koct*, *kamp*, en octave point partie décimale et en *idbout* additionnés) est ensuite estimée. *koct* ne varie pas du *koct* précédent de plus ou moins *kvar* unités décimales d'octave. Il est aussi contraint à rester dans l'intervalle *ilo* -- *ihi* (hauteur grave et hauteur aigue en octave point valeur décimale ). *kvar* peut être dynamique, par exemple les attaques dépendant des amplitudes. La résolution de hauteur utilise *ifrq*s bins/octave du *spectre* d'origine, avec interpolation parabolique entre bins adjacents. Les réglages suivants, racine carrée de la magnitude, *ifrq*s = 24, *iq* = 15 devraient capturer toutes les inflexions intéressantes. Entre les trames, la sortie est soit répétée soit interpolée au taux-k. (Voir *spectrum*.)

## Exemples

```

a1, a2  ins                                ; read a stereo clarinet input
krms    rms                                ; find a monaural rms value
kvar    = 0.6 + krms/8000                  ; & use to gate the pitch var
wsig    spectrum a1, 0.01, 7, 24, 15, 0, 3 ; get a 7-oct spectrum, 24 bins
specdisp wsig, 0.2                         ; display this and now estimate
koct, ka spectrk wsig, kvar, 7.0, 10, 9, 20, 4, 0.7, 1, 5, 1, 0.2 ; the pch and amp
aosc    oscil ka * ka * 10, cpsoct(koct), 2 ; & generate \ new tone with t
koct    = (koct < 7.0 ? 7.0 : koct)         ; replace non pitch with low C
display koct - 7.0, 0.25, 20                ; & display the pitch track
display ka, 0.25, 20                        ; plus the summed root mag
outs    a1, aosc                           ; output 1 original and 1 new

```

# specscal

specscal — Pondère un bloc spectral en entrée avec des enveloppes spectrales.

## Description

Pondère un bloc spectral en entrée avec des enveloppes spectrales.

## Syntaxe

```
wsig specscal wsigin, ifscale, ifthresh
```

## Initialisation

*ifscale* -- table de la fonction de pondération contenant les valeurs par lesquelles une valeur de magnitude est pondérée.

*ifthresh* -- table de la fonction de seuil. Si *ifthresh* est différent de zéro, chaque magnitude est réduite par sa valeur de table correspondante (on ne descend pas en-dessous de zéro).

## Exécution

*wsig* -- le spectre de sortie

*wsigin* -- le spectre d'entrée

Pondère un bloc spectral en entrée avec des enveloppes spectrales. Les tables de fonction *ifthresh* et *ifscale* sont initialement échantillonnées dans l'espace de fréquence (logarithmique) du spectre d'entrée ; puis, chaque fois qu'un nouveau spectre d'entrée est détecté, les valeurs échantillonnées sont utilisées pour pondérer chacun de ses canaux de magnitude de la manière suivante : si *ifthresh* est différent de zéro, chaque magnitude est réduite par sa valeur de table correspondante (on ne descend pas en-dessous de zéro) ; ensuite chaque magnitude est repondérée par la valeur de *ifscale* correspondante, et le spectre résultant est écrit dans *wsig*.

## Exemples

```
wsig2    specdiff    wsig1        ; sense onsets
wsig3    specfilt    wsig2, 2      ; absorb slowly
          specdisp    wsig2, 0.1    ; & display both spectra
          specdisp    wsig3, 0.1
```

## Voir aussi

*specaddm*, *specdiff*, *specfilt*, *spechist*

# specsum

specsum — Additionne les magnitudes sur tous les canaux du spectre.

## Description

Additionne les magnitudes sur tous les canaux du spectre.

## Syntaxe

```
ksum specsum wsig [, interp]
```

## Initialisation

*interp* (facultatif, 0 par défaut) -- s'il est différent de zéro, le signal de sortie (*koct* ou *ksum*) est interpolé. La valeur par défaut est 0 (répéter la valeur du signal entre les changements).

## Exécution

*ksum* -- le signal de sortie.

*wsig* -- le spectre d'entrée.

Additionne les magnitudes sur tous les canaux du spectre. A chaque nouvelle trame de *wsig*, les magnitudes sont additionnées et restituées sous la forme du signal scalaire *ksum*. Entre les trames, la sortie est soit répétée soit interpolée au taux-k. Cette unité produit un signal de taux-k, somme des magnitudes présentes dans les données spectrales, et donne ainsi la mesure courante de sa puissance globale instantanée.

## Exemples

```
ksum    specsum    wsig, 1                ; sum the spec bins, and ksmooth
        if ksum < 2000 kgoto zero          ; if sufficient amplitude
koct    specptrk    wsig                    ; pitch-track the signal
        kgoto      contin
zero:
koct    =           0                      ; else output zero
contin:
```

## Voir aussi

*specdisp*



# spectrum

**spectrum** — Génère une TFD à Q constant et espacement exponentiel.

## Description

Génère une TFD à Q constant et espacement exponentiel sur toutes les octaves d'un signal de contrôle ou audio en entrée, multiplié et sous-échantillonné.

## Syntaxe

```
wsig spectrum xsig, iprd, iocts, ifrqa [, iq] [, ihann] [, idbout] \  
      [, idsprd] [, idsinrs]
```

## Initialisation

*ihann* (facultatif) -- applique une fenêtre de Hamming ou de Hanning à l'entrée. La valeur par défaut est 0 (fenêtre de Hamming).

*idbout* (facultatif) -- conversion codée de la TFD en sortie :

- 0 = magnitude
- 1 = dB
- 2 = magnitude au carré
- 3 = racine carrée de la magnitude

La valeur par défaut est 0 (magnitude).

*idsprd* (facultatif) -- s'il est différent de zéro, le tampon composite de sous-échantillonnage est affiché toutes les *idsprd* secondes. La valeur par défaut est 0 (pas d'affichage).

*idsines* (facultatif) -- s'il est différent de zéro, les sinusoides passées dans une fenêtre de Hamming ou de Hanning et utilisées dans le filtrage par TFD sont affichées. La valeur par défaut est 0 (pas d'affichage).

## Exécution

Cette unité passe d'abord le signal *asig* ou *ksig* à travers *iocts* décimations par octave et sous-échantillonnages successifs et garde un tampon de valeurs sous-échantillonnées dans chaque octave (facultativement affiché comme tampon composite toutes les *idsprd* secondes). Puis, toutes les *iprd* secondes, les échantillons préservés sont passés dans un banc de filtres (*ifrqs* filtres parallèles par octave, espacés exponentiellement avec un rapport Q de la fréquence sur la largeur de bande égal à *iq*), et les magnitudes de sortie (*idbout*) sont éventuellement converties pour produire un spectre à bande limitée pouvant être lu par d'autres unités.

Les étapes de ce processus utilisent intensivement les moyens de calcul et le temps de calcul est proportionnel à *iocts*, *ifrqs* et *iq*, et inversement proportionnel à *iprd*. Les réglages suivants, *ifrqs* = 12, *iq* = 10, *idbout* = 3 et *iprd* = 0.02, conviendront généralement, mais on recommande l'expérimentation. Actuellement *ifrqs* a au maximum 120 divisions par octave. Pour une entrée audio, les bins de fréquence sont réglés pour coïncider avec le la 440.

Cette unité produit un bloc de données spectrales auto-définies *wsig*, dont les caractéristiques utilisées (*iprd*, *iocts*, *ifrq*, *idbout*) sont transmises via le bloc de données lui-même à tous les *wsigs* dérivés. Il peut y avoir n'importe quel nombre d'unités *spectrum* dans un instrument ou dans un orchestre, mais tous les noms de *wsig* doivent être uniques.

## Exemples

```
asig in                               ; get external audio
wsig spectrum asig, 0.02, 6, 12, 33, 0, 1, 1 ; downsample in 6 octs & calc a 72 pt dft
                                           ; (Q 33, dB out) every 20 msec
```

# splitrig

splitrig — Divise un signal déclencheur.

## Description

*splitrig* divise un signal déclencheur (c-à-d une suite temporelle d'impulsions au taux de contrôle) en plusieurs canaux suivant une structure conçue par l'utilisateur.

## Syntaxe

```
splitrig ktrig, kndx, imaxtics, ifn, kout1 [,kout2,...,koutN]
```

## Initialisation

*imaxtics* -- nombre de tics appartenant au motif le plus grand.

*ifn* -- numéro de la table contenant la structure des données par canal.

## Exécution

*asig* -- signal entrant

*ktrig* -- signal déclencheur

L'opcode *splitrig* divise un signal déclencheur en plusieurs canaux suivant un ou plusieurs motifs fournis par l'utilisateur. Normalement le signal déclencheur régulier généré par l'opcode *metro* est utilisé pour être transformé en motif rythmique pouvant déclencher plusieurs mélodies indépendantes ou plusieurs riffs de percussion. Mais on peut aussi partir de signaux de déclenchement non-isochrones. Ceci permet des variations de groove "interprétées" et moins "mécaniques". Les motifs sont en boucle et le cycle est répété chaque *nombre\_de\_tics\_du\_motif\_N*.

Le schéma des motifs est défini par l'utilisateur et stocké dans la table *ifn* dans le format suivant :

```
          gil  ftgen 1,0,1024, -2 \ ; la table est générée avec GEN02 dans ce cas
\
nombre_de_tics_du_motif_1, \ ;motif 1
    tic1_out1, tic1_out2, ... , tic1_outN,\
    tic2_out1, tic2_out2, ... , tic2_outN,\
    tic3_out1, tic3_out2, ... , tic3_outN,\
    .....
    ticN_out1, ticN_out2, ... , ticN_outN,\
\
nombre_de_tics_du_motif_2, \ ;motif 2
    tic1_out1, tic1_out2, ... , tic1_outN,\
    tic2_out1, tic2_out2, ... , tic2_outN,\
    tic3_out1, tic3_out2, ... , tic3_outN,\
    .....
    ticN_out1, ticN_out2, ... , ticN_outN,\
    .....
\
nombre_de_tics_du_motif_N, \ ;motif N
    tic1_out1, tic1_out2, ... , tic1_outN,\
    tic2_out1, tic2_out2, ... , tic2_outN,\
    tic3_out1, tic3_out2, ... , tic3_outN,\
```

```
.....  
ticN_out1, ticN_out2, ... , ticN_outN,\
```

Ce schéma peut contenir plus d'un motif, chacun avec un nombre différent de lignes. Chaque motif est précédé par une ligne spéciale contenant un seul champ *nombre\_de\_tics\_du\_motif\_N* ; ce champ donne le nombre de tics constituant le motif correspondant. Chaque ligne du motif compose un tic. Chaque colonne du motif correspond à un canal, et chaque champ d'une ligne est un nombre qui constitue la valeur sortie par le canal correspondant *koutXX* (si ce nombre est zéro, le canal de sortie correspondant ne déclenchera rien dans cet argument particulier). Evidemment, chaque ligne doit contenir le même nombre de champs qui doit égaler le nombre de canaux *koutXX*. Tous les motifs doivent contenir le même nombre de lignes ; ce nombre doit être égal au plus grand des motifs et il est défini par la variable *imaxtics*. Même si un motif compte moins de tics que le motif le plus grand, il doit contenir le même nombre de lignes. Dans ce cas, certaines de ces lignes, à la fin du motif, ne seront pas utilisées (et peuvent ainsi prendre n'importe quelle valeur, car elle est sans importance).

La variable *kndx* donne le numéro du motif à jouer, zéro indiquant le premier motif. Chaque fois que la partie entière de *kndx* change, le compteur de tic est remis à zéro.

Les motifs sont en boucle et le cycle est répété chaque *nombre\_de\_tics\_du\_motif\_N*.

exemples 4 - calcule la valeur moyenne de *asig* dans l'intervalle de temps.

Cet opcode peut être utile dans certaines situations, par exemple pour implémenter un vu-mètre.

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (n'était disponible auparavant que dans CsoundAV).

# sprintf

`sprintf` — Sortie formatée à la `printf` dans une variable chaîne de caractères.

## Description

*sprintf* écrit une sortie formatée à la `printf` dans une variable chaîne de caractères, comme le fait la fonction C `sprintf()`. *sprintf* ne s'exécute que pendant l'initialisation.

## Syntaxe

```
Sdst sprintf Sfmt, xarg1[, xarg2[, ... ]]
```

## Initialisation

*Sfmt* -- chaîne de formatage comme dans `printf()` et d'autres fonctions C similaires, sauf que les modificateurs de longueur (l, ll, h, etc.) ne sont pas supportés. Les spécificateurs de conversion suivants sont permis :

- d, i, o, u, x, X, e, E, f, F, g, G, c, s

*xarg1, xarg2, ...* -- arguments d'entrée (max. 30) à formater, doivent être de `taux-i` pour tous les spécificateurs de conversion sauf pour `%s`, qui nécessite un argument chaîne de caractères. Les formats d'entiers comme `%d` arrondissent les valeurs d'entrée à l'entier le plus proche.

## Exécution

*Sdst* -- variable chaîne de caractères en sortie

## Exemples

Voici un exemple de l'opcode `sprintf`. Il utilise le fichier *sprintf.csd* [examples/sprintf.csd].

### Exemple 934. Exemple de l'opcode `sprintf`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;;realtime audio out
;-iadc     ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sprintf.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

;the file "impuls20.aiff" can be found in /manual/examples
```

```
instr 1

ifn = 20
Sname sprintf "impuls%02d.aiff", ifn
Smsg sprintf "The file name is: '%s'", Sname
puts Smsg, 1
asig soundin Sname
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 1
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
The file name is: 'impuls20.aiff'
soundin: opened 'impuls20.aiff'
```

## Voir aussi

*sprintfk*

## Crédits

Auteur : Istvan Varga  
2005

# sprintfk

sprintfk — Sortie formatée à la printf dans une variable chaîne de caractères au taux-k.

## Description

*sprintfk* écrit une sortie formatée à la printf dans une variable chaîne de caractères, comme le fait la fonction C *sprintf()*. *sprintfk* s'exécute à la fois pendant l'initialisation et pendant l'exécution.

## Syntaxe

```
Sdst sprintfk Sfmt, xarg1[, xarg2[, ... ]]
```

## Initialisation

*Sfmt* -- chaîne de formatage comme dans *printf()* et d'autres fonctions C similaires, sauf que les modificateurs de longueur (l, ll, h, etc.) ne sont pas supportés. Les spécificateurs de conversion suivants sont permis :

- d, i, o, u, x, X, e, E, f, F, g, G, c, s

*xarg1*, *xarg2*, ... -- arguments d'entrée (max. 30) à formater, doivent être de taux-i pour tous les spécificateurs de conversion sauf pour %s, qui nécessite un argument chaîne de caractères. *sprintfk* accepte aussi les arguments numériques de taux-k, mais ceux-ci doivent quand même être valides à l'initialisation (à moins que *sprintfk* ne soit évité avec un *igoto*). Les formats d'entiers comme %d arrondissent les valeurs d'entrée à l'entier le plus proche.

## Exécution

*Sdst* -- variable chaîne de caractères en sortie

## Exemples

Voici un exemple de l'opcode *sprintfk*. Il utilise le fichier *sprintfk.csd* [examples/sprintfk.csd].

### Exemple 935. Exemple de l'opcode *sprintfk*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o sprintfk.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 48000
ksmps   = 16
nchnls  = 2
0dbfs   = 1
```

```

; Example by Jonathan Murphy 2007

instr 1

S1      =  "1"
S2      =  " + 1"
ktrig    init      0
kval     init      2
if (ktrig == 1) then
  S1      strcatk   S1, S2
  kval    =  kval + 1
endif
String    sprintfk  "%s = %d", S1, kval
puts      String, kval
ktrig     metro     1

endin

</CsInstruments>
<CsScore>
i1 0 10
e
</CsScore>
</CsoundSynthesizer>

```

La sortie contiendra des lignes comme celles-ci :

```

1 + 1 = 2
1 + 1 + 1 = 3
1 + 1 + 1 + 1 = 4
1 + 1 + 1 + 1 + 1 = 5
1 + 1 + 1 + 1 + 1 + 1 = 6
1 + 1 + 1 + 1 + 1 + 1 + 1 = 7
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 8
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 9
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 10
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 11
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 12

```

## Voir aussi

*sprintf, puts, strcat*

## Crédits

Auteur : Istvan Varga

2005

Exemple par Jonathan Murphy



# spsend

spsend — Génère des signaux de sortie basés sur un opcode *space* défini auparavant.

## Description

*spsend* dépend de l'existence d'un *space* défini antérieurement. Les signaux de sortie de *spsend* sont dérivés des valeurs de *xy* et de réverbération données dans le *space* et sont prêts à être envoyés à des unités de réverbération locales ou globales (voir l'exemple ci-dessous).

## Syntaxe

*a1*, *a2*, *a3*, *a4* **spsend**

## Exécution

La configuration des coordonnées *xy* dans l'espace place le signal de la manière suivante :

- *a1* est en (-1, 1)
- *a2* est en (1, 1)
- *a3* est en (-1, -1)
- *a4* est en (1, -1)

Ceci suppose une disposition des haut-parleurs où *a1* est à l'avant-gauche, *a2* à l'avant-droit, *a3* à l'arrière-gauche et *a4* à l'arrière-droite. Les valeurs supérieures à 1 donnent un son atténué, comme s'il était éloigné. *space* considère que les haut-parleurs sont à une distance de 1 ; on peut utiliser des valeurs de *xy* inférieures, mais *space* n'amplifiera pas le signal dans ce cas. Il équilibrera le signal cependant de manière à ce qu'il soit entendu comme s'il se trouvait à l'intérieur de l'espace des quatre haut-parleurs. *x*=0, *y*=1, place le signal entre les canaux avant gauche et droite, *x*=*y*=0 place le signal également entre les quatre canaux, et ainsi de suite. Bien que *space* fournisse quatre signaux en sortie, on peut l'utiliser dans un orchestre à deux canaux. Si les *xy* sont tels que *y* reste >= 1, il fonctionnera correctement pour faire des panoramiques et des localisations fixes dans un champ stéréophonique.

## Exemples

Voici un exemple stéréo de l'opcode *spsend*. Il utilise le fichier *spsend.csd* [examples/spsend.csd].

### Exemple 936. Exemple de l'opcode *spsend*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;;realtime audio out
;-iadc    ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o spsend.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2 ;stereo output

ga1 init 0
ga2 init 0

instr 1 ;sends different amounts to reverb

irev = p6
asig disk2 "fox.wav", 1
a1, a2, a3, a4 space asig, 0, 0, irev, p4, p5 ;take position values from p4, p5
ar1, ar2, ar3, ar4 spsend ;send to reverb

ga1 = ga1+ar1
ga2 = ga2+ar2
outs a1, a2

endin

instr 99 ; reverb instrument

a1 reverb2 ga1, 2.5, .5
a2 reverb2 ga2, 2.5, .5
outs a1, a2

ga1=0
ga2=0

endin

</CsInstruments>
<CsScore>
;WITH REVERB
;place the sound in the left speaker and near
i1 0 1 -1 1 .1
;place the sound in the right speaker and far
i1 1 1 45 45 .1
;place the sound equally between left and right and in the middle ground distance
i1 2 1 0 12 .1

;NO REVERB
;place the sound in the left speaker and near
i1 6 1 -1 1 0
;place the sound in the right speaker and far
i1 7 1 45 45 0
;place the sound equally between left and right and in the middle ground distance
i1 8 1 0 12 0

i 99 0 12 ;keep reverb active all the time
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*space, spdist*

## Crédits

Auteur : Richard Karpen  
Seattle, WA USA

1998

Nouveau dans la version 3.48 de Csound.

# sqrt

sqrt — Retourne une racine carrée.

## Description

Retourne la racine carrée de  $x$  ( $x$  non-négatif).

Les valeurs de l'argument sont restreintes pour *log*, *log10* et *sqrt*.

## Syntaxe

**sqrt**( $x$ ) (pas de restriction de taux)

**sqrt**( $k/i[]$ ) ( $k$ - ou  $i$ -tableau)

où l'argument entre parenthèses peut être une expression. Les convertisseurs de valeur effectuent une transformation arithmétique d'unités d'une sorte en unités d'une autre sorte. Le résultat peut devenir ensuite un terme dans une autre expression.

## Exemples

Voici un exemple de l'opcode sqrt. Il utilise le fichier *sqrt.csd* [examples/sqrt.csd].

### Exemple 937. Exemple de l'opcode sqrt.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sqrt.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1

asig    pluck 0.7, 55, 55, 0, 1
kpan    line 0,p3,1
kleft   = sqrt(1-kpan)
kright  = sqrt(kpan)
printks "square root of left channel = %f\\n", 1, kleft ;show coarse of square root values
outs    asig*kleft, asig*kright      ;where 0.707126 is between 2 speakers

endin
</CsInstruments>
<CsScore>
```

```
i 1 0 10  
e  
</CsScore>  
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
square root of left channel = 1.000000\r  
square root of left channel = 0.948688  
square root of left channel = 0.894437  
square root of left channel = 0.836676  
square root of left channel = 0.774620  
square root of left channel = 0.707139  
square root of left channel = 0.632499  
square root of left channel = 0.547781  
square root of left channel = 0.447295  
square root of left channel = 0.316242
```

## Voir aussi

*abs, exp, frac, int, log, log10, i*

# squinewave

squinewave — Un oscillateur d'onde carrée-pulsation-dent de scie-sinus à bande limitée et variation de forme avec synchro forcée.

## Description

Cet oscillateur génère une forme d'onde variable qui peut évoluer librement entre les formes classiques de sinus, carrée, pulsation et dent de scie. La forme est contrôlée par deux valeurs interactives : clip (rectitude) et skew (symétrie). Toutes les formes utilisent un nombre minimum d'échantillons par transition (par exemple, la fin abrupte d'une dent de scie ou d'une pulsation utilise un minimum de N échantillons), ce qui produit une sortie à bande limitée. Aux plus hautes fréquences, le taux de transition minimal prédomine, ce qui fait qu'au delà d'une certaine hauteur toutes les formes se "dégradent" en sinus. Le taux de transition minimal est configuré au temps-i. La synchronisation forcée (une transition très rapide à la phase nulle) est supportée, et un signal de synchro est produit une fois par cycle.

## Syntaxe

```
aout [, asyncout] squinewave acps, aClip, aSkew, asyncin [, iMinSweep] [, iphase]
aout [, asyncout] squinewave acps, aClip, aSkew [, ksyncin] [, iMinSweep] [, iphase]
```

## Initialisation

*iMinSweep* (facultatif) -- Supérieur ou égal à 4. Durée en échantillons des transitions les plus courtes des formes carrées/pulsation. Par défaut sr/3000 (pratiquement sans repliement).

*iphase* (facultatif, -1 par défaut) -- phase initiale, entre 0 et 2, proportionnelle aux segments de la forme d'onde (voir notes). Si une valeur négative est donnée, l'initialisation de la phase est ignorée.

## Exécution

*aout* -- sortie audio normalisée à +/-1.

*asyncout* -- (facultatif) -- signal de synchro : vaut 1 à la fin de chaque cycle, sinon 0.

*acps* -- fréquence. A partir de 0. Les fréquences négatives ne sont pas traitées.

*aClip* -- "rectitude" de la forme d'onde. Entre 0 et 1. 0 pour une sinus (ou une dent de scie), 1 pour une onde carrée (ou une pulsation).

*aSkew* -- symétrie de la forme d'onde. Entre -1 et +1. 0 pour une forme symétrique comme une sinus ou une onde carrée. +1 ou -1 pour une dent de scie ou une pulsations orientée à droite ou à gauche.

*asyncin/ksyncin* -- (facultatif, ignoré s'il n'est pas de taux-a) - lorsqu'il est  $\geq 1$ , la forme d'onde glisse rapidement vers la phase nulle. La durée de la transition varie entre 0 et  $1.5 * iMinSweep$  échantillons en fonction de la phase courante.

L'opcode *squinewave* est un oscillateur à forme d'onde variable générée en interne. La forme d'onde a deux parties :

1. pente de cosinus descendante suivie d'une partie plate à -1
2. pente de cosinus ascendante suivie d'une partie plate à +1

A la fin de (2) un signal de synchro est produit.

Clip (0-1)	contrôle la proportion entre la partie plate et la transition dans chaque segment.
Skew (-1 to +1)	contrôle la proportions des segments (1) et (2) dans le cycle de la forme d'onde.

Si skew < 0, la partie (1) est plus courte, if skew > 0, (1) est plus longue que (2)

Les formes d'onde classiques ont des valeurs simples :

- sinus : clip=0, skew=0
- dent de scie : clip=0, skew=+1 or -1 (orientation à gauche ou à droite)
- carrée : clip=1, skew=0
- pulsation: clip=1, skew=+1 or -1

Les valeurs fractionnaires produisent des formes d'onde intermédiaires.

*asynclin*

L'entrée de synchronisation forcée (asynclin >=1) fait glisser rapidement la forme d'onde vers sa fin en élévant la fréquence à  $2 * sr/iMinSweep$ . Les pulsations de synchro sont ainsi plus raides que la forme d'onde pulsation.

*iMinSweep*

la forme d'onde est à bande limitée grâce à l'utilisation d'un nombre minimal d'échantillons pour les transition de cosinus même si clip/skew ont des valeurs extrêmes. Ceci est contrôlé par *iMinSweep*. Comme *iMinSweep* est compté en échantillons, la forme d'onde produite dépend du taux d'échantillonnage, mais le spectre sera très proche d'un spectre indépendant de sr. La valeur par défaut sr/3000 est plutôt "douce", les valeurs étant de 14 échantillons à 44.1 kHz, 16 à 48 kHz et 32 à 96 kHz, etc. *iMinSweep* est compté en nombres entiers bien que ce ne soit pas strictement nécessaire.

Si plusieurs unités de *squinewave* sont jouées à l'unisson, il est recommandé d'utiliser différentes valeurs de *iMinSweep*. La valeur minimale de transition crée des zones plus calmes dans la série des harmoniques. En utilisant des valeurs minimales de transition légèrement différentes, le spectre est rempli plutôt que de souligner le contour spectral.

## Notes

*squinewave* est basé sur des cosinus plutôt que sur des sinus pour générer la forme d'onde. (Ceci simplifie la logique de contrôle). La différence est que  $\cos(0) = 1$ , alors que  $\sin(0) = 0$ . Cela signifie que la synchronisation forcée se produit quand la forme d'onde atteint une crête, juste avant d'entamer sa pente descendante. (Une synchronisation forcée sinusoïdale se produirait au passage à zéro).

*Conseil*

La durée des transitions de synchronisation forcée permet d'enchaîner la synchronisation de plusieurs unités de *squinewave* ce qui crée des impulsions à synchronisation forcée décalées.

*Stabilité des hauteurs*

Noter que *iMinSweep* limite la possibilité de *squinewave* d'atteindre une fréquence exacte. Quand skew ou clip sont actifs, et qu'une MF est appliquée, la forme d'onde de *squinewave* sera plus longue ou plus courte que la période exacte. Cependant, les différences se compensent, si bien qu'ave une MF symétrique, *squinewave* dévie en approchant une fréquence moyenne. Un signal sinusoïdal non mis en forme (clip=skew=0) correspond à la sortie de poscil sur 7 à 8 chiffres significatifs (également sous MF).

*Phase initiale*

Il est utile de fixer la phase initiale si squinewave est utilisé comme un LFO formé. La phase initiale est divisée en quatre segments avec des valeurs symboliques comprises entre 0 et 2, et elle démarrera ainsi à l'endroit attendu indépendamment des valeurs de skew/clip. 0-1 représente la première partie, 1-2 la seconde. Voici quelques valeurs de *iphase* intéressantes :

- 0 - début de la première pente descendante.
- 0.5 - fin de la pente descendante (début de la section plate "basse").
- 1 - point médian, fin de la première section plate, début de la seconde pente ascendante.
- 1.5 - fin de la pente ascendante (début de la section plate "haute").
- 0.25 et 1.25 sont les passages à zéro au milieu des sections de pente descendante/ascendante.
- 0.75 et 1.75 sont au milieu des sections plates basse/haute.

Si *iphase* < 0 (ignoré) à la première utilisation, la phase initiale est fixée à 1.25, c'est-à-dire au passage à zéro de la pente ascendante. La sortie ressemble alors à une onde sinusoïdale.

## Exemples

Voici un exemple de l'opcode squinewave. Il utilise le fichier *squinewave.csd* [exemples/squinewave.csd].

### Exemple 938. Exemple de l'opcode squinewave.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sqrt.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

#####
0dbfs = 1.1
nchnls = 2
ksmps = 100

; aSyncin, FMod for instr 2
gaFmod init 0
gasync init 0

; Modulator squinewave
instr 1
; freq start, end
acps line p4, p3, p5
; shape start, end
aclip line p6, p3, p7
askew line p8, p3, p9

; ar, async squinewave aFreq , aclip, askew [, asyncin, iMinSweep, iphase]
aout1, gasync squinewave cpsoct(acps), aclip, askew, 0, 17

outs1 aout1

gaFMod = aout1
endin
```



```

; squinewave using gaFMod and gasync input from i1
instr 2
; freq & shape start, end
acps line p4, p3, p5
aclip line p6, p3, p7
askew line p8, p3, p9

aFMinIndex line p10, p3, p11
asynccin = gasync * p12
afreq = cpsoct(acps + aFMinIndex * gaFMod)

; ar squinewave afreq, aclip, askew [, asynccin, iMinSweep, iphase]
aout2 squinewave afreq, aclip, askew, asynccin

outs2 aout2
endin

</CsInstruments>
<CsScore>

```

```

; First part instr 1 hardsyncs instr 2 (p12)

; p4=fund clip skew
i1 0 1. 6.11 6.06 0 1 -.1 +.1
i1 + 1. pp5 2.03 pp7 0 pp9 0
i1 + 1. pp5 7.11 pp7 .8 pp9 -.8
i1 + 2. pp5 8.11 pp7 .2 pp9 1
i1 + .5 pp5 6.05 pp7 .5 pp9 -.6
i1 + 2.5 pp5 6.05 pp7 1 pp9 1

; p4=fund clip skew p10=FM p12=sync
i2 0 .5 1.08 2.06 0 .3 -.5 +.5 0 0 1
i2 + .5 pp5 4.03 pp7 .5 pp9 -.6 pp11 . .
i2 + .5 pp5 5.11 pp7 1 pp9 .5 pp11 . .
i2 + .5 pp5 6.01 pp7 .8 pp9 -.5 pp11 . .
i2 + .5 pp5 2.11 pp7 .1 pp9 .3 pp11 . .
i2 + .5 pp5 3.11 pp7 .8 pp9 -.8 pp11 . .
i2 + 2. pp5 4.00 pp7 0 pp9 0 pp11 . .
i2 + 3. pp5 3.00 pp7 .3 pp9 1 pp11 . .

s ; End section, reset clock

; Second part instr 1 outputs FM for instr 2 (p10, p11)

; p4=fund clip skew
i1 0 1. 6.11 6.06 0 1 -.3 +.3
i1 + 1. pp5 2.03 pp7 0 pp9 0
i1 + 1. pp5 7.11 pp7 .8 pp9 .8
i1 + 2. pp5 8.11 pp7 0 pp9 .4
i1 + .5 pp5 6.05 pp7 .5 pp9 -.6
i1 + 2.5 pp5 6.05 pp7 .4 pp9 .8

; p4=fund clip skew p10=FM p12=sync
i2 0 .5 8.08 6.06 0 .3 -.5 +.5 0 3 0
i2 + .5 pp5 6.03 pp7 .5 pp9 -.6 pp11 2 .
i2 + .5 pp5 5.11 pp7 1 pp9 .5 pp11 < .
i2 + .5 pp5 6.01 pp7 .8 pp9 -.5 pp11 1 .
i2 + .5 pp5 5.11 pp7 .5 pp9 .3 pp11 < .
i2 + .5 pp5 9.04 pp7 .1 pp9 -.3 pp11 .5 .
i2 + 2. pp5 8.11 pp7 .4 pp9 .4 pp11 2 .
i2 + 3. pp5 8.11 pp7 0 pp9 0 pp11 3 .

```

e

```
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : Rasmus Ekman  
Novembre 2017

Nouveau dans la version 6.10

# sr

sr — Fixe la taux d'échantillonnage audio.

## Description

Ces instructions sont des *affectations* de valeurs globales réalisées au début d'un orchestre, avant que tout bloc d'instrument ne soit défini. Leur fonction est de fixer certaines *variables* dont le nom est un mot réservé et qui sont nécessaires à l'exécution. Une fois fixés, ces mots réservés peuvent être utilisés dans des expressions n'importe où dans l'orchestre.

## Syntaxe

```
sr = iarg
```

## Initialisation

*sr* = (facultatif) -- fixe le taux d'échantillonnage à *iarg* échantillons par seconde par canal. La valeur par défaut est 44100.

De plus, toute *variable globale* peut être initialisée par une *instruction de la période d'initialisation* n'importe où avant la première *instruction instr*. Toutes les affectations ci-dessus sont exécutées dans l'instrument 0 (passe-i seulement) au début de l'exécution réelle.

Depuis la version 3.46 de Csound, on peut omettre *sr*. Le taux d'échantillonnage sera calculé à partir de *kr* et de *ksmps*, mais le résultat doit être une valeur entière. Si aucune de ces valeurs globales n'est définie, le taux d'échantillonnage par défaut sera 44100. Habituellement, vous utiliserez une valeur supportée par votre carte son, comme 44100 ou 48000, sinon, le résultat audio généré par csound risque d'être injouable, ou bien vous aurez une erreur si vous essayez une exécution en . Vous pouvez naturellement utiliser un taux d'échantillonnage comme 96000, pour un rendu différé, même si votre carte son ne le supporte pas. Csound générera un fichier valide jouable sur des systèmes offrant cette possibilité.

## Exemples

```
sr = 10000
kr = 500
ksmps = 20
gil = sr/2.
ga init 0
itranspose = octpch(.01)
```

Voici un autre exemple de l'opcode sr. Il utilise le fichier *sr.csd* [examples/sr.csd].

### Exemple 939. Exemple de l'opcode sr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac ;;;realtime audio out
```

```

; -iadc      ;;uncomment -iadc if real audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;use sr to find maximum harmonics

ihar = int(sr/2/p4) ; maximum possible number of harmonics w/o aliasing
prints "maximum number of harmonics = %d \\n", ihar
kenv linen .5, 1, p3, .2 ; envelope
asig buzz kenv, p4, ihar, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 4096 10 1 ;sine wave

i 1 0 3 100 ;different frequencies
i 1 + 3 1000
i 1 + 3 10000
e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

maximum number of harmonics = 240
maximum number of harmonics = 24
maximum number of harmonics = 2

```

## Voir aussi

*kr, ksmps, nchnls, nchnls\_i.*

# statevar

statevar — Un filtre à variable d'état.

## Description

*statevar* est une nouvelle implémentation numérique du filtre analogique à variable d'état. Ce filtre a quatre sorties simultanées : passe-haut, passe-bas, passe-bande et réjecteur de bande. Ce filtre utilise le sur-échantillonnage pour obtenir une résonance plus raide (sur-échantillonné 3 fois par défaut). Il comprend un limiteur de résonance qui empêche le filtre de devenir instable.

## Syntaxe

```
ahp,alp,abp,abr statevar ain, xcf, xq [, iosamps, istor]
```

## Initialisation

*iosamps* -- nom de fois que le sur-échantillonnage est utilisé dans le processus de filtrage. Cela détermine la raideur maximale de la résonance du filtre (Q). Plus de sur-échantillonnage permet des valeurs de Q plus élevées, moins de sur-échantillonnage limite la résonance. La valeur par défaut est 3 fois (*iosamps*=0).

*istor* -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*ahp* -- signal de sortie du passe-haut.

*alp* -- signal de sortie du passe-bas.

*abp* -- signal de sortie du passe-bande.

*abr* -- signal de sortie du réjecteur de bande.

*asig* -- signal d'entrée.

*xcf* -- fréquence de coupure du filtre (taux-k ou taux-a).

*xq* -- Q du filtre (taux-k ou taux-a). Cette valeur est limitée en interne en fonction de la fréquence et du nombre de fois que le sur-échantillonnage est utilisé durant le processus (sur-échantillonnage de 3 fois par défaut).

## Exemples

Voici un exemple de l'opcode *statevar*. Il utilise le fichier *statevar.csd* [examples/statevar.csd].

### Exemple 940. Exemple de l'opcode *statevar*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o statevar.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1

kenv linseg 0,0.1,1, p3-0.2,1, 0.1, 0 ;declick envelope
asig buzz .6*kenv, 100, 100, 1
kf expseg 100, p3/2, 5000, p3/2, 1000 ;envelope for filter cutoff
ahp,alp,abp,abr statevar asig, kf, 4
      outs alp,ahp ; lowpass left, highpass right

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave

i1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini

Janvier 2005

Nouveau greffon dans la version 5

Janvier 2005.

Paramètres de taux audio introduits dans la version 6.02

Octobre 2013.

# stix

stix — Modèle semi-physique d'un son de baguette.

## Description

*stix* est un modèle semi-physique d'un son de baguette. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntaxe

```
ares stix iamp, idettack [, inum] [, idamp] [, imaxshake]
```

## Initialisation

*iamp* -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 30.

*idamp* (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

$$\text{damping\_amount} = 0,998 + (\text{idamp} * 0,002)$$

La valeur par défaut de *damping\_amount* est 0,998 ce qui signifie que la valeur par défaut de *idamp* est 0. Le maximum de *damping\_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 1,0.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

*imaxshake* (facultatif) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

## Exemples

Voici un exemple de l'opcode stix. Il utilise le fichier *stix.csd* [examples/stix.csd].

### Exemple 941. Exemple de l'opcode stix.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o stix.wav -W ;; for file output any platform
</CsOptions>
```

```
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1

idamp = p4 ;vary damping amount
asig stix .5, 0.01, 30, idamp
outs asig, asig

endin
</CsInstruments>
<CsScore>

i1 0 1 .3
i1 + 1 >
i1 + 1 >
i1 + 1 .95

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*cabasa, crunch, sandpaper, sekere*

## Crédits

Auteur : Perry Cook, fait partie de PhOLIES (Physically-Oriented Library of Imitated Environmental Sounds)

Adapté par John ffitich

Université de Bath, Codemist Ltd.

Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.



# STKBandedWG

STKBandedWG — STKBandedWG utilise des techniques de guide d'onde à bandes pour modéliser une variété de sons.

## Description

Opcode du greffon stkopd.

Cette opcode utilise des techniques de guide d'onde à bandes pour modéliser une variété de sons, comprenant des barres frottées, des verres et des bols.

## Syntaxe

```
asignal STKBandedWG ifrequency, iamplitude, [kpress, kv1[, kmot, kv2[, klfo, kv3[, klfodepth, kv4[, kv5[
```

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kpress* -- contrôleur 2, pression de l'archet. kv1 compris entre 0 et 127.

*kmot* -- contrôleur 4, mouvement de l'archet. kv2 compris entre 0 et 127.

*klfo* -- contrôleur 11, vitesse de l'oscillateur basse-fréquence. kv3 compris entre 0 et 127.

*klfodepth* -- contrôleur 1, intensité de l'oscillateur basse-fréquence. kv4 compris entre 0 et 127.

*kvel* -- contrôleur 128, vitesse de l'archet. kv5 compris entre 0 et 127.

*kstrk* -- contrôleur 64, frappe de l'archet. kv6 compris entre 0 et 127.

*kinstr* -- contrôleur 16, préréglages de l'instrument (0 = barre uniforme, 1 = barre accordée, 2 = harmonica de verre, 3 = bol tibétain). kv7 compris entre 0 et 3.



### Notes

Le code pour cet opcode vient directement de la classe *BandedWG* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [https://ccrma.stanford.edu/software/stk/classes.html]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire.

mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKBandedWG. Il utilise le fichier *STKBandedWG.csd* [examples/STKBandedWG.csd]

### Exemple 942. Exemple de l'opcode STKBandedWG.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKBandedWG.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq =    p4
kv1 line p5, p3, p6      ;pressure of bow
kenv line 1, p3, 0

asig STKBandedWG cpspch(ifrq), 1, 2, kv1, 4, 100, 11, 0, 1, 0, 64, 100, 128, 120, 16, 2
asig = asig * kenv        ;simple envelope
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 10 5.00 100 0
i 1 10 8 6.03 10 .
i 1 20 5 7.05 50 127

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael Gogins (d'après Georg Essl)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# STKBeeThree

STKBeeThree — Instrument STK de synthèse MF ressemblant à un orgue type Hammond.

## Description

Opcode du greffon `stkopd`.

Instrument STK de synthèse MF ressemblant à un orgue type Hammond.

Cet opcode a une simple topologie à 4 opérateurs, aussi référencée comme l'algorithme 8 du TX81Z. Il simule le son d'un orgue Hammond, et d'autre sons en relation.

## Syntaxe

```
asignal STKBeeThree ifrequency, iamplitude, [kop4, kv1[, kop3, kv2[, klfo, kv3[, klfodepth, kv4[, kadsr
```

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kop4* -- contrôleur 2, gain et feedback de l'opérateur 4. *kv1* compris entre 0 et 127.

*kop3* -- contrôleur 4, gain de l'opérateur 3. *kv2* compris entre 0 et 127.

*klfo* -- contrôleur 11, vitesse de l'oscillateur basse-fréquence. *kv3* compris entre 0 et 127.

*klfodepth* -- contrôleur 1, intensité de l'oscillateur basse-fréquence. *kv4* compris entre 0 et 127.

*kadsr* -- contrôleur 128, cible de l'ADSR 2 et 4. *kv5* compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *BeeThree* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [https://ccrma.stanford.edu/software/stk/classes.html]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKBeeThree. Il utilise le fichier *STKBeeThree.csd* [examples/STKBeeThree.csd].

### Exemple 943. Exemple de l'opcode STKBeeThree.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKBeeThree.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kv1 = p6      ;feedback of operator 4
kv2 line p4, p3, p5      ;gain of operator 3
kv5 line 0, p3, 100
ipch = p7

asig STKBeeThree cpspch(ipch), 1, 2, kv1, 4, kv2, 11, 50, 1, 0, 128, kv5
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 2 20 100 127 8.00
i 1 + 8 120 0 0 6.09
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# STKBlowBotl

STKBlowBotl — STKBlowBotl utilise un résonateur de Helmholtz (filtre biquadratique) avec une excitation par jet polynomial.

## Description

Opcode du greffon stkopd.

Cet opcode implémente un résonateur de Helmholtz (filtre biquadratique) avec une excitation par jet polynomial (à la Cook).

## Syntaxe

```
asignal STKBlowBotl ifrequency, iamplitude, [knoise, kv1[, klfo, kv2[, klfodepth, kv3[, kvol, kv4]]]]
```

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*knoise* -- contrôleur 4, gain du bruit. kv1 compris entre 0 et 127.

*klfo* -- contrôleur 11, vitesse de l'oscillateur basse-fréquence. kv2 compris entre 0 et 127.

*klfodepth* -- contrôleur 1, intensité de l'oscillateur basse-fréquence. kv3 compris entre 0 et 127.

*kvol* -- contrôleur 128, volume. kv4 compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *BlowBotl* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKBlowBotl. Il utilise le fichier *STKBlowBotl.csd* [exemples/STKBlowBotl.csd].

**Exemple 944. Exemple de l'opcode STKBlowBotl.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKBlowBotl.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ipch = p4
kv1 line p5, p3, p6      ;gain of noise
kv4 line 100, p3, 70     ;volume

asig STKBlowBotl cpspch(ipch), 1, 4, kv1, 11, 10, 1, 50, 128, kv4
asig = asig * .5         ;too loud
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 2 9.00 20 100
i 1 + 5 8.03 120 0
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# STKBlowHole

STKBlowHole — Modèle physique de clarinette STK avec un trou de registre et un trou d'intonation.

## Description

Opcode du greffon stkopd.

Cet opcode est basé sur le modèle de la clarinette, avec en plus l'implémentation d'un trou de registre à deux ports et d'un trou d'intonation à trois ports.

Dans cette implémentation, les distances entre trou de registre/anche et trou d'intonation/pavillon sont fixes. Ainsi, le trou d'intonation et le trou de registre auront tous deux une influence variable sur la fréquence jouée, qui dépend de la longueur de la colonne d'air. De plus, la fréquence la plus haute pouvant être jouée est limitée par ces longueurs fixes.

## Syntaxe

```
asignal STKBlowHole ifrequency, iamplitude, [kreed, kv1[, knoise, kv2[, khole, kv3[, kreg, kv4[, kbreath
```

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kreed* -- contrôleur 2, raideur de l'anche. kv1 compris entre 0 et 127.

*knoise* -- contrôleur 4, gain du bruit. kv2 compris entre 0 et 127.

*khole* -- contrôleur 11, état du trou d'intonation. kv3 compris entre 0 et 127.

*kreg* -- contrôleur 1, état du registre. kv4 compris entre 0 et 127.

*kbreath* -- contrôleur 128, pression du souffle. kv5 compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *BlowHole* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [https://ccrma.stanford.edu/software/stk/classes.html]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire

mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKBlowHole opcode. Il utilise le fichier *STKBlowHole.csd* [examples/STKBlowHole.csd].

### Exemple 945. Exemple de l'opcode STKBlowHole opcode.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKBlowHole.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ipch = p4
kv1 = p7      ;stiffness of reed
kv3 line p5, p3, p6      ;state of tonehole

asig STKBlowHole cpspch(ipch), 1, 2, kv1, 4, 100, 11, kv3, 1, 10, 128, 100
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 4 10.00 20 127 100
i 1 + 7 6.09 120 0 10
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.



# STKBowed

STKBowed — STKBowed est un instrument à corde frottée.

## Description

Opcode du greffon stkopd.

*STKBowed* est un instrument à corde frottée, utilisant un modèle à guide d'onde.

## Syntaxe

```
asignal STKBowed ifrequency, iamplitude, [kpress, kv1[, kpos, kv2[, klfo, kv3[, klfodepth, kv4[, kvol,
```

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kpress* -- contrôleur 2, pression de l'archet. kv1 compris entre 0 et 127.

*kpos* -- contrôleur 4, position de l'archet. kv2 compris entre 0 et 127.

*klfo* -- contrôleur 11, vitesse de l'oscillateur basse-fréquence. kv3 compris entre 0 et 127.

*klfodepth* -- contrôleur 1, intensité de l'oscillateur basse-fréquence. kv4 compris entre 0 et 127.

*kvol* -- contrôleur 128, volume. kv5 compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *Bowed* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKBowed. Il utilise le fichier *STKBowed.csd* [examples/STKBowed.csd].

**Exemple 946. Exemple de l'opcode STKBowed.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKBowed.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ipch = p4
kv2 = p7      ;position on bow
kv1 line p5, p3, p6      ;bow pressure
kv4 line 0, p3, 7      ;depth of low-frequency oscillator

asig STKBowed cpspch(ipch), 1, 2, kv1, 4, kv2, 11, 40, 1, kv4, 128, 100
asig = asig*2      ;amplify
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 15 6.00 20 100 115
i 1 17 3 7.00 120 0 0
i 1 21 3 7.09 120 0 30
i 1 21 4 7.03 50 0 0
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# STKBrass

STKBrass — STKBrass est un simple instrument de la famille des cuivres.

## Description

Opcode du greffon stkopd.

*STKBrass* utilise un modèle simple à guide d'onde d'un instrument de la famille des cuivres, à la Cook.

## Syntaxe

```
asignal STKBrass ifrequency, iamplitude, [klip, kv1[, kslide, kv2[, klfo, kv3[, klfodepth, kv4[, kvol,
```

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*klip* -- contrôleur 2, tension des lèvres. kv1 compris entre 0 et 127.

*kslide* -- contrôleur 4, longueur de la coulisse. kv2 compris entre 0 et 127.

*klfo* -- contrôleur 11, vitesse de l'oscillateur basse-fréquence. kv3 compris entre 0 et 127.

*klfodepth* -- contrôleur 1, intensité de l'oscillateur basse-fréquence. kv4 compris entre 0 et 127.

*kvol* -- contrôleur 128, volume. kv5 compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *Brass* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKBrass. Il utilise le fichier *STKBrass.csd* [exemples/STKBrass.csd].

**Exemple 947. Exemple de l'opcode STKBrass.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKBrass.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq = p4
kv1 line p5, p3, p6      ;lip tension
kv3 line p7, p3, p8      ;speed of low-frequency oscillator

asig STKBrass cpspch(ifrq), 1, 2, kv1, 4, 100, 11, kv3, 1, 10, 128, 40
asig = asig *3           ;amplify
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 2 8.05 100 120 50 0
i 1 + 3 9.00 80 82 10 0
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# STKClarinet

STKClarinet — STKClarinet utilise un modèle physique simple de clarinette.

## Description

Opcode du greffon stkopd.

*STKClarinet* utilise un modèle physique simple de clarinette.

## Syntaxe

assignal **STKClarinet** ifrequency, iamplitude, [kstiff, kv1[, knoise, kv2[, klfo, kv3[, klfodepth, kv4[, k

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kstiff* -- contrôleur 2, raideur de l'anche. kv1 compris entre 0 et 127.

*knoise* -- contrôleur 4, gain du bruit. kv2 compris entre 0 et 127.

*klfo* -- contrôleur 11, vitesse de l'oscillateur basse-fréquence. kv3 compris entre 0 et 127.

*klfodepth* -- contrôleur 1, intensité de l'oscillateur basse-fréquence. kv4 compris entre 0 et 127.

*kbreath* -- contrôleur 128, pression du souffle. kv5 compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *Clarinet* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKClarinet. Il utilise le fichier *STKClarinet.csd* [exemples/STKClarinet.csd].

## Exemple 948. Exemple de l'opcode STKClarinet.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKclarinet.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq = p4
kv5 = p5      ;breath pressure
kv1 line p6, p3, p7 ;reed stiffness
asig STKClarinet cpspch(p4), 1, 2, kv1, 4, 100, 11, 60, 1, 10, 128, kv5
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 3 8.00 100 127 10
i 1 + 10 8.08 80 60 100
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*STKFlute.*

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
 Irreducible Productions  
 New York, NY

Nouveau dans la version 5.11 de Csound.

# STKDrummer

STKDrummer — STKDrummer est un synthétiseur à échantillon de tambour.

## Description

Opcode du greffon stkopd.

*STKDrummer* est un synthétiseur à échantillon de tambour qui utilise des formes d'onde brutes et des filtres à un pôle. Les fichiers des formes d'ondes brutes sont échantillonnés à 22050 Hz, mais seront interpolés de manière appropriée pour les autres taux d'échantillonnage.

## Syntaxe

asignal **STKDrummer** ifrequency, iamplitude

## Initialisation

*ifrequency* -- Echantillons joués.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).



### Note

Le code pour cet opcode vient directement de la classe *Drummer* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

## Exemples

Voici un exemple de l'opcode STKDrummer. Il utilise le fichier *STKDrummer.csd* [examples/STKDrummer.csd].

### Exemple 949. Exemple de l'opcode STKDrummer.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      -M0   ;;realtime audio out and midi in
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKDrummer.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;STK Drummer - has no controllers but plays samples (11)
```

```
icps cpsmidi
iamp ampmidi 1
asig STKDrummer icps, iamp
  outs asig, asig
endin

</CsInstruments>
<CsScore>
; play 5 minutes
f0 300
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*STKClarinet.*

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.



# STKFlute

STKFlute — STKFlute utilise un simple modèle physique de flûte.

## Description

Opcode du greffon stkopd.

*STKFlute* utilise un simple modèle physique de flûte. Le modèle du jet utilise un polynôme, à la Cook.

## Syntaxe

```
asignal STKFlute ifrequency, iamplitude, [kjet, kv1[, knoise, kv2[, klfo, kv3[, klfodepth, kv4[, kbreat
```

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kjet* -- contrôleur 2, délai du jet. kv1 compris entre 0 et 127.

*knoise* -- contrôleur 4, gain du bruit. kv2 compris entre 0 et 127.

*klfo* -- contrôleur 11, vitesse de l'oscillateur basse-fréquence. kv3 compris entre 0 et 127.

*klfodepth* -- contrôleur 1, intensité de l'oscillateur basse-fréquence. kv4 compris entre 0 et 127.

*kbreath* -- contrôleur 128, pression du souffle. kv5 compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *Flute* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKFlute. Il utilise le fichier *STKFlute.csd* [examples/STKFlute.csd].

## Exemple 950. Exemple de l'opcode STKFlute.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKFlute.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq = p4
kv1 line p5, p3, p6      ;jet delay
kv4 line 0, p3, 100      ;vibrato depth

asig STKFlute cpspch(ifrq), 1, 2, kv1, 4, 100, 11, 100, 1, kv4, 128, 100
outs asig, asig
endin

</CsInstruments>
<CsScore>
i 1 0 5 8.00 0 0
i 1 7 3 9.00 20 120
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*STKClarinet.*

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
 Irreducible Productions  
 New York, NY

Nouveau dans la version 5.11 de Csound.

# STKFMVoices

STKFMVoices — STKFMVoices est un instrument de synthèse de voix FM.

## Description

Opcode du greffon stkopd.

STKFMVoices est un instrument de synthèse de voix FM. Il a trois porteuses et un modulateur commun, aussi connu comme l'algorithme 6 du TX81Z.

## Syntaxe

asignal **STKFMVoices** ifrequency, iamplitude, [kvowel, kv1[, kspec, kv2[, klfo, kv3[, klfodepth, kv4[, ka

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kvowel* -- contrôleur 2, voyelle. kv1 compris entre 0 et 127.

*kspec* -- contrôleur 4, pente spectrale. kv2 compris entre 0 et 127.

*klfo* -- contrôleur 11, vitesse de l'oscillateur basse-fréquence. kv3 compris entre 0 et 127.

*klfodepth* -- contrôleur 1, intensité de l'oscillateur basse-fréquence. kv4 compris entre 0 et 127.

*kadsr* -- contrôleur 128, cible de l'ADSR 2 et 4. kv5 compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *FMVoices* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKFMVoices. Il utilise le fichier *STKFMVoices.csd* [examples/STKFMVoices.csd].

**Exemple 951. Exemple de l'opcode STKFMVoices.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKFMVoices.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq = p4
kv1 line p5, p3, p6      ;vowel
kv2 line p7, p3, p8      ;specral tilt

asig STKFMVoices cpspch(ifrq), 1, 2, kv1, 4, kv2, 11, 10, 1, 10, 128, 50
asig = asig * 4          ;amplify
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 5 5.00 10 120 0 0
i 1 + 2 8.00 80 82 127 0
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*STKBeeThree.*

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# STKHeavyMetl

STKHeavyMetl — STKHeavyMetl produit des sons de type "heavy metal".

## Description

Opcode du greffon stkopd.

*STKHeavyMetl* produit des sons de type "heavy metal" en utilisant la synthèse FM. Il utilise 3 opérateurs en cascade avec modulation en boucle de retour, aussi connu comme l'algorithme 3 du TX81Z.

## Syntaxe

asignal **STKHeavyMetl** ifrequency, iamplitude, [kmod, kv1[, kcross, kv2[, klfo, kv3[, klfodepth, kv4[, kac

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kmod* -- contrôleur 2, index total du modulateur. kv1 compris entre 0 et 127.

*kcross* -- contrôleur 4, fondu enchaîné du modulateur. kv2 compris entre 0 et 127.

*klfo* -- contrôleur 11, vitesse de l'oscillateur basse-fréquence. kv3 compris entre 0 et 127.

*klfodepth* -- contrôleur 1, intensité de l'oscillateur basse-fréquence. kv4 compris entre 0 et 127.

*kadsr* -- contrôleur 128, cible de l'ADSR 2 et 4. kv5 compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *HeavyMetl* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKHeavyMetl. Il utilise le fichier *STKHeavyMetl.csd* [examples/STKHeavy-Metl.csd]

**Exemple 952. Exemple de l'opcode STKHeavyMetl.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKHeavyMetl.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq = p4
kv1 line p5, p3, p6      ;Total Modulator Index
kv2 line p7, p3, 0      ;Modulator Crossfade

asig STKHeavyMetl cspspch(ifrq), 1, 2, kv1, 4, kv2, 11, 0, 1, 100, 128, 40
outs asig, asig
endin

</CsInstruments>
<CsScore>
i 1 0 7 8.05 100 0 100
i 1 3 7 9.03 20 120 0
i 1 3 .5 8.05 20 120 0
i 1 4 .5 9.09 20 120 0
i 1 5 3 9.00 20 120 0

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*STKWurley.*

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# STKMandolin

STKMandolin — STKMandolin produit des sons de type mandoline.

## Description

Opcode du greffon stkopd.

*STKMandolin* produit des sons de type mandoline, en utilisant des techniques de "synthèse commutée" pour modéliser une mandoline.

## Syntaxe

```
asignal STKMandolin ifrequency, iamplitude, [kbody, kv1[, kpos, kv2[, ksus, kv3[, kdetune, kv4[, kmic,
```

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kbody* -- contrôleur 2, taille du corps. kv1 compris entre 0 et 127.

*kpos* -- contrôleur 4, position de pincement. kv2 compris entre 0 et 127.

*ksus* -- contrôleur 11, entretien de la corde. kv3 compris entre 0 et 127.

*kdetune* -- contrôleur 1, désaccordage de la corde. kv4 compris entre 0 et 127.

*kmic* -- contrôleur 128, position du microphone. kv5 compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *Mandolin* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKMandolin. Il utilise le fichier *STKMandolin.csd* [exemples/STKMandolin.csd].

**Exemple 953. Exemple de l'opcode STKMandolin.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKMandolin.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq = p4
kv1 line p5, p3, p6      ;body size
kv3 = p7      ;sustain

asig STKMandolin cpspch(ifrq), 1, 2, kv1, 4, 10, 11, kv3, 1, 100, 128, 100
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 .3 7.00 100 0 20
i 1 + . 8.00 10 100 20
i 1 + . 8.00 100 0 120
i 1 + 4 8.00 10 10 127
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*STKPlucked.*

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.



# STKModalBar

STKModalBar — STKModalBar est un instrument à lame résonante.

## Description

Opcode du greffon stkopd.

Cet opcode est un instrument de barre résonante. Il a un certain nombre de différents instruments à lame frappée.

## Syntaxe

```
asignal STKModalBar ifrequency, iamplitude, [khard, kv1[, kpos, kv2[, klfo, kv3[, klfodepth, kv4[, kmix
```

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*khard* -- contrôleur 2, dureté de la baguette. kv1 compris entre 0 et 127.

*kpos* -- contrôleur 4, position de la baguette. kv2 compris entre 0 et 127.

*klfo* -- contrôleur 11, vitesse de l'oscillateur basse-fréquence. kv3 compris entre 0 et 127.

*klfodepth* -- contrôleur 1, intensité de l'oscillateur basse-fréquence. kv4 compris entre 0 et 127.

*kmix* -- contrôleur 8, mélange direct baguette. kv5 compris entre 0 et 127.

*kvol* -- contrôleur 128, volume. kv6 compris entre 0 et 127.

*kinstr* -- contrôleur 16, pré réglage d'instrument (0 = marimba, 1 = vibraphone, 2 = agogo, 3 = bois1, 4 = réso, 5 = bois2, 6 = battements, 7 = deux fixes, 8 = sourd). kv7 compris entre 0 et 16.



### Notes

Le code pour cet opcode vient directement de la classe *ModalBar* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [https://ccrma.stanford.edu/software/stk/classes.html]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire

mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKModalBar. Il utilise le fichier *STKModalBar.csd* [examples/STKModalBar.csd].

### Exemple 954. Exemple de l'opcode STKModalBar.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;RT audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKModalBar.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq = p4
kv1 = p5      ;stick hardness      ;

asig STKModalBar cpspch(ifrq), 1, 2, kv1, 4, 120, 11, 0, 1, 0, 8, 10, 16, 1
asig = asig * 3      ;amplify
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 2 8.00 0
i 1 + 2 8.05 120
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael Gogins (d'après Georg Essl)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# STKMoog

STKMoog — STKMoog produit des sons de filtre à balayage comme sur un moog.

## Description

Opcode du greffon `stkopd`.

*STKMoog* produit des sons de filtre à balayage comme sur un moog, en utilisant une onde pour l'attaque, une onde pour la boucle et une enveloppe ADSR, et en ajoutant deux filtres à balayage de formant.

## Syntaxe

```
asignal STKMoog ifrequency, iamplitude, [kq, kv1[, krate, kv2[, klfo, kv3[, klfodepth, kv4[, kvol, kv5]
```

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kq* -- contrôleur 2, Q du filtre. kv1 compris entre 0 et 127.

*krate* -- contrôleur 4, taux de balayage du filtre. kv2 compris entre 0 et 127.

*klfo* -- contrôleur 11, vitesse de l'oscillateur basse-fréquence. kv3 compris entre 0 et 127.

*klfodepth* -- contrôleur 1, intensité de l'oscillateur basse-fréquence. kv4 compris entre 0 et 127.

*kvol* -- contrôleur 128, volume. kv5 compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *Moog* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKMoog. Il utilise le fichier *STKMoog.csd* [examples/STKMoog.csd].

**Exemple 955. Exemple de l'opcode STKMoog.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKMoog.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq = p4
kv1 line p5, p3, p6      ;filter Q

asig STKMoog cpspch(ifrq), 1, 2,kv1, 4, 120, 11, 40, 1, 1, 128, 120
asig = asig * .3      ;too loud
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 .5 6.00 100 0
i 1 + . 5.05 10 127
i 1 + . 7.06 100 0
i 1 + 3 7.00 10 10
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# STKPercFlut

STKPercFlut — STKPercFlut est une flûte percussive réalisée par synthèse FM.

## Description

Opcode du greffon stkopd.

*STKPercFlut* est une flûte percussive réalisée par synthèse FM. L'instrument utilise un algorithme semblable à l'algorithme 4 du TX81Z.

## Syntaxe

asignal **STKPercFlut** ifrequency, iamplitude, [kmod, kv1[, kcross, kv2[, klfo, kv3[, klfodepth, kv4[, ka

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kmod* -- contrôleur 2, index total du moduleur. kv1 compris entre 0 et 127.

*kcross* -- contrôleur 4, fondu enchaîné du modulateur. kv2 compris entre 0 et 127.

*klfo* -- contrôleur 11, vitesse de l'oscillateur basse-fréquence. kv3 compris entre 0 et 127.

*klfodepth* -- contrôleur 1, intensité de l'oscillateur basse-fréquence. kv4 compris entre 0 et 127.

*kadsr* -- contrôleur 128, cible de l'ADSR 2 et 4. kv5 compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *PercFlut* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKPercFlut. Il utilise le fichier *STKPercFlut.csd* [exemples/STKPercFlut.csd].

**Exemple 956. Exemple de l'opcode STKPercFlut.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKPercFlut.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq = p4
kv1 line p5, p3, p6      ;Total Modulator Index
kv2 line p7, p3, 0      ;Modulator Crossfade

asig STKPercFlut cpspch(ifrq), 1, 2, kv1, 4, kv2, 11, 0, 1, 100, 128, 40
outs asig, asig
endin

</CsInstruments>
<CsScore>
i 1 0 7 8.05 100 0 100
i 1 3 7 9.03 20 120 0
i 1 3 .5 8.05 20 120 0
i 1 4 .5 9.09 20 120 0
i 1 5 3 9.00 20 120 0

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# STKPlucked

STKPlucked — STKPlucked utilise un modèle physique de corde pincée.

## Description

Opcode du greffon stkopd.

*STKPlucked* utilise un modèle physique basé sur l'algorithme de Karplus-Strong.

## Syntaxe

asignal **STKPlucked** ifrequency, iamplitude

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).



### Note

Le code pour cet opcode vient directement de la classe *Plucked* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

## Exemples

Voici un exemple de l'opcode STKPlucked. Il utilise le fichier *STKPlucked.csd* [exemples/STKPlucked.csd].

### Exemple 957. Exemple de l'opcode STKPlucked.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKPlucked.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;STKPlucked - has no controllers

ifrq = p4
```

```
asig STKPlucked cspch(ifrq), 1
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 2 6.00
i 1 + 8 5.00
i 1 + .5 8.00
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*STKSitar.*

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.



# STKResonate

STKResonate — STKResonate est un filtre à formant piloté par du bruit.

## Description

Opcode du greffon stkopd.

*STKResonate* est un filtre à formant piloté par du bruit. Cet instrument contient une source de bruit qui excite un filtre à résonance biquadratique, avec contrôle du volume par une enveloppe ADSR.

## Syntaxe

```
asignal STKResonate ifrequency, iamplitude, [kfreq, kv1[, kpole, kv2[, knotch, kv3[, kzero, kv4[, kenv,
```

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kfreq* -- contrôleur 2, fréquence de résonance. kv1 compris entre 0 et 127.

*kpole* -- contrôleur 4, rayon des pôles. kv2 compris entre 0 et 127.

*knotch* -- contrôleur 11, fréquence d'encoche. kv3 compris entre 0 et 127.

*kzero* -- contrôleur 1, rayon des zéros. kv4 compris entre 0 et 127.

*kenv* -- contrôleur 128, gain de l'enveloppe. kv5 compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *Resonate* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKResonate. Il utilise le fichier *STKResonate.csd* [examples/STKResonate.csd].

**Exemple 958. Exemple de l'opcode STKResonate.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKResonate.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; frequency and amplitude of STKResonate have no effect

kv2 = p4      ;pole radii
kv1 line 100, p3, 0 ;resonance freq + notch freq
kv3 = kv1
asig STKResonate 1, 1, 2, kv1, 4, kv2, 1, 10, 11, kv3, 128, 100
asig = asig * .3 ;too loud
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 1 0
i 1 + . >
i 1 + . >
i 1 + . >
i 1 + . 120
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# STKRhodey

STKRhodey — Instrument STK, piano électrique comme un Fender Rhodes, réalisé par synthèse FM.

## Description

Opcodes du greffon stkopd.

Instrument STK, piano électrique comme un Fender Rhodes, réalisé par synthèse FM.

Cet opcode implémente un instrument basé sur deux paires simples de MF additionnées ensemble, aussi connu comme l'algorithme 5 du TX81Z de Yamaha. Il simule le son d'un piano électrique Rhodes, et d'autres sons approchants.

## Syntaxe

```
asignal STKRhodey ifrequency, iamplitude, [kmod, kv1[, kcross, kv2[, klfo, kv3[, klfodepth, kv4[, kadsr
```

## Initialisation

*ifrequency* -- Frequency of note played, in Hertz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kmod* -- contrôleur 2, index 1 du modulateur. kv1 compris entre 0 et 127.

*kcross* -- contrôleur 4, fondu enchaîné des sorties. kv2 compris entre 0 et 127.

*klfo* -- contrôleur 11, vitesse de l'oscillateur basse-fréquence. kv3 compris entre 0 et 127.

*klfodepth* -- contrôleur, intensité de l'oscillateur basse-fréquence. kv4 compris entre 0 et 127.

*kadsr* -- contrôleur 128, cible de l'ADSR 2 et 4. kv5 compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *Rhodey* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [https://ccrma.stanford.edu/software/stk/classes.html]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode `STKRhodey`. Il utilise le fichier `STKRhodey.csd` [examples/STKRhodey.csd].

### Exemple 959. Exemple de l'opcode `STKRhodey`.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKRhodey.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq = p4
kv1 line p5, p3, p6      ;(FM) Modulator Index One
kv5 = p7                ;ADSR 2 and 4 target
asig STKRhodey cpspch(p4), 1, 2, kv1, 4, 10, 11, 100, 1, 3, 128, kv5
  outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 .5 7.00 75 0 0
i 1 + . 8.00 120 0 120
i 1 + 1 6.00 50 120 50
i 1 + 4 8.00 10 120 100
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*STKWurley*.

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# STKSaxofony

STKSaxofony — STKSaxofony simule un instrument à anche et perce conique.

## Description

Opcode du greffon `stkopd`.

*STKSaxofony* simule un instrument à anche et perce conique. Cet opcode utilise un instrument "hybride" à guide d'onde numérique qui peut générer une variété de sons de type vents. Il est aussi connu sous le nom de modèle de "corde soufflée". La section du guide d'onde est essentiellement celle d'une corde, avec une extrémité rigide et l'autre dissipative. La fonction non-linéaire est une table d'anche. La corde peut être "soufflée" n'importe où entre les extrémités, tandis que, comme pour les cordes, il est impossible d'exciter le système à l'une de ses extrémités. Si l'excitation a lieu au milieu de la corde, le son est celui d'une clarinette. Aux points plus proches du "chevalet", le son se rapproche de celui d'un saxophone.

## Syntaxe

```
asignal STKSaxofony ifrequency, iamplitude, [kstiff, kv1[, kapert, kv2[, kblow, kv3[, knoise, kv4[, klf
```

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kstiff* -- contrôleur 2, raideur de l'anche. kv1 compris entre 0 et 127.

*kapert* -- contrôleur 26, ouverture de l'anche. kv2 compris entre 0 et 127.

*kblow* -- contrôleur 11, position du souffle. kv3 compris entre 0 et 127.

*knoise* -- contrôleur 4, gain du bruit. kv4 compris entre 0 et 127.

*klfo* -- contrôleur 29, vitesse de l'oscillateur basse-fréquence. kv5 compris entre 0 et 127.

*klfodepth* -- contrôleur 1, intensité de l'oscillateur basse-fréquence. kv6 compris entre 0 et 127.

*kbreath* -- contrôleur 128, pression du souffle. kv7 compris entre 0 et 127.



## Notes

Le code pour cet opcode vient directement de la classe *Saxofony* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [https://ccrma.stanford.edu/software/stk/classes.html]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro

de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKSaxofony. Il utilise le fichier *STKSaxofony.csd* [examples/STKSaxofony.csd].

### Exemple 960. Exemple de l'opcode STKSaxofony.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKSaxofony.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifreq = p4
kv1 = p5      ;reed stiffness
kv3 line p6, p3, p7      ;blow position
kv6 line 0, p3, 127      ;depth of low-frequency oscillator

asig STKSaxofony cpspch(p4), 1, 2, kv1, 4, 100, 26, 70, 11, kv3, 1, kv6, 29, 100
asig = asig * .5      ;too loud
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 3 6.00 30 100 10
i 1 + . 8.00 30 100 100
i 1 + . 7.00 90 127 30
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael Gogins (d'après Georg Essl)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# STKShakers

STKShakers — STKShakers simule des sons environnementaux de collisions entre de multiples objets indépendants produisant des sons.

## Description

Opcodes du greffon `stkopd`.

*STKShakers* constitue un ensemble d'instruments *PhISEM* et *PhOLIES* : *PhISEM* (Physically Inspired Stochastic Event Modeling = modélisation physique inspirée d'évènements aléatoires) est une approche algorithmique de simulation de collisions entre de multiples objets indépendants produisant des sons. On peut simuler des maracas, un chekeré, une cabasa, un carillon en bambou, des gouttes d'eau, un tambourin, des grelots et un güiro. Voir *Shaker Controllers to control PhISEM* [<http://soundlab.cs.princeton.edu/research/controllers/shakers/>]. *PhOLIES* (Physically-Oriented Library of Imitated Environmental Sounds = bibliothèque physiquement orientée d'imitation de sons environnementaux) est une approche similaire pour la synthèse de sons environnementaux. Elle simule des petites branches qui se brisent, de la neige qui crisse (ou pas), une déchirure, du papier de verre, etc...

## Syntaxe

```
asignal STKShakers ifrequency, iamplitude, [kenerg, kv1[, kdecay, kv2[, kshake, kv3[, knum, kv4[, kres,
```

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kenerg* -- contrôleur 2, énergie des secousses. kv1 compris entre 0 et 127.

*kdecay* -- contrôleur 4, décroissance du système. kv2 compris entre 0 et 127.

*kshake* -- contrôleur 128, énergie des secousses. kv3 compris entre 0 et 127.

*knum* -- contrôleur 11, nombre d'objets. kv4 compris entre 0 et 127.

*kres* -- contrôleur 1, fréquence de résonance. kv5 compris entre 0 et 127.

*kinstr* -- contrôleur 1071, sélection de l'instrument (Maracas = 0, Cabasa = 1, Chekeré = 2, Güiro = 3, Gouttes d'eau = 4, Carillon de bambou = 5, Tambourin = 6, Grelots = 7, Branches = 8, Crissement = 9, Déchirure = 10, Papier de verre = 11, Canette de Coca = 12, Mug = 13, Penny + Mug = 14, Pièce de cinq cents + Mug = 15, Pièce de dix cents + Mug = 16, Pièce de vingt-cinq cents + Mug = 17, Franc + Mug = 18, Peso + Mug = 19, Gros galets = 20, Petits galets = 21, Carillon de bambou accordé = 22). kv6 compris entre 0 et 22.



### Notes

Le code pour cet opcode vient directement de la classe *Shakers* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK

ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKShakers. Il utilise le fichier *STKShakerscsd* [exemples/STKShakers.csd].

### Exemple 961. Exemple de l'opcode STKShakers.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKShakers.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq = p4

asig STKShakers cpspch(p4), 1, 2, 10, 4, 10, 11, 10, 1, 112, 128, 80, 1071, 5
asig = asig *2      ;amplify
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0.2 .5 7.00 75 0 20

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael Gogins (d'après Georg Essl)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.



# STKSimple

STKSimple — STKSimple est un instrument à table d'onde/bruit.

## Description

Opcode du greffon `stkopd`.

*STKSimple* est un instrument à table d'onde/bruit. Il combine une onde bouclée, une source de bruit, un filtre biquadratique à résonance, un filtre à un pôle et une enveloppe ADSR pour créer quelques sons intéressants.

## Syntaxe

```
asignal STKSimple ifrequency, iamplitude, [kpos, kv1[, kcross, kv2[, kenv, kv3[, kgain, kv4]]]]
```

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kpos* -- contrôleur 2, position du pôle du filtre. *kv1* compris entre 0 et 127.

*kcross* -- contrôleur 4, fondu enchaîné bruit/hauteur. *kv2* compris entre 0 et 127.

*kenv* -- contrôleur 11, taux de l'enveloppe. *kv3* compris entre 0 et 127.

*kgain* -- contrôleur 128, gain. *kv4* compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *Simple* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKSimple. Il utilise le fichier *STKSimple.csd* [examples/STKSimple.csd].

**Exemple 962. Exemple de l'opcode STKSimple.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKSimple.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq = p4
kv1 line p5, p3, p6      ;Filter Pole Position
kv2 line 20, p3, 90      ;Noise/Pitched Cross-Fade

asig STKSimple cpspch(p4), 1, 2, kv1, 4, kv2, 11, 10, 128, 120
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 5 7.00 100 0
i 1 + . 7.05 10 127
i 1 + . 8.03 100 0
i 1 + . 5.00 10 10
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*STKClarinet.*

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# STKSitar

STKSitar — STKSitar utilise un modèle physique de corde pincée.

## Description

Opcode du greffon stkopd.

*STKSitar* utilise un modèle physique de corde pincée basé sur l'algorithme de Karplus-Strong.

## Syntaxe

```
asignal STKSitar ifrequency, iamplitude
```

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).



### Note

Le code pour cet opcode vient directement de la classe *Sitar* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

## Exemples

Voici un exemple de l'opcode STKSitar. Il utilise le fichier *STKSitar.csd* [examples/STKSitar.csd].

### Exemple 963. Exemple de l'opcode STKSitar.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKSitar.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;STKSitar - has no controllers

ifrq = p4

asig STKSitar cpspch(p4), 1
asig = asig * 2 ;amplify
```

```
    outs asig, asig
  endin

</CsInstruments>
<CsScore>

i 1 0 4 6.00
i 1 + 2 7.05
i 1 + 7 5.05
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# STKStifKarp

STKStifKarp — STKStifKarp est un instrument à corde pincée rigide.

## Description

Opcode du greffon stkopd.

*STKStifKarp* est un instrument à corde pincée rigide. C'est un simple algorithme de corde pincée (Karplus Strong) avec des améliorations, comprenant le contrôle de la rigidité de la corde et la position de pincement. La rigidité est modélisée avec des filtres passe-tout.

## Syntaxe

```
asignal STKStifKarp ifrequency, iamplitude, [kpos, kv1[, ksus, kv2[, kstretch, kv3]]]
```

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kpos* -- contrôleur 4, position de lecture. kv1 compris entre 0 et 127.

*ksus* -- contrôleur 11, entretien de la corde. kv2 compris entre 0 et 127.

*kstretch* -- contrôleur 1, élasticité de la corde. kv3 compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *StifKarp* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKStifKarp. Il utilise le fichier *STKStifKarp.csd* [exemples/STKStifKarp.csd].

**Exemple 964. Exemple de l'opcode STKStifKarp.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKStifKarp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq = p4
kv1 line p6, p3, p7      ;Pickup Position
kv2 = p5      ;String Sustain

asig STKStifKarp cpspch(p4), 1, 4, kv1, 11, kv2, 1, 10
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 2 5.00 0 100 100
i 1 + 40 5.00 127 1 127
i 1 10 32 5.00 127 1 10
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# STKTubeBell

STKTubeBell — STKTubeBell est instrument de cloche-tube (cloche d'orchestre) par synthèse FM.

## Description

Opcode du greffon stkopd.

*STKTubeBell* est un instrument de cloche-tube (cloche d'orchestre) par synthèse FM. Il utilise deux paires FM simples additionnées ensemble, aussi connu comme l'algorithme 5 du TX81Z.

## Syntaxe

asignal **STKTubeBell** ifrequency, iamplitude, [kmod, kv1[, kcross, kv2[, klfo, kv3[, klfodepth, kv4[, kac

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kmod* -- contrôleur 2, index 1 du modulateur. kv1 compris entre 0 et 127.

*kcross* -- contrôleur 4, fondu enchaîné des sorties. kv2 compris entre 0 et 127.

*klfo* -- contrôleur 11, vitesse de l'oscillateur basse-fréquence. kv3 compris entre 0 et 127.

*klfodepth* -- contrôleur 1, intensité de l'oscillateur basse-fréquence. kv4 compris entre 0 et 127.

*kadsr* -- contrôleur 128, cible de l'ADSR 2 et 4. kv5 compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *TubeBell* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKTubeBell. Il utilise le fichier *STKTubeBell.csd* [exemples/STKTubeBell.csd].

**Exemple 965. Exemple de l'opcode STKTubeBell.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKTubeBell.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq = p4
kv2 line p6, p3, p7      ;Crossfade of Outputs
kv1 = p5      ;(FM) Modulator Index One

asig STKTubeBell cpspch(p4), 1, 2, kv1, 4, kv2, 11, 10, 1, 70, 128,50
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 2 7.05 0 100 100
i 1 + 4 9.00 127 127 30
i 1 + 1 10.00 127 12 30
i 1 + 3 6.08 127 1 100
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.



# STKVoicForm

STKVoicForm — STKVoicForm est un instrument de synthèse à quatre formants.

## Description

Opcodes du greffon stkopd.

*STKVoicForm* est un instrument de synthèse à quatre formants. Cet instrument contient une table d'onde d'excitation de voix chantée (onde bouclée avec vibrato aléatoire et périodique, lissage de fréquence, etc.), bruit d'excitation et quatre résonances complexes balayables. Des données de formant mesurées sont comprises et il y a suffisamment de données pour supporter la synthèse en parallèle ou en cascade. En virgule flottante, la synthèse en cascade est la plus naturelle et c'est donc celle-ci que l'on trouve ici.

## Syntaxe

`asignal STKVoicForm ifrequency, iamplitude, [kmix, kv1[, ksel, kv2[, klfo, kv3[, klfodepth, kv4[, kloud`

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kmix* -- contrôleur 2, mélange voisé/non voisé. kv1 compris entre 0 et 127.

*ksel* -- contrôleur 4, sélection voyelle/phonème. kv2 compris entre 0 et 127.

*klfo* -- contrôleur 11, vitesse de l'oscillateur basse-fréquence. kv3 compris entre 0 et 127.

*klfodepth* -- contrôleur 1, intensité de l'oscillateur basse-fréquence. kv4 compris entre 0 et 127.

*kloud* -- contrôleur 128, intensité (pente spectrale). kv5 compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *VoicForm* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [https://ccrma.stanford.edu/software/stk/classes.html]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode `STKVoicForm`. Il utilise le fichier `STKVoicForm.csd` [exemples/STKVoicForm.csd].

### Exemple 966. Exemple de l'opcode `STKVoicForm`.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKVoicForm.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq = p4
kv2 line p5, p3, p6      ;Vowel/Phoneme Selection

asig STKVoicForm cpspch(p4), 1, 2, 1, 4, kv2, 128, 100, 1, 10, 11, 100
asig = asig * .5        ;too loud
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 5 7.00 100 0
i 1 + 10 7.00 1 50
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# STKWhistle

STKWhistle — STKWhistle produit des sons de sifflet.

## Description

Opcode du greffon `stkopd`.

*STKWhistle* produit des sons de sifflet (de police). Il utilise un modèle hybride physique/spectral d'un sifflet de police (à la Cook).

## Syntaxe

`asignal STKWhistle ifrequency, iamplitude, [kmod, kv1[, knoise, kv2[, kfipfreq, kv3[, kfipgain, kv4[, kvol]]]]`

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kmod* -- contrôleur 2, modulation de la fréquence du souffle. *kv1* compris entre 0 et 127.

*knoise* -- contrôleur 4, gain du bruit. *kv2* compris entre 0 et 127.

*kfipfreq* -- contrôleur 11, modulation de fréquence du biseau. *kv3* compris entre 0 et 127.

*kfipgain* -- contrôleur 1, gain de la modulation du biseau. *kv4* compris entre 0 et 127.

*kvol* -- contrôleur 128, volume. *kv5* compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *Whistle* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode *STKWhistle*. Il utilise le fichier *STKWhistle.csd* [[examples/STKWhistle.csd](#)].

**Exemple 967. Exemple de l'opcode STKWhistle.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKWhistle.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq = p4
kv1 line p5, p3, p6      ;Blowing Frequency Modulation
kv3 = p7      ;Fipple Modulation Frequency

asig STKWhistle cpspch(p4), 1, 4, 20, 11, kv3, 1, 100, 2, kv1, 128, 127
asig = asig*.7      ;too loud
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 .5 9.00 100 30 30
i 1 1 3 9.00 100 0 20
i 1 4.5 . 9.00 1 0 100
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# STKWurley

STKWurley — STKWurley simule par synthèse FM un piano électrique Wurlitzer.

## Description

Opcode du greffon `stkopd`.

*STKWurley* simule par synthèse FM un piano électrique Wurlitzer. Il utilise deux paires FM simples additionnées ensemble, aussi connu comme l'algorithme 5 du TX81Z.

## Syntaxe

`asignal STKWurley ifrequency, iamplitude, [kmod, kv1[, kcross, kv2[, klfo, kv3[, klfodepth, kv4[, kadsr`

## Initialisation

*ifrequency* -- Fréquence de la note jouée, en Hz.

*iamplitude* -- Amplitude de la note jouée (entre 0 et 1).

## Exécution

*kmod* -- contrôleur 2, index 1 du modulateur. *kv1* compris entre 0 et 127.

*kcross* -- contrôleur 4, fondu enchaîné des sorties. *kv2* compris entre 0 et 127.

*klfo* -- contrôleur 11, vitesse de l'oscillateur basse-fréquence. *kv3* compris entre 0 et 127.

*klfodepth* -- contrôleur 1, intensité de l'oscillateur basse-fréquence. *kv4* compris entre 0 et 127.

*kadsr* -- contrôleur 128, cible de l'ADSR 2 et 4. *kv5* compris entre 0 et 127.



### Notes

Le code pour cet opcode vient directement de la classe *Wurley* du Synthesis Toolkit en C++ par Perry R. Cook et Gary P. Scavone. On peut en savoir plus sur les classes STK ici : <https://ccrma.stanford.edu/software/stk/classes.html> [<https://ccrma.stanford.edu/software/stk/classes.html>]

*kc1, kv1, kc2, kv2, kc3, kv3, kc4, kv4, kc5, kv5, kc6, kv6, kc7, kv7, kc8, kv8* -- Jusqu'à 8 paires de contrôle facultatives au taux-k pour les opcodes STK. Chaque paire de contrôle est constituée d'un numéro de contrôleur (*kc*) suivi d'une valeur de contrôleur (*kv*). Le numéro de contrôleur ainsi que la valeur associée sont des variables de taux-k. Cependant, durant l'exécution, les numéros de contrôleur sont habituellement constants tandis que les valeurs correspondantes peuvent changer à tout moment. L'ordre des paires de contrôle est arbitraire mais elles doivent apparaître après *iamplitude*. Il n'est pas non plus nécessaire d'utiliser toutes les paires.

## Exemples

Voici un exemple de l'opcode STKWurley. Il utilise le fichier *STKWurley.csd* [exemples/STKWurley.csd].

**Exemple 968. Exemple de l'opcode STKWurley.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o STKWurley.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifrq = p4
kv1 line p5, p3, p6      ;(FM) Modulator Index One
kv3 = p7

asig STKWurley cpspch(p4), 1, 2,kv1, 4, 10, 11, kv3, 1, 30, 128, 75
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 .5 7.00 75 0 20
i 1 + . 8.00 120 0 20
i 1 + 1 6.00 50 120 20
i 1 + 4 8.00 10 10 127
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*STKRhodey.*

## Crédits

Auteur : Michael Gogins (d'après Perry Cook)  
Irreducible Productions  
New York, NY

Nouveau dans la version 5.11 de Csound.

# strchar

strchar — Retourne le code ASCII d'un caractère dans une chaîne.

## Description

Retourne le code ASCII du caractère de *Sstr* à la position *ipos* (qui vaut zéro par défaut, position du premier caractère), ou zéro si *ipos* est hors limites. *strchar* ne s'exécute que pendant l'initialisation.

## Syntaxe

```
ichr strchar Sstr[, ipos]
```

## Exemples

Voici un exemple de l'opcode strchar. Il utilise le fichier *strchar.csd* [examples/strchar.csd].

### Exemple 969. Exemple de l'opcode strchar.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-ndm0
</CsOptions>
<CsInstruments>
;example by joachim heintz 2013

    opcode ToAscii, S, S    ;returns the ASCII numbers of the input string as string

Sin      xin          ;input string
ilen     strlen      Sin    ;its length
ipos      =           0    ;set counter to zero
Sres      =           ""    ;initialize output string
loop: ;for all characters in input string:
ichr      strchar    Sin, ipos  ;get its ascii code number
Snew      sprintf    "%d ", ichr  ;put this number into a new string
Sres      strcat     Sres, Snew  ;append this to the output string
          loop_lt    ipos, 1, ilen, loop ;see comment for 'loop:'
          xout       Sres      ;return output string

    endop

    instr Characters

printf_i "\nCharacters:\n  given as single strings: %s%s%s%s%s\n", 1, "c", "s", "o", "u", "n", "d"
printf_i "  but can also be given as numbers: %c%c%c%c%c\n", 1, 99, 115, 111, 117, 110, 100
Scsound ToAscii "csound"
printf_i "  in csound, the ASCII code of a character can be accessed with the opcode strchar.%s", 1, "c"
printf_i "  the name 'csound' returns the numbers %s\n\n", 1, Scsound
    endin

</CsInstruments>
<CsScore>

i "Characters" 0 0
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*strchark*

## Crédits

Auteur : Istvan Varga  
2006



# strchark

strchark — Retourne le code ASCII d'un caractère dans une chaîne.

## Description

Retourne le code ASCII du caractère de *Sstr* à la position *kpos* (qui vaut zéro par défaut, position du premier caractère), ou zéro si *kpos* est hors limites. *strchark* s'exécute à l'initialisation ainsi que durant la note.

## Syntaxe

```
kchr strchark Sstr[, kpos]
```

## Voir aussi

*strchar*

## Crédits

Auteur : Istvan Varga  
2006

Nouveau dans la version 5.02

# strcpy

strcpy — Affecte une valeur à une variable chaîne de caractères.

## Description

Affectation à une variable chaîne en copiant la source qui peut être une constante ou une autre variable chaîne. *strcpy* et *=* ne copient la chaîne que pendant l'initialisation.

## Syntaxe

```
Sdst strcpy Ssrc
```

```
Sdst = Ssrc
```

## Exemples

```
Sfoo  strcpy "Hello, world !"
      puts  Sfoo, 1
```

## Voir aussi

*strcpyk*

## Crédits

Auteur : Istvan Varga  
2005

# strcpyk

strcpyk — Affecte une valeur à une variable chaîne de caractères (taux-k).

## Description

Affectation à une variable chaîne en copiant la source qui peut être une constante ou une autre variable chaîne. *strcpyk* fait l'affectation à la fois pendant l'initialisation et pendant l'exécution.

## Syntaxe

Sdst **strcpyk** Ssrc

## Exemples

Voici un exemple de l'opcode strcpyk. Il utilise le fichier *strcpyk.csd* [examples/strcpyk.csd].

### Exemple 970. Exemple de l'opcode strcpyk.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o strcpyk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

seed 0

instr 1
;get one element of the input string whenever the metro
;triggers, and call a subinstrument to play the file

Smember    strget    p4
istrlen    strlen    Smember
kprint     init      0
ktrig      metro     .6

;whenever the trigger gives signal
if ktrig == 1 then
;choose a random element (0, 1 or 2)
kel        random    0, 3.9999
kel        =          int(kel)
;make a copy for leaving Smember intact
Scopy      strcpyk    Smember
;set the initial index for reading substrings
kndx       =          0
```

```

;set counter for searching the element
kcount      =      0
;start looping over the elements in Smember
loop:
kdelim      strindexk Scopy, ":"
;as long as ":" occurs in Scopy, do:
if kdelim > 0 then
;if this is the element to get
if kel == kcount then
;read it as substring
Sfile      strsubk Scopy, kndx, kdelim
kprint = kprint+1
;and jump out
kgoto      call
;if not
else
;cut off this element from Scopy
Scopy      strsubk Scopy, kdelim+1, istrlen
endif
;if no element has been found,go back to loop
;and look for the next element
kcount      =      kcount+1
kgoto      loop
;if there is no delimiter left, the rest is the searched element
else
Sfile      strcpyk Scopy
endif
call:
;print the result, call the subinstrument and play the file
printf      "kel = %d, file = '%s'\n", ktrig+kprint, kel, Sfile
S_call      sprintfk {{i 2 0 1 "%s"}}, Sfile
scoreline S_call, ktrig
endif
endin

instr 2 ;play
Sfile      strget      p4
ilen      filelen      Sfile
p3      =      ilen
asig      soundin      Sfile
outs      asig, asig
endin
</CsInstruments>
<CsScore>

i 1 0 30 "mary.wav:fox.wav:beats.wav:flute.aiff"
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*strcpy*

## Crédits

Auteur : Istvan Varga  
2005

# strcat

strcat — Concaténation de chaînes de caractères.

## Description

Concaténation de deux chaînes et stockage du résultat dans une variable. *strcat* ne s'exécute que pendant l'initialisation. Il est permis qu'un des arguments d'entrée soit le même que la variable de sortie.

## Syntaxe

Sdst **strcat** Ssrc1, Ssrc2

## Exemples

Voici un exemple de l'opcode *strcat*. Il utilise le fichier *strcat.csd* [examples/strcat.csd].

### Exemple 971. Exemple de l'opcode *strcat*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o strcat.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

Sname = "beats"
Sname strcat Sname, ".wav"
asig soundin Sname
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 2
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*strcatk*

## Crédits

Auteur : Istvan Varga  
2005

Nouveau dans la version 5.02

# strcatk

strcatk — Concaténation de chaînes de caractères (taux-k).

## Description

Concaténation de deux chaînes et stockage du résultat dans une variable. *strcatk* s'exécute à la fois pendant l'initialisation et pendant l'exécution. Il est permis qu'un des arguments d'entrée soit le même que la variable de sortie.

## Syntaxe

Sdst **strcatk** Ssrc1, Ssrc2

## Exemples

Voici un exemple de l'opcode strcatk. Il utilise le fichier *strcatk.csd* [exemples/strcatk.csd].

### Exemple 972. Exemple de l'opcode strcatk.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o strcatk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

    sr      = 48000
    ksmpps  = 16
    nchnls  = 2
    odbfs   = 1

; Example by Jonathan Murphy 2007

    instr 1

    S1      = "1"
    S2      = " + 1"
    ktrig    init      0
    kval     init      2
    if (ktrig == 1) then
        S1    strcatk  S1, S2
        kval  = kval + 1
    endif
    String   sprintfk  "%s = %d", S1, kval
    puts     String, kval
    ktrig    metro     1

    endin

</CsInstruments>
```

```
<CsScore>
i1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
1 + 1 = 2
1 + 1 + 1 = 3
1 + 1 + 1 + 1 = 4
1 + 1 + 1 + 1 + 1 = 5
1 + 1 + 1 + 1 + 1 + 1 = 6
1 + 1 + 1 + 1 + 1 + 1 + 1 = 7
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 8
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 9
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 10
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 11
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 12
```

## Voir aussi

*strcat*

## Crédits

Auteur : Istvan Varga  
2005

Nouveau dans la version 5.02



# strcmp

strcmp — Compare des chaînes de caractères.

## Description

Compare des chaînes et retourne -1, 0 ou 1 si la première chaîne est inférieure, égale ou supérieure à la seconde, respectivement. *strcmp* ne compare que pendant l'initialisation.

## Syntaxe

```
ires strcmp S1, S2
```

## Exemples

Voici un exemple de l'opcode strcmp. Il utilise le fichier *strcmp.csd* [examples/strcmp.csd].

### Exemple 973. Exemple de l'opcode strcmp.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o strcmp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
;modified example from Joachim Heintz
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

opcode Triad, iiii, S      ;define UDO
Sname      xin
iMaj      strcmp      "maj", Sname
iMin      strcmp      "min", Sname
iPrim      =          8.00 ;notes in pitch notation
iQuint     =          8.05
    if iMaj == 0 then
iTer      =          8.03
    elseif iMin == 0 then
iTer      =          8.02
    endif
    xout      iPrim, iTer, iQuint
endop

instr 1

Sname strget p4
ia, ib, ic Triad Sname      ;apply UDO
    print ia, ib, ic
asig1 pluck 0.7, cpspch(ia), 220, 0, 1
asig2 pluck 0.7, cpspch(ib), 220, 0, 1
asig3 pluck 0.7, cpspch(ic), 220, 0, 1
asig = (asig1+asig2+asig3)*.5
```

```
    outs asig, asig

endin
</CsInstruments>
<CsScore>
i1 0 3 "maj"
i1 4 3 "min"
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*strcmpk*

## Crédits

Auteur : Istvan Varga  
2005

# strcmpk

strcmp — Compare des chaînes de caractères.

## Description

Compare des chaînes et retourne -1, 0 ou 1 si la première chaîne est inférieure, égale ou supérieure à la seconde, respectivement. *strcmpk* effectue la comparaison à la fois pendant l'initialisation et pendant l'exécution.

## Syntaxe

```
kres strcmpk S1, S2
```

## Voir aussi

*strcmp*

## Crédits

Auteur : Istvan Varga  
2005

# streson

streson — Résonance d'une corde de fréquence fondamentale variable.

## Description

Un signal audio est modifié par un résonateur de type corde avec une fréquence fondamentale variable.

## Syntaxe

ares **streson** asig, kfr, kfdbgain

## Exécution

*asig* -- le signal d'entrée audio.

*kfrq* -- la fréquence fondamentale de la corde.

*kfdbgain* -- gain de rétroaction, entre 0 et 1, de la ligne à retard interne. Une valeur proche de 1 crée une décroissance plus lente et une résonance plus prononcée. Avec de petites valeurs, le signal d'entrée peut ne pas être affecté. Dépend de la fréquence du filtre, les valeurs typiques étant > 0.9. Les valeurs jusqu'à 1 sont aussi utiles.

*streson* fait passer l'entrée *asig* à travers un réseau composé de filtres en peigne, passe-bas et passe-tout, comme celui qui est utilisé dans certaines versions de l'algorithme de Karplus-Strong, créant un effet de résonance d'une corde. La fréquence fondamentale de la « corde » est contrôlée par la variable de taux-*k* *kfr*. On peut utiliser cet opcode pour simuler des résonances sympathiques sur un signal d'entrée.

Voir *Rapports de Fréquence Modale* pour les rapports de fréquence d'instruments réels pouvant être utilisés pour déterminer les valeurs de *kfrq*.

*streson* est une adaptation de l'objet *StringFlt* de la bibliothèque d'objets sonores *SndObj* développée par l'auteur.

## Exemples

Voici en exemple de l'opcode *streson*. Il utilise le fichier *streson.csd* [examples/streson.csd].

### Exemple 974. Exemple de l'opcode *streson*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o streson.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

sr = 44100
ksmps = 32
Odbfs = 1
nchnls = 2

instr 1

asig disk2 "fox.wav", 1, 0, 1

kfr = p4
ifdbgain = 0.90

astr streson asig, kfr, ifdbgain
asig clip astr, 0, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 1 20
i 1 + . >
i 1 + . >
i 1 + . >
i 1 + . >
i 1 + . >
i 1 + . 1000
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Victor Lazzarini  
 Music Department  
 National University of Ireland, Maynooth  
 Maynooth, Co. Kildare  
 1998

Nouveau dans la version 3.494 de Csound

# strfromurl

strfromurl — Donne à une variable chaîne de caractères une valeur lue depuis une URL.

## Description

*strfromurl* donne à une variable chaîne de caractères durant l'initialisation une valeur trouvée en lisant une URL.

## Syntaxe

Sdst **strfromurl** StringURL

## Initialisation

*StringURL* -- chaîne de caractères nommant une URL.

*Sdst* -- variable chaîne de caractères de destination.

## Exemples

Voici un exemple de l'opcode strfromurl. Il utilise le fichier *strfromurl.csd* [examples/strfromurl.csd].

### Exemple 975. Exemple de l'opcode strfromurl.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o strget.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

#include "http://codemist.co.uk/jpff/test.in"

instr 2

Sfile strfromurl "http://codemist.co.uk/jpff/test.in"
prints Sfile
endin
</CsInstruments>
<CsScore>

i 1 0 1
i 2 1 1
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*strset*

## Crédits

Auteur : John ffitch

2013

Les URLs sont disponibles à partir de Csound 6.02 s'il a été compilé avec leur support.

# strget

**strget** — Donne à une variable chaîne de caractères une valeur venant de la table de strset ou d'un p-champ chaîne de caractères.

## Description

*strget* donne à une variable chaîne de caractères pendant l'initialisation une valeur mémorisée de la table de *strset* à l'indice spécifié ou dans un p-champ chaîne de caractères de la partition. S'il n'y a pas de chaîne définie pour cet indice, la variable reçoit une chaîne vide.

## Syntaxe

*Sdst* **strget** *indx*

## Initialisation

*indx* -- indice de *strset* ou p-champ de la partition.

*Sdst* -- variable chaîne de caractères de destination.

## Exemples

Voici un exemple de l'opcode *strget*. Il utilise le fichier *strget.csd* [exemples/strget.csd].

### Exemple 976. Exemple de l'opcode *strget*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o strget.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

strset 1, "fox.wav"
strset 2, "beats.wav"

instr 1

Sfile strget p4
asig soundin Sfile
outs asig, asig

endin
```



```
</CsInstruments>
<CsScore>

i 1 0 2.7 1
i 1 + 2 2
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*strset*

## Crédits

Auteur : Istvan Varga

2005

# strindex

`strindex` — Retourne la position de la première occurrence d'une chaîne de caractères dans une autre chaîne.

## Description

Retourne la position de la première occurrence de *S2* dans *S1*, ou -1 si elle n'est pas trouvée. Si *S2* est vide, 0 est retourné. *strindex* ne s'exécute que pendant l'initialisation.

## Syntaxe

```
ipos strindex S1, S2
```

## Voir aussi

*strindexk*

## Crédits

Auteur : Istvan Varga  
2006

Nouveau dans la version 5.02

# strindexk

strindexk — Retourne la position de la première occurrence d'une chaîne de caractères dans une autre chaîne.

## Description

Retourne la position de la première occurrence de *S2* dans *S1*, ou -1 si elle n'est pas trouvée. Si *S2* est vide, 0 est retourné. *strindex* s'exécute à la fois pendant l'initialisation et pendant l'exécution.

## Syntaxe

kpos **strindexk** S1, S2

## Exemples

Voici un exemple de l'opcode strindexk. Il utilise le fichier *strindexk.csd* [examples/strindexk.csd].

### Exemple 977. Exemple de l'opcode strindexk.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o strindexk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

seed 0

instr 1
;get one element of the input string whenever the metro
;triggers, and call a subinstrument to play the file

Smember   strget      p4
istrlen    strlen     Smember
kprint     init        0
ktrig      metro      .5

;whenever the trigger gives signal
if ktrig == 1 then
;choose a random element (0, 1 or 2)
kel        random     0, 2.9999
kel        =          int(kel)
;make a copy for leaving Smember intact
Scopy      strcpyk    Smember
;set the initial index for reading substrings
```

```

kndx      =      0
;set counter for searching the element
kcount    =      0
;start looping over the elements in Smember
loop:
kdelim     strindex Scopy, ":"
;as long as ":" occurs in Scopy, do:
if kdelim > 0 then
;if this is the element to get
if kel == kcount then
;read it as substring
Sfile      strsubk   Scopy, kndx, kdelim
kprint = kprint+1
;and jump out
kgoto      call
;if not
else
;cut off this element from Scopy
Scopy      strsubk   Scopy, kdelim+1, istrlen
endif
;if no element has been found,go back to loop
;and look for the next element
kcount     =      kcount+1
kgoto      loop
;if there is no delimiter left, the rest is the searched element
else
Sfile      strcpyk   Scopy
endif
call:
;print the result, call the subinstrument and play the file
printf     "kel = %d, file = '%s'\n", ktrig+kprint, kel, Sfile
S_call     sprintfk  {{i 2 0 1 "%s"}}, Sfile
scoreline  S_call, ktrig
endif

endin

instr 2 ;play
Sfile      strget     p4
ilen       filelen    Sfile
p3         =          ilen
asig       soundin    Sfile
           outs       asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 30 "mary.wav:fox.wav:beats.wav"
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*strindex*

## Crédits

Auteur : Istvan Varga  
2006

Nouveau dans la version 5.02

# strlen

strlen — Retourne la longueur d'une chaîne de caractères.

## Description

Retourne la longueur d'une chaîne, ou zéro si elle est vide. *strlen* ne s'exécute que pendant l'initialisation.

## Syntaxe

```
ilen strlen Sstr
```

## Voir aussi

*strlenk*

## Crédits

Auteur : Istvan Varga  
2006

Nouveau dans la version 5.02

# strlen

strlen — Retourne la longueur d'une chaîne de caractères.

## Description

Retourne la longueur d'une chaîne, ou zéro si elle est vide. *strlen* s'exécute à la fois pendant l'initialisation et pendant l'exécution.

## Syntaxe

```
klen strlen Sstr
```

## Voir aussi

*strlen*

## Crédits

Auteur : Istvan Varga  
2006

Nouveau dans la version 5.02

# strlower

strlower — Convertit une chaîne de caractères en minuscules.

## Description

Convertit *Ssrc* en minuscules, et écrit le résultat dans *Sdst*. *strlower* ne s'exécute que pendant l'initialisation.

## Syntaxe

*Sdst* **strlower** *Ssrc*

## Exemples

Voici un exemple de l'opcode *strlower*. Il utilise le fichier *strlower.csd* [exemples/strlower.csd].

### Exemple 978. Exemple de l'opcode *strlower*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n      ;;no sound output
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
</CsOptions>
<CsInstruments>
;example of Joachim Heintz

    opcode FilSuf, S, So
    ;returns the suffix of a filename or path, optional in lower case
    Spath, ilow xin
    ipos strrindex Spath, "." ;look for the rightmost '.'
    Suf strsub Spath, ipos+1 ;extract the substring after "."
    if ilow != 0 then ;if ilow input is not 0 then
    Suf strlower Suf ;convert to lower case
    endif
    xout Suf
    endop

instr suff

ilow = p4
prints "Printing suffix:\n"
Suf FilSuf "/my/dir/my/file.WAV", ilow
puts Suf, 1

endin
</CsInstruments>
<CsScore>
i "suff" 0 1 0 ;do not convert to lower case
i "suff" 3 1 1 ;convert to lower case
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*strlowerk*

## Crédits

Auteur : Istvan Varga  
2006

Nouveau dans la version 5.02



# strlowerk

strlowerk — Convertit une chaîne de caractères en minuscules.

## Description

Convertit *Ssrc* en minuscules, et écrit le résultat dans *Sdst*. *strlowerk* s'exécute à l'initialisation ainsi que durant la note.

## Syntaxe

*Sdst* **strlowerk** *Ssrc*

## Voir aussi

*strlower*

## Crédits

Auteur : Istvan Varga  
2006

Nouveau dans la version 5.02

# strrindex

strrindex — Retourne la position de la dernière occurrence d'une chaîne de caractères dans une autre chaîne.

## Description

Retourne la position de la dernière occurrence de *S2* dans *S1*, ou -1 si elle n'est pas trouvée. Si *S2* est vide, la longueur de *S1* est retournée. *strrindex* ne s'exécute que pendant l'initialisation.

## Syntaxe

ipos **strrindex** S1, S2

## Exemples

Voici un exemple de l'opcode *strrindex*. Il utilise le fichier *strrindex.csd* [examples/strrindex.csd].

### Exemple 979. Exemple de l'opcode *strrindex*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n      ;;no sound output
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
</CsOptions>
<CsInstruments>
;example by Joachim Heintz

    opcode FilNam, S, S
    ;returns the name of a file path
    Spath xin
    ipos   strrindex Spath, "/" ;look for the rightmost '/'
    Snam   strsub    Spath, ipos+1 ;extract the substring
           xout      Snam
    endop

    instr name
        prints      "Printing name:\n"
    Snam   FilNam    "/my/dir/my/file.WAV"
        puts       Snam, 1
    endin
</CsInstruments>
<CsScore>
i "name" 0 0
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*strrindexk*

## Crédits

Auteur : Istvan Varga  
2006

Nouveau dans la version 5.02

# strrindexk

**strrindexk** — Retourne la position de la dernière occurrence d'une chaîne de caractères dans une autre chaîne.

## Description

Retourne la position de la dernière occurrence de *S2* dans *S1*, ou -1 si elle n'est pas trouvée. Si *S2* est vide, la longueur de *S1* est retournée. *strrindexk* s'exécute à la fois pendant l'initialisation et pendant l'exécution.

## Syntaxe

kpos **strrindexk** S1, S2

## Voir aussi

*strrindex*

## Crédits

Auteur : Istvan Varga  
2006

Nouveau dans la version 5.02

# strset

strset — Permet de lier une chaîne de caractères à une valeur numérique.

## Description

Permet de lier une chaîne de caractères à une valeur numérique.

## Syntaxe

```
strset iarg, istring
```

## Initialisation

*iarg* -- la valeur numérique.

*istring* -- la chaîne alphanumérique (entre guillemets).

*strset* (facultatif) permet de lier une chaîne de caractères, telle qu'un nom de fichier, à une valeur numérique. Son usage est facultatif.

## Exemples

L'instruction suivante, utilisée dans l'en-tête de l'orchestre, permet de substituer la valeur 10 partout où l'on a besoin du fichier son *asound.wav*.

```
strset 10, "asound.wav"
```

## Exemples

Voici un exemple de l'opcode *strset*. Il utilise le fichier *strset.csd* [examples/strset.csd].

### Exemple 980. Exemple de l'opcode *strset*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d          ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 1
nchnls = 1

;Example by Andres Cabrera 2008

; \\n is used to denote "new line"
strset 1, "String 1\\n"
```

```
strset 2, "String 2\\n"

instr 1
Str strget p4
prints Str
endin

</CsInstruments>
<CsScore>
;      p4 is used to select string
i 1 0 1 1
i 1 3 1 2
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pset et strget*

# rstrip

rstrip — Enlève des espaces d'une chaîne de caractères.

## Description

Opcodes du greffon emugens.

Enlève des espaces d'une chaîne de caractères. Les espaces peuvent être supprimés à gauche, à droite ou des deux côtés. Fonctionne à l'initialisation.

## Syntaxe

Sout **rstrip** Sin [, Smode]

## Initialisation

**Sin** -- Chaîne en entrée.

**Smode** -- S'il est absent, les espaces sont supprimés des deux côtés. S'il vaut "l", les espaces à gauche sont supprimés. S'il vaut "r", les espaces à droite sont supprimés.

## Exemples

Voici un exemple de l'opcode rstrip. Il utilise le fichier *rstrip.csd* [examples/rstrip.csd].

### Exemple 981. Exemple de l'opcode rstrip.

```
<CsoundSynthesizer>
<CsOptions>
--nosound

</CsOptions>

<CsInstruments>

/* rstrip: strip whitespace from string

Sout rstrip Sin [, Smode]

Args
  Sin - string to strip whitespace from
  Smode - if not given, whitespace is stripped from left and right edges
          "l" - strip whitespace from left edge only
          "r" - strip whitespace from right edge only
*/

instr 1
  Sout = rstrip(" \t\n foo bar \t\n")
  ; Sout = rstrip(" center ", "l")
  ; Sout = rstrip(" center ", "r")

  prints "string: '%s'\n", Sout
  turnoff
endin
```

```
</CsInstruments>

<CsScore>

i1 0 1

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*strcat, strsub*

## Crédits

Par : Eduardo Moguillansky 2020



# strsub

strsub — Extrait une sous-chaîne de caractères.

## Description

Retourne une sous-chaîne d'une chaîne source. *strsub* ne s'exécute que pendant l'initialisation.

## Syntaxe

```
Sdst strsub Ssrc[, istart[, iend]]
```

## Initialisation

*istart* (facultatif, 0 par défaut) -- position du début dans *Ssrc*, comptée à partir de 0. Une valeur négative signifie à partir de la fin de la chaîne.

*iend* (facultatif, 1 par défaut) -- position de la fin dans *Ssrc*, comptée à partir de 0. Une valeur négative signifie à partir de la fin de la chaîne. Si *iend* est inférieure à *istart*, la sortie est inversée.

## Exemples

Voici un exemple de l'opcode *strsub*. Il utilise le fichier *strsub.csd* [examples/strsub.csd].

### Exemple 982. Exemple de l'opcode *strsub*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac            -iadc          ;;-d      RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o strsub.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
; By: Jonathan Murphy 2007

instr 1
  Smember strget p4

  ; Parse Smember
  istrlen  strlen  Smember
  idelimiter strindex Smember, ":"

  S1      strsub Smember, 0, idelimiter ; "String1"
  S2      strsub Smember, idelimiter + 1, istrlen ; "String2"

  printf "First string: %s\nSecond string: %s\n", 1, S1, S2

endin

</CsInstruments>
```

```
<CsScore>  
i 1 0 1 "String1:String2"  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*strsubk*

## Crédits

Auteur : Istvan Varga  
2006

# strsubk

strsubk — Extrait une sous-chaîne de caractères.

## Description

Retourne une sous-chaîne d'une chaîne source. *strsubk* s'exécute à la fois pendant l'initialisation et pendant l'exécution.

## Syntaxe

Sdst **strsubk** Ssrc, kstart, kend

## Exécution

*kstart* -- position du début dans *Ssrc*, comptée à partir de 0. Une valeur négative signifie à partir de la fin de la chaîne.

*kend* position de la fin dans *Ssrc*, comptée à partir de 0. Une valeur négative signifie à partir de la fin de la chaîne. Si *kend* est inférieure à *kstart*, la sortie est inversée.

## Voir aussi

*strsub*

## Crédits

Auteur : Istvan Varga  
2006

# strtod

strtod — Convertit une chaîne de caractères en un nombre flottant (taux-i).

## Description

Convertit une chaîne de caractères en un nombre flottant. Il est également possible de passer un indice de *strset* ou un p-champ chaîne depuis la partition au lieu de l'argument chaîne. Si la chaîne ne peut pas être traduite en nombre flottant ou en nombre entier, il y a une erreur d'initialisation et l'instrument est désactivé.

## Syntaxe

```
ir strtod Sstr
```

```
ir strtod indx
```

## Initialisation

*Sstr* -- Chaîne à convertir.

*indx* -- indice d'une chaîne fixé par *strset*.

## Exécution

*ir* -- Valeur traduite de la chaîne en nombre flottant.

## Crédits

Auteur : Istvan Varga  
2005

# strtodk

strtodk — Convertit une chaîne de caractères en un nombre flottant (taux-k).

## Description

Convertit une chaîne de caractères en un nombre flottant au taux-i ou au taux-k. Il est également possible de passer un indice de *strset* ou un p-champ chaîne depuis la partition au lieu de l'argument chaîne. Si la chaîne ne peut pas être traduite en nombre flottant ou en nombre entier, il y a une erreur d'initialisation ou d'exécution et l'instrument est désactivé.



### Note

Si une variable indice de taux-k est utilisée, elle doit être valide dès l'initialisation.

## Syntaxe

```
kr strtodk Sstr
```

```
kr strtodk kndx
```

## Exécution

*kr* -- Valeur traduite de la chaîne en nombre flottant.

*Sstr* -- Chaîne à convertir.

*indx* -- indice d'une chaîne fixé par *strset*.

## Crédits

Auteur : Istvan Varga  
2005

# strtol

strtol — Convertit une chaîne de caractères en un nombre entier (taux-i).

## Description

Convertit une chaîne de caractères en un nombre entier. Il est également possible de passer un indice de *strset* ou un p-champ chaîne depuis la partition au lieu de l'argument chaîne. Si la chaîne ne peut pas être traduite en nombre entier, il y a une erreur d'initialisation et l'instrument est désactivé.

## Syntaxe

```
ir strtol Sstr
```

```
ir strtol indx
```

## Initialisation

*Sstr* -- Chaîne à convertir.

*indx* -- indice d'une chaîne fixé par *strset*.

*strtol* peut traduire des nombres en format décimal, octal (préfixés par 0) et hexadécimal (avec le préfixe 0x).

## Exécution

*ir* -- Valeur traduite de la chaîne en nombre entier.

## Crédits

Auteur : Istvan Varga  
2005

# strtolk

strtolk — Convertit une chaîne de caractères en un nombre entier (taux-k).

## Description

Convertit une chaîne de caractères en un nombre entier au taux-i ou au taux-k. Il est également possible de passer un indice de *strset* ou un p-champ chaîne depuis la partition au lieu de l'argument chaîne. Si la chaîne ne peut pas être traduite en nombre entier, il y a une erreur d'initialisation ou d'exécution et l'instrument est désactivé.



### Note

Si une variable indice de taux-k est utilisée, elle doit être valide dès l'initialisation.

## Syntaxe

```
kr strtolk Sstr
```

```
kr strtolk kndx
```

*strtolk* peut traduire des nombres en format décimal, octal (préfixés par 0) et hexadécimal (avec le préfixe 0x).

## Exécution

*kr* -- Valeur traduite de la chaîne en nombre entier.

*Sstr* -- Chaîne à convertir.

*indx* -- indice d'une chaîne fixé par *strset*.

## Crédits

Auteur : Istvan Varga  
2005

# strupper

strupper — Convertit une chaîne de caractères en majuscules.

## Description

Convertit *Ssrc* en majuscules, et écrit le résultat dans *Sdst*. *strupper* ne s'exécute que pendant l'initialisation.

## Syntaxe

```
Sdst strupper Ssrc
```

## Voir aussi

*strupperk*

## Crédits

Auteur : Istvan Varga  
2006

Nouveau dans la version 5.02



# strupperk

strupperk — Convertit une chaîne de caractères en majuscules.

## Description

Convertit *Ssrc* en majuscules, et écrit le résultat dans *Sdst*. *strupper* s'exécute à l'initialisation ainsi que durant la note.

## Syntaxe

*Sdst* **strupperk** *Ssrc*

## Voir aussi

*strupper*

## Crédits

Auteur : Istvan Varga  
2006

Nouveau dans la version 5.02

# subinstr

subinstr — Crée et lance une instance d'un instrument numéroté.

## Description

Crée une instance d'un autre instrument qui est utilisé comme s'il était un opcode.

## Syntaxe

```
a1, [...] [, a8] subinstr instrnum [, p4] [, p5] [...]  
a1, [...] [, a8] subinstr "insname" [, p4] [, p5] [...]
```

## Initialisation

*instrnum* -- Numéro de l'instrument à appeler.

« *insname* » -- Une chaîne de caractères (entre guillemets) représentant un instrument nommé.

## Exécution

*a1*, ..., *a8* -- La sortie audio de l'instrument appelé. Elle est générée au moyen des opcodes de *Sortie de Signal*.

*p4*, *p5*, ... -- Valeurs d'entrée supplémentaires qui sont affectées aux p-champs de l'instrument appelé, en commençant par *p4*.

Les valeurs *p2* et *p3* de l'instrument appelé seront identiques aux valeurs de l'instrument hôte. Alors que l'instrument hôte peut *contrôler sa propre durée*, toute tentative similaire à l'intérieur de l'instrument appelé n'aura très probablement aucun effet.

## Voir aussi

*event*, *schedule*, *subinstrinit*

## Exemples

Voici un exemple de l'opcode subinstr. Il utilise le fichier *subinstr.csd* [examples/subinstr.csd].

### Exemple 983. Exemple de l'opcode subinstr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
; Audio out  Audio in  
-odac      -iadc      ;;RT audio I/O  
; For Non-realtime output leave only the line below:  
; -o subinstr.wav -W ;;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - Creates a basic tone.
instr 1
; Print the value of p4, should be equal to
; Instrument #2's iamp field.
print p4

; Print the value of p5, should be equal to
; Instrument #2's ipitch field.
print p5

; Create a tone.
asig oscils p4, p5, 0

out asig
endin

; Instrument #2 - Demonstrates the subinstr opcode.
instr 2
iamp = 20000
ipitch = 440

; Use Instrument #1 to create a basic sine-wave tone.
; Its p4 parameter will be set using the iamp variable.
; Its p5 parameter will be set using the ipitch variable.
abasic subinstr 1, iamp, ipitch

; Output the basic tone that we have created.
out abasic
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #2 for one second.
i 2 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Voici un exemple de l'opcode `subinstr` utilisant un instrument nommé. Il utilise le fichier `subinstr_named.csd` [examples/subinstr\_named.csd].

### Exemple 984. Exemple de l'opcode `subinstr` utilisant un instrument nommé.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O

```

```

; For Non-realtime ouput leave only the line below:
; -o subinstr_named.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument "basic_tone" - Creates a basic tone.
instr basic_tone
; Print the value of p4, should be equal to
; Instrument #2's iamp field.
print p4

; Print the value of p5, should be equal to
; Instrument #2's ipitch field.
print p5

; Create a tone.
asig oscils p4, p5, 0

out asig
endin

; Instrument #1 - Demonstrates the subinstr opcode.
instr 1
iamp = 20000
ipitch = 440

; Use the "basic_tone" named instrument to create a
; basic sine-wave tone.
; Its p4 parameter will be set using the iamp variable.
; Its p5 parameter will be set using the ipitch variable.
abasic subinstr "basic_tone", iamp, ipitch

; Output the basic tone that we have created.
out abasic
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsSoundSynthesizer>

```

## Crédits

Nouveau dans la version 4.21

# subinstrinit

subinstrinit — Crée et lance une instance d'un instrument numéroté à l'initialisation.

## Description

Identique à *subinstr*, mais seulement à l'initialisation et sans arguments de sortie.

## Syntaxe

```
subinstrinit instrnum [, p4] [, p5] [...]  
subinstrinit "insname" [, p4] [, p5] [...]
```

## Initialisation

*instrnum* -- Numéro de l'instrument à appeler.

« *insname* » -- Une chaîne de caractères (entre guillemets) représentant un instrument nommé.

*p4*, *p5*, ... -- Valeurs d'entrée supplémentaires qui sont affectées aux p-champs de l'instrument appelé, en commençant par *p4*.

Les valeurs *p2* et *p3* de l'instrument appelé seront indentiques aux valeurs de l'instrument hôte. Alors que l'instrument hôte peut *contrôler sa propre durée*, toute tentative similaire à l'intérieur de l'instrument appelé n'aura très probablement aucun effet.

## Voir aussi

*event*, *schedule*, *subinstr*

## Crédits

Nouveau dans la version 4.23

# sum

sum — Somme de n'importe quel nombre de signaux de taux-a ou des éléments d'un tableau.

## Description

Somme de n'importe quel nombre de signaux de taux-a ou des éléments d'un tableau.

## Syntaxe

```
ares sum asig1 [, asig2] [, asig3] [...]  
kres sum karr  
ires sum iarr
```

## Exécution

*asig1, asig2, ...* -- signaux de taux-a à additionner (à mélanger).

*karr, iarr* -- tableaux unidimensionnels.

## Exemples

voici un exemple de l'opcode sum. Il utilise le fichier *sum.csd* [examples/sum.csd].

### Exemple 985. Exemple de l'opcode sum.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac    ;;;realtime audio out  
;-iadc    ;;;uncomment -iadc if realtime audio input is needed too  
; For Non-realtime ouput leave only the line below:  
; -o sum.wav -W ;;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 32  
nchnls = 2  
0dbfs = 1  
  
gisine ftgen 0, 0, 2^10, 10, 1  
  
instr 1  
  
a1 oscili 1, 10.0, gisine ;combine 3 sinusses  
a2 oscili 1, 1.0, gisine ;at different rates  
a3 oscili 1, 3.0, gisine  
ares sum a1, a2, a3 ;sum them  
  
ares = ares*100 ;scale result and
```

```
asig poscil .5, ares+110, gisine ;add to frequency
outs asig, asig

endin
</CsInstruments>
<CsScore>

i1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Avril 1999

Nouveau dans la version 3.54 de Csound

Ajouts pour les tableaux dans la version 6.09

# sumarray

sumarray — Retourne la somme des éléments dans un tableau.

## Description

L'opcode *sumarray* retourne la somme de tous les éléments d'un tableau de taux-k.

## Syntaxe

```
ksum sumarray karray
```

## Exécution

*ksum* -- variable pour le résultat.

*karray* -- tableau à lire.

## Exemples

Voici un exemple de l'opcode sumarray. Il utilise le fichier *sumarray.csd* [examples/sumarray.csd].

### Exemple 986. Exemple de l'opcode sumarray.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n
</CsOptions>
<CsInstruments>
;example by joachim heintz

      seed          0

instr 1
;create an array with 10 elements
kArr[] init        10
;fill in random numbers and print them out
kIndx  =            0
      until kIndx == 10 do
kNum   random      0, 10
kArr[kIndx] =      kNum
      printf       "kArr[%d] = %10f\n", kIndx+1, kIndx, kNum
kIndx  +=          1
      od
;calculate sum of all values and print it out
kSum   sumarray   kArr
      printf       "Sum of all values in kArr = %f\n", kIndx+1, kSum
      turnoff
endin
</CsInstruments>
<CsScore>
i1 0 0.1
```



```
e  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*minarray, sumarray, scalearray,*

## Crédits

Auteur : John ffitch  
Octobre 2011

Nouveau dans la version 5.14 de Csound.

Révisé dans la version 6.00 de Csound pour fonctionner avec les tableau multidimensionnels.

# svfilter

**svfilter** — Un filtre à résonance du second ordre, avec sortie passe-bas, passe-haut et passe-bande simultanées.

## Description

Implémentation d'un filtre à résonance du second ordre, avec sortie passe-bas, passe-haut et passe-bande simultanées.

## Syntaxe

```
alow, ahigh, aband svfilter asig, kcf, kq [, iscl][, iskip]
```

## Initialisation

*iscl* -- facteur de pondération codé semblable à celui de *reson*. Une valeur différente de zéro signifie que la crête du facteur de réponse est 1, c-à-d. toutes les fréquences autres que *kcf* sont atténuées selon la courbe de réponse (normalisée). Une valeur de zéro signifie aucune pondération du signal, laissant cette tâche à un ajustement ultérieur (voir *balance*). La valeur par défaut est 0.

*iskip* (facultatif, 0 par défaut) -- disposition initiale de l'espace de données internes. Comme le filtrage comprend une boucle de rétroaction, l'état initial de l'espace de stockage est significatif. La valeur zéro efface l'espace ; une valeur non nulle maintient l'information antérieure. La valeur par défaut est 0.

## Exécution

*svfilter* est un filtre à variable d'état du second ordre, avec contrôle au taux-k de la fréquence de coupure et de Q. Lorsque Q augmente, un pic de résonance se forme autour de la fréquence de coupure. *svfilter* a des sorties passe-bas, passe-haut et passe-bande simultanées ; en mélangeant les sorties, on peut générer des réponses en fréquence variées. Le filtre à variable d'état, ou filtre "multimodal", se rencontrait fréquemment dans les premiers synthétiseurs analogiques, en raison de la grande variété de sonorités produites par l'interaction entre la fréquence de coupure, la résonance et les rapports de mélange en sortie. *svfilter* est bien adapté à la simulation de sonorités "analogiques", ainsi que pour d'autres applications nécessitant des filtres à résonance.

*asig* -- signal d'entrée à filtrer.

*kcf* -- fréquence de coupure ou de résonance du filtre, mesurée en Hz.

*kq* -- Q du filtre, défini (pour les filtres passe-bande) comme le rapport (largeur de bande)/(fréquence de coupure). *kq* doit être compris entre 1 et 500. Lorsque *kq* augmente, la résonance du filtre augmente, ce qui correspond à une augmentation de la magnitude et de la "raideur" du pic de résonance. Si l'on utilise *svfilter* sans pondération du signal (*iscl* absent ou nul), le volume du pic de résonance augmente en même temps que Q. Pour de grandes valeurs de Q, il est recommandé de donner à *iscl* une valeur différente de zéro, ou bien d'utiliser une fonction de mise à l'échelle externe telle que *balance*.

*svfilter* est basé sur un algorithme du livre de Hal Chamberlin, *Musical Applications of Microprocessors* (Hayden Books, 1985).

## Exemples

Voici un exemple de l'opcode `svfilter`. Il utilise le fichier `svfilter.csd` [examples/svfilter.csd].

### Exemple 987. Exemple de l'opcode `svfilter`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o svfilter.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Orchestra file for resonant filter sweep of a sawtooth-like waveform.
; The separate outputs of the filter are scaled by values from the score,
; and are mixed together.
sr = 44100
kr = 2205
ksmps = 20
nchnls = 1

instr 1

    idur      = p3
    ifreq     = p4
    iamp      = p5
    ilowamp   = p6           ; determines amount of lowpass output in signal
    ihighamp  = p7           ; determines amount of highpass output in signal
    ibandamp  = p8           ; determines amount of bandpass output in signal
    iq        = p9           ; value of q

    iharms    = (sr*.4) / ifreq

    asig      gbuzz 1, ifreq, iharms, 1, .9, 1           ; Sawtooth-like waveform
    kfreq     linseg 1, idur * 0.5, 4000, idur * 0.5, 1 ; Envelope to control filter cutoff

    alow, ahigh, aband svfilter asig, kfreq, iq

    aout1     = alow * ilowamp
    aout2     = ahigh * ihighamp
    aout3     = aband * ibandamp
    asum      = aout1 + aout2 + aout3
    kenv      linseg 0, .1, iamp, idur -.2, iamp, .1, 0 ; Simple amplitude envelope
    out       asum * kenv

endin

</CsInstruments>
<CsScore>

f1 0 8192 9 1 1 .25

i1 0 5 100 1000 1 0 0 5 ; lowpass sweep
i1 5 5 200 1000 1 0 0 30 ; lowpass sweep, octave higher, higher q
i1 10 5 100 1000 0 1 0 5 ; highpass sweep
i1 15 5 200 1000 0 1 0 30 ; highpass sweep, octave higher, higher q
i1 20 5 100 1000 0 0 1 5 ; bandpass sweep
```

```
i1 25 5 200 1000 0 0 1 30 ; bandpass sweep, octave higher, higher q
i1 30 5 200 2000 .4 .6 0 ; notch sweep - notch formed by combining highpass and lowpass outputs
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Sean Costello  
Seattle, Washington  
1999

Nouveau dans la version 3.55 de Csound.

# syncgrain

syncgrain — Synthèse granulaire synchrone.

## Description

*syncgrain* implémente la synthèse granulaire synchrone. La source de son pour les grains est obtenue par la lecture d'une table de fonction contenant les échantillons de la forme d'onde source. Pour les sources de son échantillonné, on utilise *GEN01*. *syncgrain* acceptera des tables allouées en différé.

Le générateur de grain exerce un contrôle total sur la fréquence (grains/sec), l'amplitude générale, la hauteur du grain (incrément d'échantillonnage) et la taille du grain (en sec), comme paramètres constants ou variant dans le temps (signaux). Le taux du pointeur de grain est un paramètre supplémentaire qui contrôle à quelle position le générateur commencera à lire les échantillons dans la table pour chaque grain successif. Il est mesuré en fraction de la taille du grain ; s'il vaut 1 (la valeur par défaut) chaque grain successif est lu à partir de l'endroit où le grain précédent s'est terminé. S'il vaut 0,5 le grain suivant commencera à mi-chemin entre la position de début et la position de fin du grain précédent, etc. S'il vaut 0 le générateur lira toujours à partir de la même position dans la table (quelque soit l'endroit où le pointeur se trouvait juste avant). Avec une valeur négative le pointeur évoluera en décrémentant sa position. Ce contrôle apporte plus de flexibilité dans la création de modifications de l'échelle temporelle lors de la resynthèse.

*syncgrain* générera n'importe quel nombre de flux parallèles de grains (en fonction de la densité/fréquence de grains), borné supérieurement par la valeur de *iolaps* (100 par défaut). Le nombre de flux (grains se chevauchant) est déterminé par  $\text{taille\_du\_grain} \times \text{fréquence\_du\_grain}$ . Plus il y aura de chevauchements de grains, plus il y aura de calculs et il se peut que la synthèse ne s'effectue pas en temps réel (cela dépend de la puissance du processeur).

*syncgrain* peut simuler une synthèse formantique à la FOF, si l'on utilise une forme d'enveloppe de grain adéquate et une sinusoïde comme forme d'onde du grain. Dans ce cas, on pourra utiliser des tailles de grain d'environ 0,04 sec. La fréquence centrale du formant est déterminée par la hauteur du grain. Comme l'incrément est en échantillons, si l'on veut utiliser une fréquence en Hz, cette valeur doit être multipliée par  $\text{taille\_de\_la\_table} / \text{sr}$ . La fréquence du grain détermine le fondamental.

*syncgrain* utilise des indices en virgule flottante, ce qui fait qu'il n'est pas affecté par des tables de grande taille. Cet opcode est basé sur la class *SynGrain* de la bibliothèque *SndObj*.

## Syntaxe

```
asig syncgrain kamp, kfreq, kpitch, kgrsize, kprate, ifun1, \
      ifun2, iolaps
```

## Initialisation

*ifun1* -- table de fonction du signal source. Des tables avec allocation différée sont acceptées (voir *GEN01*), mais l'opcode attend une source mono.

*ifun2* -- table de fonction de l'enveloppe du grain.

*iolaps* -- nombre maximum de chevauchements,  $\max(kfreq) \times \max(kgrsize)$ . Une grande valeur d'estimation ne devrait pas affecter l'exécution, mais le dépassement de cette valeur aura probablement des conséquences désastreuses.

## Exécution

*kamp* -- pondération de l'amplitude.

*kgreq* -- fréquence de génération des grains, ou densité, en grains/sec.

*kpitch* -- transposition de hauteur des grains (1 = hauteur normale, < 1 plus bas, > 1 plus haut ; négatif, lecture à l'envers).

*kgrsize* -- taille du grain en secondes.

*kprate* -- vitesse du pointeur de lecture, en grains. Une valeur de 1 avancera le pointeur de lecture d'un grain dans la table source. Des valeurs supérieures provoqueront une compression temporelle et des valeurs inférieures une expansion temporelle du signal source. Avec des valeurs négatives, le pointeur progressera à l'envers et zéro l'immobilisera.

## Exemples

Voici un exemple de l'opcode *syncgrain*. Il utilise le fichier *syncgrain.csd* [exemples/syncgrain.csd].

### Exemple 988. Exemple de l'opcode *syncgrain*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o syncgrain.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1

iolaps = 2
igrsize = 0.04
ifreq = iolaps/igrsize
ips = 1/iolaps

istr = .3 /* timescale */
ipitch = p4 /* pitchscale */

asig syncgrain 1, ifreq, ipitch, igrsize, ips*istr, 1, 2, iolaps
outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 0 1 "fox.wav" 0 0 0 ;deferred table
f2 0 8192 20 2 1

i1 0 5 1
```

```
i1 + 5 4  
i1 + 5 .8  
e  
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur: Victor Lazzarini  
Janvier 2005

Nouveau greffon dans la version 5

Janvier 2005.

# syncloop

syncloop — Synthèse granulaire synchrone.

## Description

*syncloop* est une variation sur *syncgrain*, qui implémente la synthèse granulaire synchrone. *syncloop* ajoute des points de début et de fin de boucle et une position de départ facultative. Le début et la fin de boucle contrôlent les positions de démarrage des grains, si bien que les grains réalisés peuvent s'étendre au-delà des points de la boucle (si les points de la boucle ne sont pas aux extrémités de la table), ce qui permet des transitions fluides. Pour plus d'information sur le procédé de synthèse granulaire, voir la page du manuel sur *syncgrain*.

## Syntaxe

```
asig syncloop kamp, kfreq, kpitch, kgrsize, kprate, klstart, \  
      klend, ifun1, ifun2, iolaps[,istart, iskip]
```

## Initialisation

*ifun1* -- table de fonction du signal source. Des tables avec allocation différée sont acceptées (voir *GEN01*), mais l'opcode attend une source mono.

*ifun2* -- table de fonction de l'enveloppe du grain.

*iolaps* -- nombre maximum de chevauchements,  $\max(kfreq) \cdot \max(kgrsize)$ . Une grande valeur d'estimation ne devrait pas affecter l'exécution, mais le dépassement de cette valeur aura probablement des conséquences désastreuses.

*istart* -- point de départ de la synthèse en secs (0 par défaut).

*iskip* -- s'il vaut 1, l'initialisation de l'opcode est ignorée, pour les notes liées, l'exécution continuant depuis la position à l'intérieur de la boucle où la note précédente s'est terminée. La valeur par défaut de 0 signifie que l'initialisation n'est pas ignorée.

## Exécution

*kamp* -- pondération de l'amplitude.

*kfreq* -- fréquence de génération des grains, ou densité, en grains/sec.

*kpitch* -- transposition de hauteur des grains (1 = hauteur normale, < 1 plus bas, > 1 plus haut ; négatif, lecture à l'envers).

*kgrsize* -- taille du grain en secondes.

*kprate* -- vitesse du pointeur de lecture, en grains. Une valeur de 1 avancera le pointeur de lecture d'un grain dans la table source. Des valeurs supérieures provoqueront une compression temporelle et des valeurs inférieures une expansion temporelle du signal source. Avec des valeurs négatives, le pointeur progressera à l'envers et zéro l'immobilisera.

*klstart* -- début de la boucle en secs.



*klend* -- fin de la boucle en secs.

## Exemples

Voici un exemple de l'opcode *syncloop*. Il utilise le fichier *syncloop.csd* [examples/syncloop.csd].

### Exemple 989. Exemple de l'opcode *syncloop*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o syncloop.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
Odbfs = 1
nchnls = 2

instr 1

iolaps = 2
igrsize = 0.01
ifreq = iolaps/igrsize
ips = 1/iolaps

istr = p4 /* timescale */
ipitch = 1 /* pitchscale */

asig syncloop 1, ifreq, ipitch, igrsize, ips*istr, .3, .75, 1, 2, iolaps
outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 0 1 "beats.wav" 0 0 0
f2 0 8192 20 2 1

i1 0 6 .5
i1 7 6 .15
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
Janvier 2005

Nouveau greffon dans la version 5

Janvier 2005.

# syncphasor

syncphasor — Produit une valeur de phase mobile normalisée avec entrée et sortie de synchronisation.

## Description

Produit une valeur de phase mobile entre zéro et un et une impulsion supplémentaire en sortie ("sync out") chaque fois que sa valeur de phase traverse le zéro ou est remise à zéro. La phase peut être réinitialisée à tout instant par une impulsion sur le paramètre "sync in".

## Syntaxe

*aphase*, *asyncout* **syncphasor** *xcps*, *asyncin*, [, *iphs*]

## Initialisation

*iphs* (facultatif) -- phase initiale, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation de la phase sera ignorée. La valeur par défaut est zéro.

## Exécution

*aphase* -- la valeur de phase en sortie ; toujours entre 0 et 1.

*asyncout* -- la sortie de synchronisation prend la valeur 1.0 durant un échantillon chaque fois que la valeur de phase traverse le zéro ou que l'entrée de synchronisation a une valeur non nulle. Elle vaut zéro aux autres moments.

*asyncin* -- l'entrée de synchronisation provoque la remise à zéro de la phase chaque fois que *asyncin* est non nul.

*xcps* -- fréquence du phaseur en Hertz. Si *xcps* est négatif, la phase sera décrémentée de 1 à 0 au lieu d'être incrémentée.

Une phase interne est augmentée successivement selon la fréquence de *xcps* pour produire une valeur de phase mobile, normalisée pour se trouver dans l'intervalle  $0 \leq \text{phs} < 1$ . Lorsqu'elle est utilisée comme indice dans une *table*, cette phase (multipliée par la longueur de la table de fonction) permettra de l'utiliser comme un oscillateur.

La phase de *syncphasor* peut être synchronisée à un autre phaseur (ou à un autre signal) au moyen du paramètre *asyncin*. Chaque fois que *asyncin* prend une valeur non nulle, la valeur de *aphase* est remise à zéro. *syncphasor* sort aussi son propre signal de "synchro" qui consiste en une impulsion d'un échantillon chaque fois que sa phase traverse le zéro ou est réinitialisée. On peut ainsi facilement mettre en série plusieurs opcodes *syncphasor* pour créer un effet d'oscillateur "hard sync".

## Exemples

Voici un exemple de l'opcode *syncphasor*. Il utilise le fichier *syncphasor.csd* [examples/syncphasor.csd].

### Exemple 990. Exemple de l'opcode syncphasor.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o abs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

instr 1
; Use two syncphasors - one is the "master",
; the other the "slave"

; master's frequency determines pitch
imastercps =      cpspch(p4)
imaxamp     =      10000

; the slave's frequency affects the timbre
kslavecps   line   imastercps, p3, imastercps * 3

; the master "oscillator"
; the master has no sync input
anosync     init   0.0
am, async   syncphasor imastercps, anosync

; the slave "oscillator"
aout, as    syncphasor kslavecps, async

adeclick    linseg  0.0, 0.05, 1.0, p3 - 0.1, 1.0, 0.05, 0.0

; Output the slave's phase value which is a rising
; sawtooth wave. This produces aliasing, but hey, this
; this is just an example ;)

          out      aout * adeclick * imaxamp

endin

</CsInstruments>
<CsScore>

i1 0 1      7.00
i1 + 0.5    7.02
i1 + .      7.05
i1 + .      7.07
i1 + .      7.09
i1 + 2      7.06

e

</CsScore>
</CsoundSynthesizer>

```

Voici un autre exemple de l'opcode syncphasor. Il utilise le fichier *syncphasor-CZresonance.csd* [examples/syncphasor-CZresonance.csd].

### Exemple 991. Un autre exemple de l'opcode syncphasor.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o syncphasor-CZresonance.wav -W ;; for file output any platform

```

```

</CsOptions>
<CsInstruments>
; by Anthony Kozar. February 2008
; http://www.anthonykozar.net/

; Imitation of the Casio CZ-series synthesizer's "Resonance" waveforms
; using a synced phasor to read a sinusoid table. The jumps at the sync
; points are smoothed by multiplying with a windowing function controlled
; by the master phasor.

; Based on information from the Wikipedia article on phase distortion:
; http://en.wikipedia.org/wiki/Phase_distortion_synthesis

; Sawtooth Resonance waveform. Smoothing function is just the inverted
; master phasor.

; The Wikipedia article shows an inverted cosine as the stored waveform,
; which implies that it must be unipolar for the smoothing to work.
; I have substituted a sine wave in the first phrase to keep the output
; bipolar. The second phrase demonstrates the much "rezzier" sound of the
; bipolar cosine due to discontinuities.

instr 1
    ifreq      =          cpspch(p4)
    initReson  =          p5
    itable     =          p6
    imaxamp    =          10000
    anosync    init     0.0

    kslavecps  line      ifreq * initReson, p3, ifreq
    amaster, async syncphasor ifreq, anosync ; pair of phasors
    aslave, async2 syncphasor kslavecps, async ; slave synced to master
    aosc       tablei     aslave, itable, 1 ; use slave phasor to read a (co)sine table
    aout        =          aosc * (1.0 - amaster) ; inverted master smoothes jumps
    adeclick    linseg     0.0, 0.05, 1.0, p3 - 0.1, 1.0, 0.05, 0.0

                                out          aout * adeclick * imaxamp
endinstr

; Triangle or Trapezoidal Resonance waveform. Uses a second table to change
; the shape of the smoothing function. (This is my best guess so far as to
; how these worked). The cosine table works fine with the triangular smoothing
; but we once again need to use a sine table with the trapezoidal smoothing.

; (It might be interesting to be able to vary the "width" of the trapezoid.
; This could be done with the pdhalf opcode).

instr 2
    ifreq      =          cpspch(p4)
    initReson  =          p5
    itable     =          p6
    ismoothtbl =          p7
    imaxamp    =          10000
    anosync    init     0.0

    kslavecps  line      ifreq * initReson, p3, ifreq
    amaster, async syncphasor ifreq, anosync ; pair of phasors
    aslave, async2 syncphasor kslavecps, async ; slave synced to master
    aosc       tablei     aslave, itable, 1 ; use slave phasor to read a (co)sine table
    asmooth    tablei     amaster, ismoothtbl, 1 ; use master phasor to read smoothing table
    aout        =          aosc * asmooth
    adeclick    linseg     0.0, 0.05, 1.0, p3 - 0.1, 1.0, 0.05, 0.0

                                out          aout * adeclick * imaxamp
endinstr

</CsInstruments>

```

```

<CsScore>
f1 0 16385 10 1
f3 0 16385 9 1 1 270 ; inverted cosine
f5 0 4097 7 0.0 2048 1.0 2049 0.0 ; unipolar triangle
f6 0 4097 7 1.0 2048 1.0 2049 0.0 ; "trapezoid"

; Sawtooth resonance with a sine table
i1 0 1 7.00 5.0 1
i. + 0.5 7.02 4.0
i. + . 7.05 3.0
i. + . 7.07 2.0
i. + . 7.09 1.0
i. + 2 7.06 12.0
f0 6
s

; Sawtooth resonance with a cosine table
i1 0 1 7.00 5.0 3
i. + 0.5 7.02 4.0
i. + . 7.05 3.0
i. + . 7.07 2.0
i. + . 7.09 1.0
i. + 2 7.06 12.0
f0 6
s

; Triangle resonance with a cosine table
i2 0 1 7.00 5.0 3 5
i. + 0.5 7.02 4.0
i. + . 7.05 3.0
i. + . 7.07 2.0
i. + . 7.09 1.0
i. + 2 7.06 12.0
f0 6
s

; Trapezoidal resonance with a sine table
i2 0 1 7.00 5.0 1 6
i. + 0.5 7.02 4.0
i. + . 7.05 3.0
i. + . 7.07 2.0
i. + . 7.09 1.0
i. + 2 7.06 12.0

e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*phasor*.

Et les opcodes d'Accès aux Table comme : *table*, *tablei*, *table3* et *tab*.

## Crédits

Adapté d'après l'opcode *phasor* par Anthony Kozar  
Janvier 2008

Nouveau dans la version 5.08 de Csound

# system

system — Appelle un programme externe via le système.

## Description

Opcodes du greffon `system_call`.

**system** et **system\_i** appellent toute commande externe comprise par le système d'exploitation, comme le fait la fonction `system()` du langage C. **system\_i** ne s'exécute que pendant l'initialisation, tandis que **system** s'exécute pendant l'initialisation aussi bien que durant l'exécution.

## Syntaxe

```
ires system_i itrig, Scmd, [inowait]
kres system ktrig, Scmd, [knowait]
```

## Initialisation

*Scmd* -- chaîne de commande.

*itrig* -- s'il est supérieur à zéro, l'opcode exécute la commande demandée ; sinon c'est une opération nulle.

## Exécution

*ktrig* -- s'il est supérieur à zéro et différent de la valeur qu'il avait lors du précédent cycle de contrôle, l'opcode exécute la commande demandée. La valeur précédente initiale est prise à zéro.

*inowait*, *knowait* -- s'il est différent de zéro, la commande est exécutée en arrière-plan et l'on attend pas son résultat (0 par défaut).

*ires*, *kres* -- le code retourné par la commande en mode attente et si la commande est exécutée. Retourne zéro dans les autres cas.

Un seul opcode **system** peut exécuter plus d'une commande si l'on entoure la chaîne avec des accolades doubles `{ { }`.



### Note

Cet opcode dépendant fortement du système, il faut l'utiliser avec beaucoup de précautions (ou ne pas l'utiliser) si l'on désire rester neutre par rapport à la plateforme.

## Exemples

Voici un exemple de l'opcode `system_i`. Il utilise le fichier `system.csd` [examples/system.csd].

### Exemple 992. Exemple de l'opcode `system_i`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          ; -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o system.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
; Waits for command to execute before continuing
ires system_i 1, {{      ps
                        date
                        cd ~/Desktop
                        pwd
                        ls -l
                        whois csounds.com
                        }}
print ires
turnoff
endin

instr 2
; Runs command in a separate thread
ires system_i 1, {{      ps
                        date
                        cd ~/Desktop
                        pwd
                        ls -l
                        whois csounds.com
                        }}, 1

print ires
turnoff
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for thirty seconds.
i 1 0 1
i 2 5 1
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur: John fitch  
2007

Nouveau dans la version 5.06

# tab

tab — Opcodes de table rapides.

## Description

Opcodes de table rapides. Plus rapides que *table* et que *tablew* parce qu'ils fonctionnent sans indexation cyclique et sans limite et qu'ils ne testent pas la validité des index. Ils ont été implémentés pour fournir un accès rapide aux tableaux.

## Syntaxe

```
ir tab_i indx, ifn[, ixmode]
kr tab kndx, ifn[, ixmode]
ar tab xndx, ifn[, ixmode]
tabw_i isig, indx, ifn [,ixmode]
tabw ksig, kndx, ifn [,ixmode]
tabw asig, andx, ifn [,ixmode]
```

## Initialisation

*ifn* -- numéro de la table.

*ixmode* -- zéro par défaut. S'il est nul, l'intervalle de variation de *xndx* est la longueur de la table ; s'il est non nul, *xndx* varie entre 0 et 1.

*isig* -- valeur d'entrée à écrire.

*indx* -- index de la table.

## Exécution

*asig*, *ksig* -- signal d'entrée à écrire.

*andx*, *kndx* -- index de la table.

Les opcodes *tab* et *tabw* sont semblables à *table* et à *tablew*, mais ils sont plus rapides et ils utilisent une indexation avec arrondi.

## Exemples

Voici un exemple de l'opcode *tab*. Il utilise le fichier *tab.csd* [examples/tab.csd].

### Exemple 993. Exemple de l'opcode *tab*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.



```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tab.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gifn1 ftgen 1, 0, 0, 1, "flute.aiff", 0, 0, 0 ;deferred-size table

instr 1

atab init 0
isize tableng 1      ;length of table?
print isize
andx phasor 1 / (isize / sr)
asig tab andx, 1, 1    ;has a 0 to 1 range
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 2.3
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Ecrit par Gabriel Maldonado.

# tabifd

tabifd — Distribution de fréquence instantanée, analyse d'amplitude et de phase.

## Description

L'opcode *tabifd* prend en entrée une table de fonction et fait une analyse de fréquence instantanée, amplitude et phase, au moyen de la TFCT et *tabifd* (distribution de fréquence instantanée) comme décrit dans Lazzarini et al, "Time-stretching using the Instantaneous Frequency Distribution and Partial Tracking", Proc.of ICMC05, Barcelone. Il génère deux signaux de flot PV, l'un contenant les amplitudes et les fréquences (une sortie similaire à celle de *pvsanal*) et l'autre contenant les amplitudes et les phases non reliées.

## Syntaxe

```
ffr,fphs tabifd ktimpt, kamp, kpitch, ifftsize, ihopsize, iwintype,ifn
```

## Initialisation

*ifftsize* -- taille d'analyse de la TFR, doit être une puissance de deux et un multiple entier de la taille du saut.

*ihopsize* -- taille du saut en échantillons.

*iwintype* -- type de la fenêtre (0 : Hamming, 1 : Hanning).

*ifn* -- table de fonction source.

## Exécution

*ffr* -- flot PV en sortie au format AMP\_FREQ.

*fphs* -- flot PV en sortie au format AMP\_PHASE.

*ktimpt* -- point (en secondes) où commence la lecture dans la table (s'il est inférieur à zéro ou supérieur à la longueur de la table, il est ramené dans les limites par une opération de modulo).

*kamp* -- mise à l'échelle de l'amplitude.

*kpitch* -- mise à l'échelle de la hauteur (transposition).

## Exemples

Voici un exemple de l'opcode *tabifd*. Il utilise le fichier *tabifd.csd* [examples/tabifd.csd].

### Exemple 994. Exemple de l'opcode *tabifd*.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>
```

```
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tabifd.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr=44100
ksmps=1
nchnls=1
opcode TrackPlay, a, kkiiii
ktime,kthr,tsiz,ihsiz,ifcos,ifn xin
idel = tsiz-ihsiz*(tsiz/(2*ihsiz)-1)
ffr,fphs tabifd ktime,10000,1, tsiz, ihsiz, 1, ifn
ftrk partials ffr, fphs,kthr, 1, 1, 500
aout tradsyn ftrk, 2,1, 500, ifcos
xout aout
endop
instr 1
p3 = ftlen(2)/sr
ktime line 0,p3,p3
ares TrackPlay ktime, 0.003,2048,256,1,2
outs ares
endin

</CsInstruments>
<CsScore>
f1 0 16384 9 1 1 90
f2 0 0 1 "fox.wav" 0 0 1
i1 0 1
</CsScore>
</CsoundSynthesizer>
```

L'exemple ci-dessus montre l'analyse *tabifd* alimentant une resynthèse additive avec transposition de hauteur par poursuite de partiel et phase cubique.

## Crédits

Auteur : Victor Lazzarini

Août 2015

Nouveau greffon dans la version 6

Août 2015

# table

table — Accède aux valeurs d'une table par indexation directe.

## Description

Accède aux valeurs d'une table par indexation directe.

## Syntaxe

```
ares table andx, ifn [, ixmode] [, ixoff] [, iwrap]
ires table indx, ifn [, ixmode] [, ixoff] [, iwrap]
kres table kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

## Initialisation

*ifn* -- numéro de la table de fonction.

*ixmode* (facultatif) -- type de l'index. La valeur par défaut est 0.

- 0 = index brut
- 1 = normalisé (de 0 à 1)

*ixoff* (facultatif) -- décalage de l'index. Pour une table dont l'origine est au centre, utiliser *taille\_table/2* (brut) ou 0.5 (normalisé). La valeur par défaut est 0.

*iwrap* (facultatif) -- indicateur d'indexation cyclique. La valeur par défaut est 0.

- 0 = indexation normale (index < 0 traité comme index=0 ; index > *taille\_table* ramené à index=*taille\_table*)
- 1 = indexation cyclique.

## Exécution

*table* effectue une consultation de table avec des index variant au taux d'initialisation, de contrôle ou audio. Ces index peuvent être des nombres bruts (0, 1, 2, ..., *taille* - 1) ou des valeurs normalisées (0 à 1). Les index sont d'abord modifiés par la valeur de décalage puis leur appartenance à un intervalle valable est testée avant la consultation de la table (voir *iwrap*). Si l'index peut prendre la valeur maximale ou si l'on utilise l'interpolation, la table doit avoir un point de garde. Une *table* indexée par un phaseur périodique (voir *phasor*) simulera un oscillateur.

## Exemples

Voici un exemple de l'opcode *table*. Il utilise le fichier *table.csd* [examples/table.csd].

### Exemple 995. Exemple de l'opcode *table*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac        -iadc       -d        ;;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o table.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Vary our index linearly from 0 to 1.
kndx line 0, p3, 1

; Read Table #1 with our index.
ifn = 1
ixmode = 1
kfreq table kndx, ifn, ixmode

; Generate a sine waveform, use our table values
; to vary its frequency.
a1 oscil 20000, kfreq, 2
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a line from 200 to 2,000.
f 1 0 1025 -7 200 1024 2000
; Table #2, a sine wave.
f 2 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*tablei, table3, oscil1, oscilli, osciln*

## Crédits

Exemple écrit par Kevin Conder.

# table3

table3 — Accède aux valeurs d'une table par indexation directe avec interpolation cubique.

## Description

Accède aux valeurs d'une table par indexation directe avec interpolation cubique.

## Syntaxe

```
ares table3 andx, ifn [, ixmode] [, ixoff] [, iwrap]
ires table3 indx, ifn [, ixmode] [, ixoff] [, iwrap]
kres table3 kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

## Initialisation

*ifn* -- numéro de la table de fonction.

*ixmode* (facultatif) -- type de l'index. La valeur par défaut est 0.

- 0 = index brut
- 1 = normalisé (de 0 à 1)

*ixoff* (facultatif) -- décalage de l'index. Pour une table dont l'origine est au centre, utiliser *taille\_table/2* (brut) ou 0.5 (normalisé). La valeur par défaut est 0.

*iwrap* (facultatif) -- indicateur d'indexation cyclique. La valeur par défaut est 0.

- 0 = indexation normale (index < 0 traité comme index=0 ; index > *taille\_table* ramené à index=*taille\_table*)
- 1 = indexation cyclique.

## Exécution

*table3* est semblable à *tablei*, sauf qu'il utilise l'interpolation cubique. (Nouveau dans la version 3.50 de Csound).



### Avertissement

La lecture de tables contenant une information stéréo ou multicanaux causera probablement du bruit non désiré car l'opcode interpole entre des positions successives de la table sans tenir compte de l'origine de son contenu. Habituellement seul le contenu d'un canal est attendu. Il faut plutôt considérer l'utilisation de *loscilx*.

## Voir aussi

*table*, *tablei*, *oscil1*, *oscil1i*, *osciln*, *loscilx*

# tablecopy

tablecopy — Opcode de copie de table simple et rapide.

## Description

Opcode de copie de table simple et rapide.

## Syntaxe

```
tablecopy kdft, ksft
```

## Exécution

*kdft* -- Table de fonction destination.

*ksft* -- Numéro de la table de fonction source.

*tablecopy* -- Opcode de copie de table simple et rapide. Il prend la longueur de la table destination et lit à partir du début de la table source. Pour aller vite, il ne teste pas la longueur de la source - il copie quoiqu'il arrive - en mode « cyclique ». Ainsi, la table source peut-être lue plusieurs fois. Avec une table source de longueur 1, toutes les positions de la tables destination recevront son unique valeur.

*tablecopy* ne peut pas lire ou écrire le point de garde. Pour le lire, il faut utiliser *table*, avec *ndx* = la longueur de la table. De même, il faut utiliser une écriture de table pour l'écrire.

Pour écrire le point de garde avec la valeur de la position 0, utiliser *tablegpw*.

Cet opcode sert principalement à changer les tables de fonction rapidement dans une situation de temps réel.

## Exemples

Voici un exemple de l'opcode tablecopy. Il utilise le fichier *tablecopy.csd* [examples/tablecopy.csd].

### Exemple 996. Exemple de l'opcode tablecopy.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tablecopy.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 32
nchnls  = 2
0dbfs   = 1
;after an example from Jonathan Murphy
```

```

gilen = 7
gilist  ftgen 1, 0, gilen + 1, -2, 2, 3, 4, 5, 6, 7 ;only 6 elements, so 1 is doubled
gitmp   ftgen 2, 0, gilen + 1, -2, 0   ;empty table
gkmax   init gilen

seed 0           ;each time different

instr 1

ktrig metro 3           ;trigger values
krnd  random 0, gkmax

if (ktrig == 1) then
  kval  table krnd, gilist
        tablew 0, krnd, gilist
  kread = 0
  kwrite = 0
start:
  knew  table kread, gilist
if (knew != 0) then
  tablew knew, kwrite, gitmp
  kwrite = kwrite + 1
endif
  kread = kread + 1
if (kread <= gilen) kgoto start
  tablecopy gilist, gitmp ;fill with zeroes
  gkmax = gkmax - 1
endif

printk2 kval

if (gkmax < 0) then
  event "i", 2, 0, 1/kr ;when ready, then stop
endif

asig vco2 .5, 40*kval ;sound generation
outs asig, asig

endin

instr 2

exitnow

endin
</CsInstruments>
<CsScore>
i1 0 5
e
</CsScore>
</CsoundSynthesizer>

```

La sortie contiendra des lignes comme celles-ci :

```

i1 5.00000
i1 3.00000
i1 2.00000
i1 4.00000
i1 7.00000
i1 6.00000
i1 7.00000

```

## Voir aussi

*tablegpw, tablemix, tableicopy, tableigpw, tableimix*



## Crédits

Auteur : Robin Whittle

Australie

Mai 1997

Nouveau dans la version 3.47

# tablefilter

tablefilter — Filtre une table source et écrit le résultat dans une table de destination.

## Description

On peut utiliser cet opcode pour filtrer les valeurs de tables de fonction selon certains algorithmes. La sortie filtrée est écrite dans une table de destination et le nombre d'éléments qui ont passé le filtre est retourné.

## Syntaxe

knumpassed **tablefilter** kouttable, kintatble, kmode, kparam

## Exécution

*knumpassed* -- le nombre d'éléments qui ont passé le filtre.

*kouttable* -- le numéro de la table contenant les valeurs qui sont passées.

*kintatble* -- le numéro de la table à filtrer.

*kmode* -- mode du filtre :

- 1 -- teste le poids des dénominateurs des fractions dans la table source. Ne passent que les valeurs de la source qui ont un poids moins lourd que celui du seuil.
- 2 -- teste le poids des dénominateurs des fractions dans la table source. Ne passent que les valeurs de la source qui ont un poids supérieur ou égal à celui du seuil.

*kparam* -- entier, paramètre de seuil pour le filtre. Cela signifie que les dénominateurs dont le poids est plus lourd que celui de ce seuil ne passent pas à travers le filtre. Le poids d'un entier est calculé au moyen de la fonction de Clarence Barlow d'indigestibilité d'un nombre. Selon cette fonction, les grands nombres premiers contribuent à un accroissement du poids de tout nombre entier naturel qu'ils divisent. L'ordre des 16 premiers entiers selon leur indigestibilité est : 1, 2, 4, 3, 8, 6, 16, 12, 9, 5, 10, 15, 7, 14.

## Exemples

Voici un exemple de l'opcode tablefilter. Il utilise le fichier *tablefilter.csd* [examples/tablefilter.csd].

### Exemple 997. Exemple de l'opcode tablefilter.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>

</CsOptions>
<CsInstruments>

sr=44100
ksmps=10
nchnls=1
```

```

gifarn init 8 ; initialise integer for Farey Sequence F_8
gires fareyleni gifarn ; calculate length of F_8, returns 23
; the table length won't be a power of 2
; (The length of a Farey Sequence with n > 1 is always odd)
gilen init gires * -1

gifarey ftgen 200, 0, gilen, "farey", gifarn, 0

; initialize destination table with 0s
gifiltered ftgen 0, 0, gilen, 21, 1, 0

; initialize second destination table with 0s
gifiltered2 ftgen 0, 0, gilen, 21, 1, 0

; table filtering opcode: dest. source, mode, threshold
ginumpassed tablefilteri gifiltered, gifarey, 1, 6
; the threshold parameter indicates that denominators whose weights are heavier
; than 6 are not passing through the filter. The weight is calculated using
; Clarence Barlow's function of indigestibility of a number. According to this function,
; higher prime numbers contribute to an increased weight of any natural integer they divide.
; ginumpassed is the number of elements from the source table 'gifarey'
; that have passed the test and which have been copied to the destination table 'gifiltered'

; apply a different filter:
ginumpassed2 tablefilteri gifiltered2, gifarey, 2, 5
; In mode=2 we again test the digestibility of the denominators of the
; fractions in the source table.
; The difference to mode=1 is that we now let pass only vaules from the
; source that are as heavy as the threshold or greater.

instr 4
  kndx init 0 ; read out elements of now filtered F_8 sequentially and print to file
  if (kndx < ginumpassed) then
    kelem tab kndx, gifiltered
    fprintks "fareyfilter_lp.txt", "%2.6f\\n", kelem
    kndx = kndx+1
  endif
endin

instr 5
  kndx init 0 ; read out elements and print to file
  if (kndx < ginumpassed2) then
    kelem tab kndx, gifiltered2
    fprintks "fareyfilter_hp.txt", "%2.6f\\n", kelem
    kndx = kndx+1
  endif
endin

</CsInstruments>
<CsScore>

i4      0      1
i5      0      1
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Georg Boenn  
 Université de Glamorgan, UK

New dans la version 5.13 de Csound.

# tablefilteri

tablefilteri — Filtre une table source et écrit le résultat dans une table de destination.

## Description

On peut utiliser cet opcode pour filtrer les valeurs de tables de fonction selon certains algorithmes. La sortie filtrée est écrite dans une table de destination et le nombre d'éléments qui ont passé le filtre est retourné.

## Syntaxe

```
inunpassed tablefilteri iouttable, iintatble, imode, iparam
```

## Initialisation

*inunpassed* -- le nombre d'éléments qui ont passé le filtre.

*iouttable* -- le numéro de la table contenant les valeurs qui sont passées.

*iintatble* -- le numéro de la table à filtrer.

*imode* -- mode du filtre :

- 1 -- teste le poids des dénominateurs des fractions dans la table source. Ne passent que les valeurs de la source qui ont un poids moins lourd que celui du seuil.
- 2 -- teste le poids des dénominateurs des fractions dans la table source. Ne passent que les valeurs de la source qui ont un poids supérieur ou égal à celui du seuil.

*iparam* -- entier, paramètre de seuil pour le filtre. Cela signifie que les dénominateurs dont le poids est plus lourd que celui de ce seuil ne passent pas à travers le filtre. Le poids d'un entier est calculé au moyen de la fonction de Clarence Barlow d'indigestibilité d'un nombre. Selon cette fonction, les grands nombres premiers contribuent à un accroissement du poids de tout nombre entier naturel qu'ils divisent. L'ordre des 16 premiers entiers selon leur indigestibilité est : 1, 2, 4, 3, 8, 6, 16, 12, 9, 5, 10, 15, 7, 14.

## Exemples

Voici un exemple de l'opcode tablefilteri. Il utilise le fichier *tablefilter.csd* [examples/tablefilter.csd].

### Exemple 998. Exemple de l'opcode tablefilteri.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>

</CsOptions>
<CsInstruments>

sr=44100
ksmps=10
nchnls=1
```

```

gifarn init 8 ; initialise integer for Farey Sequence F_8
gires fareyleni gifarn ; calculate length of F_8, returns 23
; the table length won't be a power of 2
; (The length of a Farey Sequence with n > 1 is always odd)
gilen init gires * -1

gifarey ftgen 200, 0, gilen, "farey", gifarn, 0

; initialize destination table with 0s
gifiltered ftgen 0, 0, gilen, 21, 1, 0

; initialize second destination table with 0s
gifiltered2 ftgen 0, 0, gilen, 21, 1, 0

; table filtering opcode: dest. source, mode, threshold
ginumpassed tablefilteri gifiltered, gifarey, 1, 6
; the threshold parameter indicates that denominators whose weights are heavier
; than 6 are not passing through the filter. The weight is calculated using
; Clarence Barlow's function of indigestibility of a number. According to this function,
; higher prime numbers contribute to an increased weight of any natural integer they divide.
; ginumpassed is the number of elements from the source table 'gifarey'
; that have passed the test and which have been copied to the destination table 'gifiltered'

; apply a different filter:
ginumpassed2 tablefilteri gifiltered2, gifarey, 2, 5
; In mode=2 we again test the digestibility of the denominators of the
; fractions in the source table.
; The difference to mode=1 is that we now let pass only vaules from the
; source that are as heavy as the threshold or greater.

instr 4
  kndx init 0 ; read out elements of now filtered F_8 sequentially and print to file
  if (kndx < ginumpassed) then
    kelem tab kndx, gifiltered
    fprintks "fareyfilter_lp.txt", "%2.6f\\n", kelem
    kndx = kndx+1
  endif
endin

instr 5
  kndx init 0 ; read out elements and print to file
  if (kndx < ginumpassed2) then
    kelem tab kndx, gifiltered2
    fprintks "fareyfilter_hp.txt", "%2.6f\\n", kelem
    kndx = kndx+1
  endif
endin

</CsInstruments>
<CsScore>

i4 0 1
i5 0 1
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Georg Boenn  
Université de Glamorgan, UK

New dans la version 5.13 de Csound.

# tablegpw

tablegpw — Ecrit le point de garde d'une table.

## Description

Ecrit le point de garde d'une table.

## Syntaxe

```
tablegpw kfn
```

## Exécution

*kfn* -- Numéro de la table.

*tablegpw* -- Pour écrire le point de garde d'une table, avec la valeur de la position 0. Ne fait rien si la table n'existe pas.

Peut être utile après avoir manipulé une table avec *tablemix* ou *tablecopy*.

## Voir aussi

*tablecopy*, *tablemix*, *tableicopy*, *tableigpw*, *tableimix*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.47

# tablei

tablei — Accède aux valeurs d'une table par indexation directe avec interpolation linéaire.

## Description

Accède aux valeurs d'une table par indexation directe avec interpolation linéaire.

## Syntaxe

```
ares tablei andx, ifn [, ixmode] [, ixoff] [, iwrap]
ires tablei indx, ifn [, ixmode] [, ixoff] [, iwrap]
kres tablei kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

## Initialisation

*ifn* -- numéro de la table de fonction. *tablei* nécessite un point de garde.

*ixmode* (facultatif) -- type de l'index. La valeur par défaut est 0.

- 0 = index brut
- 1 = normalisé (de 0 à 1)

*ixoff* (facultatif) -- décalage de l'index. Pour une table dont l'origine est au centre, utiliser *taille\_table/2* (brut) ou 0.5 (normalisé). La valeur par défaut est 0.

*iwrap* (facultatif) -- indicateur d'indexation cyclique. La valeur par défaut est 0.

- 0 = indexation normale (index < 0 traité comme index=0 ; index > *taille\_table* ramené à index=*taille\_table*)
- 1 = indexation cyclique.

## Exécution

*tablei* est une unité avec interpolation dans laquelle la partie fractionnaire de l'index est utilisée pour interpoler entre les entrées adjacentes de la table. La régularité apportée par l'interpolation se paie par une légère augmentation du temps d'exécution (voir aussi *oscili*, etc.), mais sinon les unités avec ou sans interpolation sont interchangeables. Noter que lorsque *tablei* utilise un index périodique dont la valeur modulo *n* est inférieure à la puissance de 2, longueur de la table, l'interpolation nécessite qu'il existe une (*n* + 1)ème valeur dans la table qui est une copie de la première valeur (voir l'*instruction f* de la partition).



### Avertissement

La lecture de tables contenant une information stéréo ou multicanaux causera probablement du bruit non désiré car l'opcode interpole entre des positions successives de la table sans tenir compte de l'origine de son contenu. Habituellement seul le contenu d'un canal est attendu. Il faut plutôt considérer l'utilisation de *loscilx*.

## Exemples

Voici un exemple de l'opcode `tablei`. Il utilise le fichier `tablei.csd` [examples/tablei.csd].

### Exemple 999. Exemple de l'opcode `tablei`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tablei.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

seed 0 ;generate new values every time the instr is played

instr 1

ifn = p4
isize = p5
ithresh = 0.5

itemp ftgen ifn, 0, isize, 21, 2

iwrite_value = 0
i_index = 0

loop_start:
  iread_value tablei i_index, ifn

  if iread_value > ithresh then
    iwrite_value = 1
  else
    iwrite_value = -1
  endif
  tableiw iwrite_value, i_index, ifn
  loop_lt i_index, 1, isize, loop_start
  turnoff

endin

instr 2

ifn = p4
isize = ftlen(ifn)
prints "Index\tValue\n"

i_index = 0
loop_start:
  ivalue tablei i_index, ifn
  prints "%d:\t%f\n", i_index, ivalue

  loop_lt i_index, 1, isize, loop_start ;read table 1 with our index
```



```
aout oscili .5, 100, ifn ;use table to play the polypulse
outs aout, aout

endin
</CsInstruments>
<CsScore>
i 1 0 1 100 16
i 2 0 2 100
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*table, table3, oscil1, oscil1i, osciln*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/issue12/genInstruments.html> [<http://www.csoundjournal.com/issue12/genInstruments.html>], écrit par Jacob Joaquin.

# tablecopy

tablecopy — Opcode de copie de table simple et rapide.

## Description

Opcode de copie de table simple et rapide.

## Syntaxe

```
tablecopy idft, isft
```

## Initialisation

*idft* -- Table de fonction destination.

*isft* -- Numéro de la table de fonction source.

## Exécution

*tablecopy* -- Opcode de copie de table simple et rapide. Il prend la longueur de la table destination et lit à partir du début de la table source. Pour aller vite, il ne teste pas la longueur de la source - il copie quoiqu'il arrive - en mode « cyclique ». Ainsi, la table source peut-être lue plusieurs fois. Avec une table source de longueur 1, toutes les positions de la tables destination recevront son unique valeur.

*tablecopy* ne peut pas lire ou écrire le point de garde. Pour le lire, il faut utiliser *table*, avec *ndx* = la longueur de la table. De même, il faut utiliser une écriture de table pour l'écrire.

Pour écrire le point de garde avec la valeur de la position 0, utiliser *tablegpw*.

Cet opcode sert principalement à changer les tables de fonction rapidement dans une situation de temps réel.

## Voir aussi

*tablecopy*, *tablegpw*, *tablemix*, *tableigpw*, *tableimix*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.47

# tableigpw

tableigpw — Ecrit le point de garde d'une table.

## Description

Ecrit le point de garde d'une table.

## Syntaxe

```
tableigpw ifn
```

## Initialisation

*ifn* -- Numéro de la table.

## Exécution

*tableigpw* -- Pour écrire le point de garde d'une table, avec la valeur de la position 0. Ne fait rien si la table n'existe pas.

Peut être utile après avoir manipulé une table avec *tablemix* ou *tablecopy*.

## Voir aussi

*tablecopy*, *tablegpw*, *tablemix*, *tableicopy*, *tableimix*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.47

# tableikt

tableikt — Permet de contrôler au taux-k les numéros de table.

## Description

Contrôle des numéros de table au taux-k. La lecture dans la table se fait avec interpolation linéaire.

L'opcode standard *tablei* de Csound, bien que produisant un résultat au taux-k ou au taux-a, ne peut utiliser qu'une variable de taux-i pour choisir le numéro de la table. *tableikt* accepte un contrôle au taux-k aussi bien qu'au taux-i. Pour le reste, il est semblable à l'opcode original.

## Syntaxe

```
ares tableikt xndx, kfn [, ixmode] [, ixoff] [, iwrap]
```

```
kres tableikt kndx, kfn [, ixmode] [, ixoff] [, iwrap]
```

## Initialisation

*ixmode* -- s'il vaut 0, *xndx* et *ixoff* couvrent toute la longueur de la table. S'il est différent de zéro, *xndx* et *ixoff* varient de 0 à 1. La valeur par défaut est 0.

*ixoff* -- s'il vaut 0, l'indice résultant est directement contrôlé par *xndx*, démarrant au début de la table. S'il est différent de zéro, l'indexation démarre à l'intérieur de la table. Sa valeur doit être positive et inférieure à la longueur de la table (*ixmode* = 0) ou inférieure à 1 (*ixmode* différent de 0). La valeur par défaut est 0.

*iwrap* -- si *iwrap* = 0, *mode Limite* : lorsque l'indice résultant est inférieur à 0, l'indice final vaut 0. Un indice résultant dépassant la longueur de la table donne un indice final égal à la longueur de la table : les indices résultants trop grands se limitent à l'index supérieur de la table. Si *iwrap* est différent de 0, *mode Cyclique* : l'indice résultant est replié modulo la longueur de la table de façon à ce que tous les indices résultants tombent dans la table. Par exemple, dans une table de longueur 8, *xndx* = 5 et *ixoff* = 6 donnent un indice résultant de 11, qui se replie en un indice final de 3. La valeur par défaut est 0.

## Exécution

*kndx* -- Indice dans la table, un nombre positif compris entre 0 et la longueur de la table (*ixmode* = 0) ou entre 0 et 1 (*ixmode* différent de 0).

*xndx* -- varie sur la longueur de la table (*ixmode* = 0) ou dans l'intervalle allant de 0 à 1 (*ixmode* différent de 0).

*kfn* -- Numéro de table. Doit être  $\geq 1$ . Les valeurs flottantes sont arrondies à un entier. Si un numéro de table n'indique pas une table valide, ou si la table n'a pas encore été chargée (*GEN01*) une erreur se produit et l'instrument est désactivé.



### Attention avec les numéros de table au taux-k

Au taux-k, si un numéro de table  $< 1$  est donné, ou si le numéro de table indique une table inexistante ou une table de longueur nulle (devant être chargée à partir d'un fichier ultérieurement), une erreur se produit et l'instrument est désactivé. *kfn* doit être initialisé au taux

approprié en utilisant *init*. Si l'on essaie de charger une valeur de taux-i dans *kfn*, il y aura une erreur.

## Exemples

Voici un exemple de l'opcode *tableikt*. Il utilise le fichier *tableikt.csd* [examples/tableikt.csd].

### Exemple 1000. Exemple de l'opcode *tableikt*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tableikt.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ilowfn = p4      ;lowest ftable wave
ihighfn = p5     ;highest ftable wave

kswpenv line 1, p3, 0      ;sweep envelope, calculate current table pair and interpolation amount
inumtables = ihighfn - ilowfn ;1 less than number of tables
kfn1 = int(kswpenv*inumtables) + ilowfn
printks "play table no: %d\n", 1, kfn1
kfn2 = kfn1 + 1
kinterp = frac(kswpenv*inumtables)
ixmode = 1      ;read tables with phasor
aphase phasor 40
asig tableikt aphase, kfn1, ixmode ;normalized index
if kswpenv == 1.0 kgoto skipfn2 ;if kfn1 is last table, there is no kfn2
asig2 tableikt aphase, kfn2, ixmode
skipfn2:
amix ntrpol asig, asig2, kinterp ;interpolate between tables and output
outs amix*.5, amix*.5

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1
f 2 0 16384 10 1 .5
f 3 0 16384 1 "fox.wav" 0 0 0 ;a sample
f 4 0 16384 10 1 .5 .3 .25 .2 .16 .14 .125 .111 ;sawtooth
f 5 0 16384 10 1 .4 .3 .25 .2
f 6 0 16384 10 1 .3 .3 .25 .2 .16
f 7 0 16384 10 1 1 1 1 .7 .5 .3 .1 ;pulse
f 8 0 16384 1 "beats.wav" 0 0 0 ;a sample

i 1 0 10 1 8
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie comprendra des lignes comme celles-ci :

```
play table no: 8
play table no: 7
play table no: 6
.....
play table no: 2
play table no: 1
```

## Voir aussi

*tablekt*

## Crédits

Auteur: Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.47

# tableimix

tableimix — Mélange deux tables.

## Description

Mélange deux tables.

## Syntaxe

```
tableimix idft, idoff, ilen, islft, isloff, islg, is2ft, is2off, is2g
```

## Initialisation

*idft* -- Table de fonction destination.

*idoff* -- Décalage de l'origine de l'écriture. Peut être négatif.

*ilen* -- Nombre d'opérations d'écriture à réaliser. Une valeur négative signifie écrire avec des indices descendants.

*islft, is2ft* -- Tables de fonction sources. Peuvent être identique à la table destination, si l'on fait attention au sens d'écriture lors de la copie des données.

*isloff, is2off* -- Décalages de l'origine de la lecture dans les tables sources.

*islg, is2g* -- Gains à appliquer lors de la lecture dans les tables source. Les résultats sont additionnés et la somme est écrite dans la table destination.

## Exécution

*tableimix* -- Cet opcode mélange deux tables, avec des gains séparés dans une table destination. L'écriture se fait sur *ilen* positions, habituellement en avançant dans la table si *ilen* est positif. S'il est négatif, l'écriture et la lecture se font avec des indices décroissants dans les tables. Cet option bi-directionnelle permet de déplacer facilement le contenu d'une table en lisant et en écrivant dans celle-ci avec un décalage différent.

Si *ilen* vaut 0, il n'y a pas d'écriture. Noter que la valeur entière interne de *ilen* est obtenue de la fonction *floor()* du C ANSI qui retourne l'entier négatif directement inférieur. Ainsi avec une valeur fractionnaire négative de *ilen* de -2.3 on aura une longueur interne de 3, et la copie commencera à partir des positions décalées et se fera sur deux positions vers la gauche.

L'indice résultant pour la lecture et l'écriture dans les tables est calculé à partir du décalage de l'origine pour chaque table auquel est additionnée la valeur de l'index, qui commence à 0 et augmente ou diminue d'un pas unité tout au long du mixage.

Ces indices résultants peuvent devenir très grands, car il n'y a aucune restriction pour le décalage ou *ilen*. Cependant l'indice résultant pour chaque table subit un ET logique avec un masque de longueur (tel que 0000 0111 pour une table de longueur 8) pour former l'indice final qui sera utilisé pour la lecture ou l'écriture. Ainsi il ne peut y avoir aucune lecture ou écriture en dehors des tables. C'est la même chose que le mode « wrap » (cyclique) dans la lecture et l'écriture de table. Ces opcodes ne lisent pas ou n'écrivent pas le point de garde. Si une table a été réécrite par l'un de ceux-ci et si elle a un point de garde sensé contenir la même valeur que la position 0, il faut ensuite appeler *tableigpw*.

Les indices et les décalages sont exprimés en pas de table - ils ne sont pas normalisés entre 0 et 1. Ainsi pour une table de longueur 256, *ilen* doit être fixé à 256 si toute la table doit être lue ou écrite.

Il n'est pas nécessaire que les tables soient de même longueur - le parcours cyclique se fait individuellement pour chaque table.

## Exemples

Voici un exemple de l'opcode `tableimix`. Il utilise le fichier `tableimix.csd` [examples/tableimix.csd].

### Exemple 1001. Exemple de l'opcode `tableimix`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tableimix.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisine ftgen 1, 0, 256, 10, 1, 0, 0, .4 ;sinoid
gisaw  ftgen 2, 0, 1024, 7, 0, 256, 1 ;saw
gimix  ftgen 100, 0, 256, 7, 0, 256, 1 ;used to mix

instr 1

    tableimix 100, 0, 256, 1, 0, 1, 2, 0, .5
    asig poscil .5, 110, gimix ;mix table 1 & 2
    outs asig, asig

endin
</CsInstruments>
<CsScore>

i1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*tablecopy, tablegpw, tablemix, tableicopy, tableigpw*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.47



# tablekt

tablekt — Permet de contrôler au taux-k les numéros de table.

## Description

Contrôle des numéros de table au taux-k.

L'opcode standard *table* de Csound, bien que produisant un résultat au taux-k ou au taux-a, ne peut utiliser qu'une variable de taux-i pour choisir le numéro de la table. *tablekt* accepte un contrôle au taux-k aussi bien qu'au taux-i. Pour le reste, il est semblable à l'opcode original.

## Syntaxe

```
ares tablekt xndx, kfn [, ixmode] [, ixoff] [, iwrap]
kres tablekt kndx, kfn [, ixmode] [, ixoff] [, iwrap]
```

## Initialisation

*ixmode* -- s'il vaut 0, *xndx* et *ixoff* couvrent toute la longueur de la table. S'il est différent de zéro, *xndx* et *ixoff* varient de 0 à 1. La valeur par défaut est 0.

*ixoff* -- s'il vaut 0, l'indice résultant est directement contrôlé par *xndx*, démarrant au début de la table. S'il est différent de zéro, l'indexation démarre à l'intérieur de la table. Sa valeur doit être positive et inférieure à la longueur de la table (*ixmode* = 0) ou inférieure à 1 (*ixmode* différent de 0). La valeur par défaut est 0.

*iwrap* -- si *iwrap* = 0, *mode Limite* : lorsque l'indice résultant est inférieur à 0, l'indice final vaut 0. Un indice résultant dépassant la longueur de la table donne un indice final égal à la longueur de la table : les indices résultants trop grands se limitent à l'index supérieur de la table. Si *iwrap* est différent de 0, *mode Cyclique* : l'indice résultant est replié modulo la longueur de la table de façon à ce que tous les indices résultants tombent dans la table. Par exemple, dans une table de longueur 8, *xndx* = 5 et *ixoff* = 6 donnent un indice résultant de 11, qui se replie en un indice final de 3. La valeur par défaut est 0.

## Exécution

*kndx* -- Indice dans la table, un nombre positif compris entre 0 et la longueur de la table (*ixmode* = 0) ou entre 0 et 1 (*ixmode* différent de 0).

*xndx* -- varie sur la longueur de la table (*ixmode* = 0) ou dans l'intervalle allant de 0 à 1 (*ixmode* différent de 0).

*kfn* -- Numéro de table. Doit être  $\geq 1$ . Les valeurs flottantes sont arrondies à un entier. Si un numéro de table n'indique pas une table valide, ou si la table n'a pas encore été chargée (*GEN01*) une erreur se produit et l'instrument est désactivé.



### Attention avec les numéros de table au taux-k

Au taux-k, si un numéro de table  $< 1$  est donné, ou si le numéro de table indique une table inexistante ou une table de longueur nulle (devant être chargée à partir d'un fichier ultérieurement), une erreur se produit et l'instrument est désactivé. *kfn* doit être initialisé au taux

approprié en utilisant *init*. Si l'on essaie de charger une valeur de taux-i dans *kfn*, il y aura une erreur.

## Exemples

Voici un exemple de l'opcode *tablekt*. Il utilise le fichier *tablekt.csd* [examples/tablekt.csd].

### Exemple 1002. Exemple de l'opcode *tablekt*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tablekt.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

seed 0

gift1 ftgen 1, 0, 1024, 10, 1      ;sine wave
gift2 ftgen 2, 0, 1024, 10, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ;pulse

instr 1
andx phasor 400      ;phasor for reading the index
kfn init 1          ;initialize the choice of the function table
kmetro init 1       ;initialize the frequency of the metro
knewft metro kmetro ;make a new choice for selecting the function table once a second

if knewft == 1 then
  kfn = (kfn == 1 ? 2 : 1) ;switch between 1 and 2
  kmetro random .5, 2     ;create new metro frequency
  printk2 kfn
endif

ares tablekt andx, kfn, 1
outs ares, ares
endin

</CsInstruments>
<CsScore>
i 1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie comprendra des lignes comme celles-ci :

```
i1      2.00000
i1      1.00000
i1      2.00000
i1      1.00000
```

....

## Voir aussi

*tableikt*

## Crédits

Auteur : Robin Whittle

Australie

Mai 1997

Nouveau dans la version 3.47

# tablemix

tablemix — Mélange deux tables.

## Description

Mélange deux tables.

## Syntaxe

```
tablemix kdft, kdoff, klen, ks1ft, ks1off, ks1g, ks2ft, ks2off, ks2g
```

## Exécution

*kdft* -- Table de fonction destination.

*kdoff* -- Décalage de l'origine de l'écriture. Peut être négatif.

*klen* -- Nombre d'opérations d'écriture à réaliser. Une valeur négative signifie écrire avec des indices descendants.

*ks1ft*, *ks2ft* -- Tables de fonction sources. Peuvent être identique à la table destination, si l'on fait attention au sens d'écriture lors de la copie des données.

*ks1off*, *ks2off* -- Décalages de l'origine de la lecture dans les tables sources.

*ks1g*, *ks2g* -- Gains à appliquer lors de la lecture dans les tables source. Les résultats sont additionnés et la somme est écrite dans la table destination.

*tablemix* -- Cet opcode mélange deux tables, avec des gains séparés dans une table destination. L'écriture se fait sur *klen* positions, habituellement en avançant dans la table si *klen* est positif. S'il est négatif, l'écriture et la lecture se font avec des indices décroissants dans les tables. Cet option bi-directionnelle permet de déplacer facilement le contenu d'une table en lisant et en écrivant dans celle-ci avec un décalage différent.

Si *klen* vaut 0, il n'y a pas d'écriture. Noter que la valeur entière interne de *klen* est obtenue de la fonction *floor()* du C ANSI qui retourne l'entier négatif directement inférieur. Ainsi avec une valeur fractionnaire négative de *klen* de -2.3 on aura une longueur interne de 3, et la copie commencera à partir des positions décalées et se fera sur deux positions vers la gauche.

L'indice résultant pour la lecture et l'écriture dans les tables est calculé à partir du décalage de l'origine pour chaque table auquel est additionnée la valeur de l'index, qui commence à 0 et augmente ou diminue d'un pas unité tout au long du mixage.

Ces indices résultants peuvent devenir très grands, car il n'y a aucune restriction pour le décalage ou *klen*. Cependant l'indice résultant pour chaque table subit un ET logique avec un masque de longueur (tel que 0000 0111 pour une table de longueur 8) pour former l'indice final qui sera utilisé pour la lecture ou l'écriture. Ainsi il ne peut y avoir aucune lecture ou écriture en dehors des tables. C'est la même chose que le mode « wrap » (cyclique) dans la lecture et l'écriture de table. Ces opcodes ne lisent pas ou n'écrivent pas le point de garde. Si une table a été réécrite par l'un de ceux-ci et si elle a un point de garde sensé contenir la même valeur que la position 0, il faut ensuite appeler *tablegpw* afterwards.

Les indices et les décalages sont exprimés en pas de table - ils ne sont pas normalisés entre 0 et 1. Ainsi pour une table de longueur 256, *klen* doit être fixé à 256 si toute la table doit être lue ou écrite.

Il n'est pas nécessaire que les tables soient de même longueur - le parcours cyclique se fait individuellement pour chaque table.

## Exemples

Voici un exemple de l'opcode `tablemix`. Il utilise le fichier `tablemix.csd` [examples/tablemix.csd].

### Exemple 1003. Exemple de l'opcode `tablemix`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tablemix.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisinoid ftgen 1, 0, 256, 10, 1, 0, 0, .4 ;sinoid
gisaw    ftgen 2, 0, 1024, 7, 0, 256, 1 ;saw
gimix    ftgen 100, 0, 256, 7, 0, 256, 1 ;destination table

instr 1

kgain linseg 0, p3*.5, .5, p3*.5, 0
tablemix 100, 0, 256, 1, 0, 1, 2, 0, kgain
asig poscil .5, 110, gimix ;mix table 1 & 2
outs asig, asig

endin
</CsInstruments>
<CsScore>

i1 0 10

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

`tablecopy`, `tablegpw`, `tableicopy`, `tableigpw`, `tableimix`

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.47

# tableng

tableng — Interroge une table de fonction sur sa longueur.

## Description

Interroge une table de fonction sur sa longueur.

## Syntaxe

```
ires tableng ifn
kres tableng kfn
```

## Initialisation

*ifn* -- Numéro de la table à interroger.

## Exécution

*kfn* -- Numéro de la table à interroger.

*tableng* retourne la longueur de la table spécifiée. Ce sera une puissance de deux dans la plupart des cas. N'indique pas si une table a ou non un point de garde. Il semble que cette information ne soit pas disponible dans la structure de données de la table. Si la table spécifiée n'est pas trouvée, retourne 0.

Peut-être utile pour configurer le code d'opérations de manipulation de table, comme *tablemix* et *tablecopy*.

## Exemples

Voici un exemple de l'opcode *tableng*. Il utilise le fichier *tableng.csd* [examples/tableng.csd].

### Exemple 1004. Exemple de l'opcode *tableng*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tableng.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gifn1 ftgen 1, 0, 0, 1, "flute.aiff", 0, 0, 0 ;deferred-size table
```

```
instr 1

isize tableng 1
print isize
andx phasor 1 / (isize / sr) ;play at correct pitch
asig tab andx, 1, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 2.3
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra une ligne comme celle-ci :

```
instr 1: isize = 115506.000
```

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

# tablera

tablera — Lecture séquentielle de tables.

## Description

Cet opcode lit séquentiellement des tables vers une variable de taux-a. Il faut bien réfléchir à la manière de l'utiliser. Il a au moins deux applications principales, et assez différentes, qui sont discutées ci-dessous.

## Syntaxe

```
ares tablera kfn, kstart, koff
```

## Exécution

*ares* -- destination au taux-a de la lecture de *ksmps* valeurs depuis une table.

*kfn* -- numéro au taux-i ou au taux-k de la table à lire.

*kstart* -- où lire dans la table.

*koff* -- décalage au taux-i ou au taux-k dans la table. Intervalle illimité - voir les explications à la fin de la section.

Dans une application, *tablera* est apparié avec un *tablewa*, ou plusieurs opcodes *tablera* sont placés avant un *tablewa* -- tous partageant la même variable *kstart*.

Ceux-ci lisent depuis ou écrivent dans des positions adjacentes d'une table au taux audio, avec *ksmps* flottants écrits et lu à chaque cycle.

*tablera* commence à lire à la position *kstart*. *tablewa* commence à écrire à la position *kstart*, et continue à écrire à *kstart* le numéro de la position étant incrémenté d'une unité. (Noter que pour *tablewa*, *kstart* est à la fois une variable d'entrée et de sortie). Si l'index d'écriture atteint la fin de la table, aucune écriture ultérieure n'a lieu et zéro est écrit dans *kstart*.

Par exemple, si la longueur de la table est 16 (positions 0 à 15), et que *ksmps* vaut 5, les étapes suivantes se produiront lors d'appels répétés de l'opcode *tablewa*, en supposant que *kstart* est parti de 0.

Numéro de l'appel	kstart initial	kstart final	Positions écrites
1	0	5	0 1 2 3 4
2	5	10	5 6 7 8 9
3	10	15	10 11 12 13 14
4	15	0	15

Ceci facilite le traitement des données de table avec du code code d'orchestre standard au taux-a entre les opcodes *tablera* et *tablewa*. Il est ainsi permis d'utiliser tous les opérateurs de taux-k de Csound (avec précaution) sur des variables de taux-a, ce qui ne serait autrement possible qu'avec *ksmps* = 1, *downsamp* et *upsamp*.





## Plusieurs précautions

- Le code de taux-k dans la boucle de traitement est réellement exécuté au taux-a, si bien que les fonctions dépendant du temps comme *port* et *oscil* travaillent plus vite que d'habitude - leur code s'attendant à fonctionner au taux-k.
- Le système produira des effets indésirables si *ksmps* n'est pas compris dans la longueur de la table. Par exemple, une table de longueur 16 supportera de 1 à 16 échantillons, et donc le système fonctionnera avec *ksmps* compris entre 1 et 16.

Ces deux opcodes génèrent une erreur et désactivent l'instrument si une table de longueur  $< ksm\text{ps}$  est choisie. Il y aura également une erreur si *kstart* est inférieur à zéro ou supérieur à la position la plus haute dans la table - si *kstart* = longueur de la table.

- *kstart* est supposé contenir des valeurs entières comprises entre 0 et (longueur de la table - 1). Des valeurs fractionnaires entre celles-ci n'affecteront pas l'opération mais ne produiront rien de spécial.
- Ces opcodes sont sans interpolation et les paramètres *kstart* et *koff* sont toujours dans l'intervalle 0 à (longueur de la table - 1) - pas 0 à 1 comme c'est possible dans d'autres opcodes de lecture/écriture de table. *koff* peut se trouver en dehors de cet intervalle mais il y est ramené par le ET final.
- Ces opcodes sont en permanence en mode cyclique. Quand *koff* vaut 0, aucun repliement n'est nécessaire, car l'indice *kstart++* se trouve toujours dans l'intervalle normal de la table. *koff* différent de 0 peut conduire à un repliement.
- Le décalage n'affecte pas le nombre de cycles de lecture/écriture exécutés, ou la valeur écrite dans *kstart* par *tablewa*.
- Ces opcodes ne peuvent pas lire ou écrire le point de garde. Utiliser *tablegpw* pour écrire le point de garde après les manipulations effectuées avec *tablewa*.

## Exemples

```
kstart    =      0

lab1:
  atemp    tablera ktabsource, kstart, 0  ; Lit 5 valeurs de la table dans une
    ; variable de taux-a.

  atemp    =      log(atemp)  ; Traite les valeurs en utilisant
    ; du code de taux-a.

  kstart    tablewa ktabdest, atemp, 0    ; Ecriture dans la table

  if ktemp  0 goto lab1          ; Boucle jusqu'à ce que toute les positions
    ; de la table aient été traitées.
```

L'exemple ci-dessus montre une boucle de traitement qui s'exécute à chaque cycle-k, lisant chaque position dans la table *ktabsource*, et écrivant le logarithme de ces valeurs dans les mêmes positions de la table *ktabdest*.

Cela permet de manipuler en une fois, avec du code de taux-a, des tables entières, des parties de tables (avec décalages et différentes boucles de contrôle) et des données provenant de plusieurs tables, et de les écrire vers une autre (ou la même) table. C'est un peu compliqué mais c'est plus rapide que de le faire avec du code de lecture et d'écriture de taux-k.

Une autre application :

```

kzero = 0
kloop = 0

kzero tablewa 23, asignal, 0 ; écrit ksmps échantillons de taux-a
    ; dans les positions 0 à (ksmps -1) de la table 23.

lab1: ktemp table kloop, 23 ; Commence une boucle de ksmps itérations,
    ; dans laquelle chaque passage traite une des
    [ Du code pour manipuler ] ; valeurs de la table 23 avec du code de l'orchestre
    [ la valeur de ktemp. ] ; de taux-k.

tablew ktemp, kloop, 23 ; Ecrit la valeur traitée dans la table.

kloop = kloop + 1 ; Incrémente kloop, qui est à la fois
    ; le pointeur dans la table et le compteur de
if kloop < ksmps goto lab1 ; la boucle. Continue la boucle jusqu'à ce que
    ; toutes les valeurs dans la table aient été traitées.

asignal tablera 23, 0, 0 ; Copie le contenu de la table
    ; dans une variable de taux-a.

```

*koff*-- C'est un décalage qui est ajouté à la somme de *kstart* et de l'indice interne variable qui parcourt la table. Le résultat subit ensuite un ET logique avec le masque de longueur (000 0111 pour une table de longueur 8 - ou 9 avec un point de garde) et l'indice résultant est utilisé pour lire ou écrire dans la table. *koff* peut avoir n'importe quelle valeur. Il est converti en entier long au moyen de la fonction `ANSIfloor()` ; ainsi -4.3 devient -5. C'est le comportement désiré pour des décalages variant de part et d'autre de zéro.

Idéalement ce devrait être une variable facultative, valant 0 par défaut. Cependant, avec le code de lecture de l'orchestre `Csound` existant, de tels paramètres par défaut ne peuvent être que de `taux-i`. Nous voulons ici un paramètre de `taux-k` et donc, nous ne pouvons pas avoir de valeur par défaut.

## Voir aussi

*tablewa*

# tableseg

tableseg — Crée une nouvelle table de fonction en faisant des segments de droite entre les valeurs de tables de fonction en mémoire.

## Description

*tableseg* est comme *linseg* mais il interpole entre des valeurs stockées dans des tables de fonction. Le résultat est une nouvelle table de fonction passée en interne à tout *vpvoc* apparaissant avant le *tableseg* suivant (même fonctionnement que pour les paires *lpread/lpreson*). Les utilisations possibles sont décrites plus loin dans la notice de *vpvoc*.

## Syntaxe

```
tableseg ifn1, idur1, ifn2 [, idur2] [, ifn3] [...]
```

## Initialisation

*ifn1*, *ifn2*, *ifn3*, etc. -- numéros des tables de fonction. *ifn1*, *ifn2*, et les suivantes, doivent avoir la même taille.

*idur1*, *idur2*, etc. -- durée de l'interpolation d'une table à l'autre.

## Exemples

Voici un exemple de l'opcode *tableseg*. Il utilise le fichier *tableseg.csd* [exemples/tableseg.csd].

### Exemple 1005. Exemple de l'opcode *tableseg*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tableseg.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
; analyze "fox.wav" with PVANAL first
iend = p4
ktime line 0, p3, iend
tableseg p5, p3, p6 ;morph from table 1
asig vpvoc ktime, 1, "fox.pvx" ;to table 2
outs asig*3, asig*3
```

```
endin
</CsInstruments>
<CsScore>
f 1 0 512 9 .5 1 0
f 2 0 512 7 0 20 1 30 0 230 0 232 1

i 1 0 10 2.7 1 2
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvbufread, pvcross, pvinterp, pvread, tablexseg*

## Crédits

Auteur : Richard Karpen  
Seattle, Wash  
1997

Nouveau dans la version 3.44

# tableshuffle

tableshuffle — mélange le contenu d'une table de fonction de façon à ce que chaque élément de la table source se trouve dans une position aléatoire différente.

## Description

Cet opcode peut être utilisé pour mélanger le contenu de tables de fonction dans un ordre aléatoire mais en conservant tous les éléments. Imaginez la battue d'un jeu de cartes. Chaque élément de la table est copié dans une position aléatoire différente. Si cette position est déjà occupée, la position libre suivante est choisie. La longueur de la table reste la même.

## Syntaxe

```
tableshuffle ktablenum
tableshufflei itablenum
```

## Exécution

*ktablenum* ou *itablenum* -- le numéro de la table à mélanger.

## Exemples

Voici un exemple de l'opcode tableshuffle. Il utilise le fichier *farey7shuffled.csd* [examples/farey7shuffled.csd].

### Exemple 1006. Exemple de l'opcode tableshuffle.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac -+rtaudio=alsa --midioutfile=farey7.mid
</CsOptions>
<CsInstruments>
sr=48000
ksmps=10
nchnls=1
0dbfs = 1

gidelta init 100
gimult init 101

;----- loop and trigger instrument 901 using a Farey Sequence polyrhythm
instr 1
kindx init 0
kindx2 init 0
ktrigger init 0
ktime_unit init p6
kstart init p4
kloop init p5
kinitndx init 0
kfn_times init gidelta
knote init 60
kbasenote init p8
```

```

ifundam init p7
ktrigger seqtime ktime_unit, kstart, kloop, kinitndx, kfn_times
if (ktrigger > 0 ) then
    kpitch = cpspch(ifundam)
    kmult tab kindx2, gimult
    kpitch = kpitch * kmult
    knote = kbasenote + kmult
    event "i", 901, 0, .4, .1, kpitch, kpitch * .9, .4, 5, .75, .8, 1.0, .15, .0, .125, .125, .25
    kindx = kindx + 1
    kindx = kindx % kloop
    kindx2 = kindx2 + 1
    kindx2 = kindx2 % kloop
    if (kindx2 == 0) then
        tableshuffle gimult
    endif

endif
endin ; 1

;----- basic 2 Operators FM algorithm -----
instr 901
inotedur = p3
imaxamp = p4 ;ampdb(p4)
icarrfreq = p5
imodfreq = p6
ilowndx = p7
indxdiff = p8-p7
knote = p27
aampenv linseg p9, p14*p3, p10, p15*p3, p11, p16*p3, p12, p17*p3, p13
adevenv linseg p18, p23*p3, p19, p24*p3, p20, p25*p3, p21, p26*p3, p22
amodosc oscili (ilowndx+indxdiff*adevenv)*imodfreq, imodfreq, 10
acarosc oscili imaxamp*aampenv, icarrfreq+amodosc, 10
out acarosc
;----- we also write down a midi track here -----
midion 1, knote, 100
endin ; 901

</CsInstruments>

<CsScore>
f10 0 4096 10 1
f100 0 -18 "farey" 7 1
f101 0 -18 "farey" 7 2

; p4 kstart := index offset into the Farey Sequence
; p5 kloop := end index into Farey Seq.
; p6 timefac := time in seconds for one loop to complete
; p7 fundam := fundamental of the FM instrument
; p8 basenote:= root pitch of the midi voice output
; note that pitch structures of the midi file output are not equivalent to the
; ones used for the FM real-time synthesis.

; start dur kstart kloop timefac fundam. basenote
i1 0.0 44 0 18 1 6.05 60
i1 4 40 0 18 3 7.05 72
i1 10 38 0 18 1.5 8 84
i1 15 50 0 18 1 5 48
i1 22 75 5 17 1.7 4 36
e
</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

*GEN farey, tablefilter, tablecopy*

## Crédits

Auteur : Georg Boenn  
Université de Glamorgan, UK

Nouveau dans la version 5.13 de Csound.

# tablew

tablew — Change le contenu de tables de fonction existantes.

## Description

Cet opcode opère sur des tables de fonction existantes en changeant leur contenu. *tablew* sert à l'écriture au taux-k ou au taux-a, le numéro de table étant spécifié durant l'initialisation. L'utilisation de *tablew* avec un signal et des valeurs d'indice de taux-i est permise, mais les données spécifiées seront toujours écrites dans la table de fonction au taux-k, pas durant la passe d'initialisation. Les combinaisons valides des types de variable sont indiquées par la première lettre des noms de variable.

## Syntaxe

```
tablew asig, andx, ifn [, ixmode] [, ixoff] [, iwgmodes]
```

```
tablew isig, indx, ifn [, ixmode] [, ixoff] [, iwgmodes]
```

```
tablew ksig, kndx, ifn [, ixmode] [, ixoff] [, iwgmodes]
```

## Initialisation

*asig, isig, ksig* -- La valeur à écrire dans la table.

*andx, indx, kndx* -- Indice dans la table, un nombre positif compris entre 0 et la longueur de la table (*ixmode* = 0) ou entre 0 et 1 (*ixmode* différent de 0).

*ifn* -- Numéro de la table. Doit être  $\geq 1$ . Les nombres flottants sont arrondis à l'entier inférieur. Si un numéro de table ne pointe pas vers une table valide, ou si la table n'a pas encore été chargée (*GEN01*) une erreur est générée et l'instrument est désactivé.

*ixmode* (facultatif, 0 par défaut) -- mode d'indexation.

- 0 = *xndx* et *ixoff* sont compris entre 0 et la longueur de la table.
- différent de 0 = *xndx* et *ixoff* sont compris entre 0 et 1.

*ixoff* (facultatif, 0 par défaut) -- décalage de l'index.

- 0 = l'indice résultant est contrôlé directement par *xndx*, l'indexation commençant depuis le début de la table.
- Différent de 0 = l'indexation démarre dans la table. La valeur doit être positive et inférieure à la longueur de la table (*ixmode* = 0) ou inférieure à 1 (*ixmode* différent de 0).

*iwgmodes* (facultatif, 0 par défaut) -- mode cyclique et point de garde.

- 0 = mode limite.
- 1 = mode cyclique.
- 2 = mode point de garde.



## Exécution

### Mode limite (0)

Limite l'indice résultant ( $xndx + ixoff$ ) entre 0 et le point de garde. Pour une table de longueur 5, cela signifie que les positions allant de 0 à 3 et la position 4 (le point de garde) peuvent être écrites. Un indice résultant négatif provoque l'écriture en position 0.

### Mode cyclique (1)

Parcours cyclique de l'indice résultant dans les positions 0 à E, où E vaut soit la longueur de la table moins un, soit le facteur de 2 qui est égal à la longueur de la table moins un. Par exemple, un parcours cyclique entre 0 et 3, si bien que l'indice 6 signifie une écriture dans la position 2.

### Mode point de garde (2)

Le point de garde est écrit en même temps que la position 0 avec la même valeur.

Facilite l'écriture dans des tables prévues pour être lues avec interpolation pour produire des formes d'onde cycliques sans discontinuité. De plus, avant son utilisation, l'indice résultant est augmenté de la moitié de la distance entre une position et la suivante, avant d'être arrondi à l'adresse entière inférieure d'une position dans la table.

Normalement ( $igwmode = 0$  ou  $1$ ), pour une table de longueur 5, qui comprend les positions 0 à 3 en partie principale et la position 4 comme point de garde, un indice résultant compris entre 0 et 0.999 provoquera une écriture dans la position 0. ("0.999" signifie juste inférieur à 1.0), entre 1.0 et 1.999, l'écriture se fera dans la position 1, etc. La même interprétation a lieu pour les indices résultants compris entre 0 et 4.999 ( $igwmode = 0$ ) ou 3.999 ( $igwmode = 1$ ).  $igwmode = 0$  permet l'écriture dans les positions 0 à 4, avec la possibilité d'avoir dans le point de garde (4) une valeur différente de celle de la position 0.

Avec une table de longueur 5 et  $igwmode = 2$ , quand l'indice résultant est compris entre 0 et 0.499, l'écriture se fera dans les positions 0 et 4. S'il est compris entre 0.5 et 1.499, l'écriture se fera dans la position 1, etc. S'il est compris entre 3.5 et 4.0, l'écriture se fera également dans les positions 0 et 4.

Ainsi, l'écriture s'approche le plus possible des résultats de la lecture avec interpolation. Le mode point de garde ne doit être utilisé qu'avec des tables qui ont un point de garde.

Le mode point de garde se fait en ajoutant 0.5 à l'indice résultant, en l'arrondissant à l'entier inférieur le plus proche, puis en le réduisant modulo le facteur de deux égal à la longueur de la table moins un, enfin en écrivant dans la table (positions 0 à 3 dans notre exemple) et dans le point de garde si l'indice vaut 0.

*tablew* ne retourne pas de valeur. Les trois derniers paramètres sont facultatifs et valent 0 par défaut.

## Avertissement pour les numéros de table de taux-k

Au taux-k ou au taux-a, si l'on donne un numéro de table  $< 1$ , ou si le numéro de table pointe vers une table inexistante ou vers une table de longueur nulle (qui doit être chargée depuis un fichier ultérieurement), une erreur est générée et l'instrument est désactivé. Il faut initialiser *kfn* et *afn* au taux approprié en utilisant *init*. Si l'on essaie de mettre une valeur de taux-i dans *kfn* ou dans *afn* une erreur est générée.



### Avertissement

Noter que *tablew* est toujours un opcode de taux-k. Cela signifie que même sa version de taux-i est exécutée au taux-k et écrit la valeur de la variable de taux-i. Pour cette raison, le code suivant ne se comportera pas comme prévu :

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
giFt ftgen 1, 0, 8, 2, 0
instr 1
indx = 0
      tablew 10, indx, giFt
ival tab_i indx, giFt
      print ival
endin
</CsInstruments>
<CsScore>
i 1 0 1
</CsScore>
</CsoundSynthesizer>
```

Alors que l'on s'attend à ce que ce programme imprime un 10 sur la console, il imprimera 0 car *tab\_i* lit la valeur à l'initialisation de la note, avant la première passe d'exécution durant laquelle *tablew* écrit sa valeur.

## Voir aussi

*tableiw, tablewkt*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

# tablewa

tablewa — Ecrit dans une table à des positions adjacentes.

## Description

Cet opcode écrit dans une table à des positions adjacentes depuis un signal de taux-a. Un peu de réflexion est nécessaire avant de l'utiliser. Il a au moins deux applications principales et assez différentes discutées ci-dessous.

## Syntaxe

`kstart tablewa kfn, asig, koff`

## Exécution

*kstart* -- Position de lecture ou d'écriture dans la table.

*kfn* -- numéro de taux-i ou -k de la table à lire ou écrire.

*asig* -- signal de taux-a dont viennent les valeurs à écrire dans la table.

*koff* -- décalage de taux-i ou k dans la table. Intervalle illimité - voir les explications à la fin de cette section.

Dans une application, on utilise un ou plusieurs opcodes *tablera* avant un *tablewa* -- tous partageant la même variable *kstart*.

Ceux-ci lisent depuis ou écrivent dans des positions adjacentes d'une table au taux audio, avec *ksmps* flottants écrits et lu à chaque cycle.

*tablera* commence à lire à la position *kstart*. *tablewa* commence à écrire à la position *kstart*, et continue à écrire à *kstart* le numéro de la position étant incrémenté d'une unité. (Noter que pour *tablewa*, *kstart* est à la fois une variable d'entrée et de sortie). Si l'index d'écriture atteint la fin de la table, aucune écriture ultérieure n'a lieu et zéro est écrit dans *kstart*.

Par exemple, si la longueur de la table est 16 (positions 0 à 15) et que *ksmps* vaut 5, alors les étapes suivantes se produiront à chaque appel de l'opcode *tablewa*, en supposant que *kstart* est parti 0.

Numéro de l'appel	<i>kstart</i> initial	<i>kstart</i> final	Positions écrites
1	0	5	0 1 2 3 4
2	5	10	5 6 7 8 9
3	10	15	10 11 12 13 14
4	15	0	15

Ceci facilite le traitement des données de table avec du code d'orchestre standard au taux-a entre les opcodes *tablera* et *tablewa*. Il est ainsi permis d'utiliser tous les opérateurs de taux-k de Csound (avec précaution) sur des variables de taux-a, ce qui ne serait autrement possible qu'avec *ksmps* = 1, *downsamp* et *upsamp*.



## Plusieurs précautions

- Le code de taux-k dans la boucle de traitement est réellement exécuté au taux-a, si bien que les fonctions dépendantes du temps comme *port* et *oscil* travaillent plus vite que d'habitude - leur code s'attendant à fonctionner au taux-k.
- Le système produira des effets indésirables si *ksmps* n'est pas compris dans la longueur de la table. Par exemple, une table de longueur 16 supportera de 1 à 16 échantillons, et donc le système fonctionnera avec *ksmps* compris entre 1 et 16.

Ces deux opcodes génèrent une erreur et désactivent l'instrument si une table de longueur  $< ksm\text{ps}$  est choisie. Il y aura également une erreur si *kstart* est inférieur à zéro ou supérieur à la position la plus haute dans la table - si *kstart* = longueur de la table.

- *kstart* est supposé contenir des valeurs entières comprises entre 0 et (longueur de la table - 1). Des valeurs fractionnaires entre celles-ci n'affecteront pas l'opération mais ne produiront rien de spécial.
- Ces opcodes sont sans interpolation et les paramètres *kstart* et *koff* sont toujours dans l'intervalle 0 à (longueur de la table - 1) - pas 0 à 1 comme c'est possible dans d'autres opcodes de lecture/écriture de table. *koff* peut se trouver en dehors de cet intervalle mais il y est ramené par le ET final.
- Ces opcodes sont en permanence en mode cyclique. Quand *koff* vaut 0, aucun repliement n'est nécessaire, car l'indice *kstart++* se trouve toujours dans l'intervalle normal de la table. *koff* différent de 0 peut conduire à un repliement.
- Le décalage n'affecte pas le nombre de cycles de lecture/écriture exécutés, ou la valeur écrite dans *kstart* par *tablewa*.
- Ces opcodes ne peuvent pas lire ou écrire le point de garde. Utiliser *tablegpw* pour écrire le point de garde après les manipulations effectuées avec *tablewa*.

## Exemples

```
kstart    =      0

lab1:
  atemp    tablera ktabsource, kstart, 0  ; Lit 5 valeurs de la table dans une
    ; variable de taux-a.

  atemp    =      log(atemp)  ; Traite les valeurs en utilisant
    ; du code de taux-a.

  kstart    tablewa ktabdest, atemp, 0    ; Ecriture dans la table

if ktemp  0 goto lab1      ; Boucle jusqu'à ce que toute les positions
    ; de la table aient été traitées.
```

L'exemple ci-dessus montre une boucle de traitement qui s'exécute à chaque cycle-k, lisant chaque position dans la table *ktabsource*, et écrivant le logarithme de ces valeurs dans les mêmes positions de la table *ktabdest*.

Cela permet de manipuler en une fois, avec du code de taux-a, des tables entières, des parties de tables (avec décalages et différentes boucles de contrôle) et des données provenant de plusieurs tables, et de les écrire vers une autre (ou la même) table. C'est un peu compliqué mais c'est plus rapide que de le faire avec du code de lecture et d'écriture de taux-k.

Une autre application :

```

kzero = 0
kloop = 0

kzero tablewa 23, asignal, 0 ; écrit ksmps échantillons de taux-a
    ; dans les positions 0 à (ksmps -1) de la table 23.

lab1: ktemp table kloop, 23 ; Commence une boucle de ksmps itérations,
    ; dans laquelle chaque passage traite une des
    [ Du code pour manipuler ] ; valeurs de la table 23 avec du code de l'orchestre
    [ la valeur de ktemp. ] ; de taux-k.

tablew ktemp, kloop, 23 ; Ecrit la valeur traitée dans la table.

kloop = kloop + 1 ; Incrémente kloop, qui est à la fois
    ; le pointeur dans la table et le compteur de
if kloop < ksmps goto lab1 ; la boucle. Continue la boucle jusqu'à ce que
    ; toutes les valeurs dans la table aient été traitées.

asignal tablera 23, 0, 0 ; Copie le contenu de la table
    ; dans une variable de taux-a.

```

*koff*-- C'est un décalage qui est ajouté à la somme de *kstart* et de l'indice interne variable qui parcourt la table. Le résultat subit ensuite un ET logique avec le masque de longueur (000 0111 pour une table de longueur 8 - ou 9 avec un point de garde) et l'indice résultant est utilisé pour lire ou écrire dans la table. *koff* peut avoir n'importe quelle valeur. Il est converti en entier long au moyen de la fonction *ANSIfloor()* ; ainsi -4.3 devient -5. C'est le comportement désiré pour des décalages variant de part et d'autre de zéro.

Idéalement ce devrait être une variable facultative, valant 0 par défaut. Cependant, avec le code de lecture de l'orchestre *Csound* existant, de tels paramètres par défaut ne peuvent être que de *taux-i*. Nous voulons ici un paramètre de *taux-k* et donc, nous ne pouvons pas avoir de valeur par défaut.

## Crédits

Auteur : Robin Whittle  
Australie

# tablewkt

tablewkt — Change le contenu de tables de fonction existantes.

## Description

Cet opcode opère sur des tables de fonction existantes en changeant leur contenu. *tablewkt* utilise une variable de taux-k pour choisir le numéro de table. Les combinaisons valides des types de variable sont données par la première lettre des noms de variable.

## Syntaxe

```
tablewkt asig, andx, kfn [, ixmode] [, ixoff] [, iwemode]
```

```
tablewkt ksig, kndx, kfn [, ixmode] [, ixoff] [, iwemode]
```

## Initialisation

*asig, ksig* -- La valeur à écrire dans la table.

*andx, kndx* -- Indice dans la table, un nombre positif compris entre 0 et la longueur de la table (*ixmode* = 0) ou entre 0 et 1 (*ixmode* différent de 0).

*kfn* -- Numéro de la table. Doit être  $\geq 1$ . Les nombres flottants sont arrondis à l'entier inférieur. Si un numéro de table ne pointe pas vers une table valide, ou si la table n'a pas encore été chargée (*GEN01*) une erreur est générée et l'instrument est désactivé.

*ixmode* -- mode d'indexation. Zéro par défaut.

- 0 = *xndx* et *ixoff* sont compris entre 0 et la longueur de la table.
- Différent de 0 = *xndx* et *ixoff* sont compris entre 0 et 1.

*ixoff* -- décalage de l'index. 0 par défaut.

- 0 = l'indice résultant est contrôlé directement par *xndx*, l'indexation commençant depuis le début de la table.
- Différent de 0 = l'indexation démarre dans la table. La valeur doit être positive et inférieure à la longueur de la table (*ixmode* = 0) ou inférieure à 1 (*ixmode* différent de 0).

*iwemode* -- mode d'écriture dans la table. 0 par défaut.

- 0 = mode limite.
- 1 = mode cyclique.
- 2 = ode point de garde.

## Exécution

### Mode limite (0)

Limite l'indice résultant ( $xndx + ixoff$ ) entre 0 et le point de garde. Pour une table de longueur 5, cela signifie que les positions allant de 0 à 3 et la position 4 (le point de garde) peuvent être écrites. Un indice résultant négatif provoque l'écriture en position 0.

## Mode cyclique (1)

Parcours cyclique de l'indice résultant dans les positions 0 à E, où E vaut soit la longueur de la table moins un, soit le facteur de 2 qui est égal à la longueur de la table moins un. Par exemple, un parcours cyclique entre 0 et 3, si bien que l'indice 6 signifie une écriture dans la position 2.

## Mode point de garde (2)

Le point de garde est écrit en même temps que la position 0 avec la même valeur.

Facilite l'écriture dans des tables prévues pour être lues avec interpolation pour produire des formes d'onde cycliques sans discontinuité. De plus, avant son utilisation, l'indice résultant est augmenté de la moitié de la distance entre une position et la suivante, avant d'être arrondi à l'adresse entière inférieure d'une position dans la table.

Normalement (*iwgmode* = 0 ou 1), pour une table de longueur 5, qui comprend les positions 0 à 3 en partie principale et la position 4 comme point de garde, un indice résultant compris entre 0 et 0.999 provoquera une écriture dans la position 0. ("0.999" signifie juste inférieur à 1.0), entre 1.0 et 1.999, l'écriture se fera dans la position 1, etc. La même interprétation a lieu pour les indices résultants compris entre 0 et 4.999 (*igwmode* = 0) ou 3.999 (*igwmode* = 1). *igwmode* = 0 permet l'écriture dans les positions 0 à 4, avec la possibilité d'avoir dans le point de garde (4) une valeur différente de celle de la position 0.

Avec une table de longueur 5 et *iwgmode* = 2, quand l'indice résultant est compris entre 0 et 0.499, l'écriture se fera dans les positions 0 et 4. S'il est compris entre 0.5 et 1.499, l'écriture se fera dans la position 1, etc. S'il est compris entre 3.5 et 4.0, l'écriture se fera également dans les positions 0 et 4.

Ainsi, l'écriture s'approche le plus possible des résultats de la lecture avec interpolation. Le mode point de garde ne doit être utilisé qu'avec des tables qui ont un point de garde.

Le mode point de garde se fait en ajoutant 0.5 à l'indice résultant, en l'arrondissant à l'entier inférieur le plus proche, puis en le réduisant modulo le facteur de deux égal à la longueur de la table moins un, enfin en écrivant dans la table (positions 0 à 3 dans notre exemple) et dans le point de garde si l'indice vaut 0.

## Avertissement pour les numéros de table de taux-k

Au taux-k ou au taux-a, si l'on donne un numéro de table < 1, ou si le numéro de table pointe vers une table inexistante ou vers une table de longueur nulle (qui doit être chargée depuis un fichier ultérieurement), une erreur est générée et l'instrument est désactivé. Il faut initialiser *kfn* et *afn* au taux approprié en utilisant *init*. Si l'on essaie de mettre une valeur de taux-i dans *kfn* ou dans *afn* une erreur est générée.

## Voir aussi

*tableiw*, *tablew*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

# tablexkt

tablexkt — Lit des tables de fonction avec interpolation linéaire, cubique ou sinc.

## Description

Lit des tables de fonction avec interpolation linéaire, cubique ou sinc.

## Syntaxe

```
ares tablexkt xndx, kfn, kwarp, iwsiz [, ixmode] [, ixoff] [, iwrap]
```

## Initialisation

*iwsiz* -- Ce paramètre contrôle le type d'interpolation à utiliser :

- 2 : Interpolation linéaire. C'est la qualité la plus faible, mais aussi le mode le plus rapide.
- 4 : Interpolation cubique. Qualité légèrement meilleure qu'avec *iwsiz* = 2, au prix d'un traitement moins rapide.
- 8 et au-dessus (jusqu'à 1024) : interpolation sinc avec une taille de fenêtre égale à *iwsiz* (doit être un multiple entier de 4). Meilleure qualité que l'interpolation linéaire ou cubique, mais très lent. Lorsque l'on transpose vers le haut, on peut utiliser une valeur de *kwarp* supérieure à 1 pour un meilleur lissage (c'est encore plus lent).

*ixmode* (facultatif) -- mode d'indexation. La valeur par défaut est 0.

- 0 : indices bruts
- valeur différente de 0 : normalisé (0 à 1)



### Notes

Si l'on utilise *tablexkt* pour reproduire des échantillons avec boucle (par exemple avec un indice de table généré par *lphasor*), il faut qu'il y ait au moins *iwsiz* / 2 échantillons après la fin de la boucle pour l'interpolation, sinon il pourra y avoir des clics audibles (il doit y avoir aussi au moins *iwsiz* / 2 échantillons avant le début de la boucle).

*ixoff* (facultatif) -- valeur de décalage de l'indice. Pour une table dont l'origine est au centre, il faut utiliser *taille\_table* / 2 (indices bruts) ou 0.5 (indices normalisés). La valeur par défaut est 0.

*iwrap* (facultatif) -- indicateur de parcours cyclique des indices. La valeur par défaut est 0.

- 0 : Pas de lecture cyclique (les indices < 0 sont ramenés à 0 ; les indices >= à la taille de la table (ou 1.0 en mode normalisé) restent bloqués sur le point de garde).
- valeur différente de 0 : l'indice est replié dans l'intervalle autorisé (sans inclure le point de garde dans ce cas).



### Note

*iwrap* s'applique aussi aux échantillons supplémentaires pour l'interpolation.



## Exécution

*ares* -- sortie audio.

*xndx* -- index de la table.

*kfn* -- numéro de la table de fonction.

*kwarp* -- s'il est supérieur à 1, on utilise la fonction  $\sin(x / \textit{kwarp}) / x$  pour l'interpolation sinc au lieu de la fonction par défaut  $\sin(x) / x$ . C'est utile pour lisser lorsque l'on transpose vers le haut (*kwarp* doit être fixé au facteur de transposition dans ce cas, par exemple 2.0 pour une octave), cependant le rendu peut-être jusqu'à deux fois plus lent. De plus, *iwsiz*e doit valoir au moins *kwarp* \* 8. Cette possibilité est expérimentale et pourra être améliorée à la fois en termes de vitesse et de qualité dans de nouvelles versions.



### Note

*kwarp* n'a aucun effet s'il est inférieur ou égal à 1, ou si l'interpolation linéaire ou cubique est utilisée.

## Exemples

Voici un exemple de l'opcode `tablexkt`. Il utilise le fichier `tablexkt.csd` [examples/tablexkt.csd].

### Exemple 1007. Exemple de l'opcode `tablexkt`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tablexkt.wav -W ;; for file output any platform
</CsOptions>

<CsInstruments>
;Example by Jonathan Murphy

sr      = 44100
ksmps   = 10
nchnls  = 1

instr 1

ifn      = 1      ; query f1 as to number of samples
ilen     = nsamp(ifn)

itrns    = 4      ; transpose up 4 octaves
ilps     = 16     ; allow iwsiz/2 samples at start
ilpe     = ilen - 16 ; and at end
imode    = 3      ; loop forwards and backwards
istrt    = 16     ; start 16 samples into loop

alphs    lphasor   itrns, ilps, ilpe, imode, istrt
; use lphasor as index
andx     = alphs
```

```
kfn      = 1    ; read f1
kwarp    = 4    ; anti-aliasing, should be same value as itrns above
iwsiz    = 32   ; iwsiz must be at least 8 * kwarp

atab     tablexkt andx, kfn, kwarp, iwsiz

atab     = atab * 10000

        out      atab

        endin

</CsInstruments>

<CsScore>
f 1 0 262144 1 "beats.wav" 0 4 1
i1 0 60
e
</CsScore>

</CsoundSynthesizer>
```

## Crédits

Auteur : Istvan Varga  
Janvier 2002  
Exemple par Jonathan Murphy 2006

Nouveau dans la version 4.18

# tablexseg

**tablexseg** — Crée une nouvelle table de fonction en faisant des segments d'exponentielle entre les valeurs de tables de fonction en mémoire.

## Description

*tablexseg* est comme *expseg* mais il interpole entre des valeurs stockées dans des tables de fonction. Le résultat est une nouvelle table de fonction passée en interne à tout *vpvoc* apparaissant avant le *tablexseg* suivant (même fonctionnement que pour les paires *lpread/lpreson*). Les utilisations possibles sont décrites plus loin dans la notice de *vpvoc*.

## Syntaxe

```
tablexseg ifn1, idur1, ifn2 [, idur2] [, ifn3] [...]
```

## Initialisation

*ifn1*, *ifn2*, *ifn3*, etc. -- numéros des tables de fonction. *ifn1*, *ifn2*, et les suivantes, doivent avoir la même taille.

*idur1*, *idur2*, etc. -- durée de l'interpolation d'une table à l'autre.

## Exemples

Voici un exemple de l'opcode *tablexseg*. Il utilise le fichier *tablexseg.csd* [examples/tablexseg.csd].

### Exemple 1008. Exemple de l'opcode *tablexseg*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tablexseg.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
; analyze "fox.wav" with PVANAL first
iend = p4
ktime line 0, p3, iend
tablexseg p5, p3, p6 ;morph from table 1
asig vpvoc ktime, 1, "fox.pvx" ;to table 2
outs asig*3, asig*3
```

```
endin
</CsInstruments>
<CsScore>
f 1 0 512 9 .5 1 0
f 2 0 512 5 1 60 0.01 390 0.01 62 1

i 1 0 5 2.7 1 2
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvbufread, pvcross, pvinterp, pvread, tableseg*

## Crédits

Auteur : Richard Karpen  
Seattle, WA USA  
1997

# tabmorph

tabmorph — Permet le fondu enchaîné entre un ensemble de tables.

## Description

*tabmorph* permet le fondu enchaîné entre un ensemble de tables de la même taille au moyen d'une moyenne pondérée entre deux tables sélectionnées dans l'ensemble.

## Syntaxe

```
kout tabmorph kindex, kweightpoint, ktabnum1, ktabnum2, \  
    ifn1, ifn2 [, ifn3, ifn4, ..., ifnN]
```

## Initialisation

*ifn1, ifn2 [, ifn3, ifn4, ..., ifnN]* -- numéros des tables de fonction. C'est un ensemble de tables choisies par l'utilisateur pour le fondu enchaîné. Toutes les tables doivent avoir la même longueur. Il faut être conscient que seulement deux de ces tables peuvent être choisies en même temps pour le fondu enchaîné. Comme il est possible d'utiliser des nombres non-entiers pour les arguments *ktabnum1* et *ktabnum2*, le fondu enchaîné est le résultat d'une interpolation entre des tables adjacentes consécutives de l'ensemble.

## Exécution

*kout* -- la valeur retournée pour l'indice *kindex*, résultant du fondu enchaîné de deux tables (voir ci-dessous).

*kindex* -- indice principal de la table résultant du fondu enchaîné. L'intervalle va de 0 à la longueur de la table (exclue).

*kweightpoint* -- le poids de l'influence d'une paire de tables sélectionnées dans le fondu enchaîné. Cet argument est compris entre 0 et 1. 0 provoque la sortie de la première table inchangée, 1 provoque la sortie de la seconde table de la paire inchangée. Toutes les valeurs intermédiaires entre 0 et 1 déterminent la gradation du fondu enchaîné entre les deux tables de la paire.

*ktabnum1* -- la première table choisie pour le fondu enchaîné. Ce nombre n'exprime pas directement le numéro de la table mais la position de celle-ci dans la séquence de l'ensemble (de 0 à N-1). Si ce nombre est entier, la table correspondante est choisie inchangée. S'il contient une partie fractionnaire, alors une interpolation avec la table adjacente suivante a lieu.

*ktabnum2* -- la deuxième table choisie pour le fondu enchaîné. Ce nombre n'exprime pas directement le numéro de la table mais la position de celle-ci dans la séquence de l'ensemble (de 0 à N-1). Si ce nombre est entier, la table correspondante est choisie inchangée. S'il contient une partie fractionnaire, alors une interpolation avec la table adjacente suivante a lieu.

La famille d'opcodes *tabmorph* est semblable à la famille *table*, mais elle permet un fondu enchaîné entre deux tables choisies dans un ensemble de tables. D'abord, l'utilisateur doit fournir un ensemble de tables d'égale longueur (*ifn1, ifn2 [, ifn3, ifn4, ..., ifnN]*). Ensuite, il peut choisir une paire de tables dans l'ensemble afin d'effectuer le fondu enchaîné : *ktabnum1* et *ktabnum2* reçoivent des nombres (0 représente la première table dans l'ensemble, 1 la seconde, 2 la troisième et ainsi de suite). Puis il détermine le fondu enchaîné entre les deux tables choisies, avec le paramètre *kweightpoint*. Après cela, la table résultante peut

être indexée avec le paramètre *kindex* comme pour un opcode *table* normal. Si la valeur de ce paramètre dépasse la longueur de table (qui doit être la même pour toutes les tables), elle est repliée cycliquement.

*tabmorph* agit comme l'opcode *table*, c'est-à-dire sans interpolation. Cela signifie qu'il tronque la partie fractionnaire de l'argument *kindex*. Par contre, les parties fractionnaires de *ktabnum1* et de *ktabnum2* sont significatives, provoquant une interpolation linéaire entre les éléments correspondants de deux tables adjacentes consécutives.

## Exemples

Voici un exemple de l'opcode *tabmorph*. Il utilise le fichier *tabmorph.csd* [examples/tabmorph.csd].

### Exemple 1009. Exemple de l'opcode *tabmorph*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tabmorph.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine   ftgen 0, 0, 8193, 10, 1      ;sine wave
giSquare ftgen 0, 0, 8193, 7, 1, 4096, 1, 0, -1, 4096, -1 ;square wave
giTri    ftgen 0, 0, 8193, 7, 0, 2048, 1, 4096, -1, 2048, 0 ;triangle wave
giSaw    ftgen 0, 0, 8193, 7, 1, 8192, -1 ;sawtooth wave, downward slope

instr    1

iamp     = .7
kindex   phasor 440      ;read table value at this index
kindex   = kindex*8192    ;for all 8192 index points
kweightpoint = 0.5      ;set weightpoint
ktabnum1 line 0, p3, 3    ;morph through all tables
ktabnum2 = 2             ;set to triangle wave
ksig     tabmorph kindex, kweightpoint, ktabnum1, ktabnum2, giSine, giSquare, giTri, giSaw
ksig     = ksig*iamp
asig     interp ksig      ;convert to audio signal
outs     asig, asig

endin
</CsInstruments>
<CsScore>

i1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*table*, *tabmorphi*, *tabmorpha*, *tabmorphak*, *ftmorf*,

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# tabmorph

tabmorph — Permet le fondu enchaîné entre un ensemble de tables au taux audio avec interpolation.

## Description

*tabmorph* permet le fondu enchaîné entre un ensemble de tables de la même taille au moyen d'une moyenne pondérée entre deux tables sélectionnées dans l'ensemble.

## Syntaxe

```
aout tabmorph aindex, aweightpoint, atabnum1, atabnum2, \  
    ifn1, ifn2 [, ifn3, ifn4, ... ifnN]
```

## Initialisation

*ifn1, ifn2, ifn3, ifn4, ..., ifnN* -- numéros des tables de fonction. C'est un ensemble de tables choisies par l'utilisateur pour le fondu enchaîné. Toutes les tables doivent avoir la même longueur. Il faut être conscient que seulement deux de ces tables peuvent être choisies en même temps pour le fondu enchaîné. Comme il est possible d'utiliser des nombres non-entiers pour les arguments *atabnum1* et *atabnum2*, le fondu enchaîné est le résultat d'une interpolation entre des tables adjacentes consécutives de l'ensemble.

## Exécution

*aout* -- la valeur retournée pour l'indice *aindex*, résultant du fondu enchaîné de deux tables (voir ci-dessous).

*aindex* -- indice principal de la table résultant du fondu enchaîné. L'intervalle va de 0 à la longueur de la table (exclue).

*aweightpoint* -- le poids de l'influence d'une paire de tables sélectionnées dans le fondu enchaîné. Cet argument est compris entre 0 et 1. 0 provoque la sortie de la première table inchangée, 1 provoque la sortie de la seconde table de la paire inchangée. Toutes les valeurs intermédiaires entre 0 et 1 déterminent la gradation du fondu enchaîné entre les deux tables de la paire.

*atabnum1* -- la première table choisie pour le fondu enchaîné. Ce nombre n'exprime pas directement le numéro de la table mais la position de celle-ci dans la séquence de l'ensemble (de 0 à N-1). Si ce nombre est entier, la table correspondante est choisie inchangée. S'il contient une partie fractionnaire, alors une interpolation avec la table adjacente suivante a lieu.

*atabnum2* -- la deuxième table choisie pour le fondu enchaîné. Ce nombre n'exprime pas directement le numéro de la table mais la position de celle-ci dans la séquence de l'ensemble (de 0 à N-1). Si ce nombre est entier, la table correspondante est choisie inchangée. S'il contient une partie fractionnaire, alors une interpolation avec la table adjacente suivante a lieu.

La famille d'opcodes *tabmorph* est semblable à la famille *table*, mais elle permet un fondu enchaîné entre deux tables choisies dans un ensemble de tables. D'abord, l'utilisateur doit fournir un ensemble de tables d'égale longueur (*ifn1, ifn2 [, ifn3, ifn4, ..., ifnN]*). Ensuite, il peut choisir une paire de tables dans l'ensemble afin d'effectuer le fondu enchaîné : *atabnum1* et *atabnum2* reçoivent des nombres (0 représente la première table dans l'ensemble, 1 la seconde, 2 la troisième et ainsi de suite). Puis il détermine le fondu enchaîné entre les deux tables choisies, avec le paramètre *aweightpoint*. Après cela, la table résultante peut être indexée avec le paramètre *aindex* comme pour un opcode *table* normal. Si la valeur de ce paramètre dépasse la longueur de table (qui doit être la même pour toutes les tables), elle est repliée cycliquement.



*tabmorpha* est la version au taux audio de *tabmorphi* (il utilise l'interpolation). Tous les arguments d'entrée fonctionnent au taux-a.

## Exemples

Voici un exemple de l'opcode *tabmorpha*. Il utilise le fichier *tabmorpha.csd* [examples/tabmorpha.csd].

### Exemple 1010. Exemple de l'opcode *tabmorpha*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tabmorpha.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs  =1

giSine ftgen 0, 0, 8193, 10, 1      ; sine wave
giSquare ftgen 0, 0, 8193, 7, 1, 4096, 1, 0, -1, 4096, -1    ; square wave
giTri ftgen 0, 0, 8193, 7, 0, 2048, 1, 4096, -1, 2048, 0    ; triangle wave
giSaw ftgen 0, 0, 8193, 7, 1, 8192, -1      ; sawtooth wave, downward slope

instr 1

iamp = .7
aindex phasor 110    ; read table value at this index
aweightpoint = 0    ; set weightpoint
atabnum1 line 0, p3, 3    ; morph through all tables
atabnum2 = 2    ; set to triangle wave
asig tabmorpha aindex, aweightpoint, atabnum1,atabnum2, giSine, giSquare, giTri, giSaw
asig = asig*iamp
outs asig, asig

endin
</CsInstruments>
<CsScore>

i1 0 10
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*table*, *tabmorph*, *tabmorphi*, *tabmorphak*, *ftmorf*,

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# tabmorphak

tabmorphak — Permet le fondu enchaîné entre un ensemble de tables au taux audio avec interpolation.

## Description

*tabmorphak* permet le fondu enchaîné entre un ensemble de tables de la même taille au moyen d'une moyenne pondérée entre deux tables sélectionnées dans l'ensemble.

## Syntaxe

```
aout tabmorphak aindex, kweightpoint, ktabnum1, ktabnum2, \  
    ifn1, ifn2 [, ifn3, ifn4, ... ifnN]
```

## Initialisation

*ifn1*, *ifn2*, *ifn3*, *ifn4*, ..., *ifnN* -- numéros des tables de fonction. C'est un ensemble de tables choisies par l'utilisateur pour le fondu enchaîné. Toutes les tables doivent avoir la même longueur. Il faut être conscient que seulement deux de ces tables peuvent être choisies en même temps pour le fondu enchaîné. Comme il est possible d'utiliser des nombres non-entiers pour les arguments *ktabnum1* et *ktabnum2*, le fondu enchaîné est le résultat d'une interpolation entre des tables adjacentes consécutives de l'ensemble.

## Exécution

*aout* -- la valeur retournée pour l'indice *aindex*, résultant du fondu enchaîné de deux tables (voir ci-dessous).

*aindex* -- indice principal de la table résultant du fondu enchaîné. L'intervalle va de 0 à la longueur de la table (exclue).

*kweightpoint* -- le poids de l'influence d'une paire de tables sélectionnées dans le fondu enchaîné. Cet argument est compris entre 0 et 1. 0 provoque la sortie de la première table inchangée, 1 provoque la sortie de la seconde table de la paire inchangée. Toutes les valeurs intermédiaires entre 0 et 1 déterminent la gradation du fondu enchaîné entre les deux tables de la paire.

*ktabnum1* -- la première table choisie pour le fondu enchaîné. Ce nombre n'exprime pas directement le numéro de la table mais la position de celle-ci dans la séquence de l'ensemble (de 0 à N-1). Si ce nombre est entier, la table correspondante est choisie inchangée. S'il contient une partie fractionnaire, alors une interpolation avec la table adjacente suivante a lieu.

*ktabnum2* -- la deuxième table choisie pour le fondu enchaîné. Ce nombre n'exprime pas directement le numéro de la table mais la position de celle-ci dans la séquence de l'ensemble (de 0 à N-1). Si ce nombre est entier, la table correspondante est choisie inchangée. S'il contient une partie fractionnaire, alors une interpolation avec la table adjacente suivante a lieu.

La famille d'opcodes *tabmorphak* est semblable à la famille *table*, mais elle permet un fondu enchaîné entre deux tables choisies dans un ensemble de tables. D'abord, l'utilisateur doit fournir un ensemble de tables d'égale longueur (*ifn1*, *ifn2* [, *ifn3*, *ifn4*, ..., *ifnN*]). Ensuite, il peut choisir une paire de tables dans l'ensemble afin d'effectuer le fondu enchaîné : *ktabnum1* et *ktabnum2* reçoivent des nombres (0 représente la première table dans l'ensemble, 1 la seconde, 2 la troisième et ainsi de suite). Puis il détermine le fondu

enchaîné entre les deux tables choisies, avec le paramètre *kweightpoint*. Après cela, la table résultante peut être indexée avec le paramètre *aindex* comme pour un opcode *table* normal. Si la valeur de ce paramètre dépasse la longueur de table (qui doit être la même pour toutes les tables), elle est repliée cycliquement.

*tabmorphak* travaille au taux-a, mais *kweightpoint*, *ktabnum1* et *ktabnum2* travaillent au taux-k, ce qui le rend plus efficace que *tabmorpha*, car il y a moins de calculs. A part le taux de ces trois arguments, il est identique à *tabmorpha*.

## Exemples

Voici un exemple de l'opcode *tabmorphak*. Il utilise le fichier *tabmorphak.csd* [examples/tabmorphak.csd].

### Exemple 1011. Exemple de l'opcode *tabmorphak*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tabmorphak.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 8193, 10, 1      ; sine wave
giSquare ftgen 0, 0, 8193, 7, 1, 4096, 1, 0, -1, 4096, -1    ; square wave
giTri ftgen 0, 0, 8193, 7, 0, 2048, 1, 4096, -1, 2048, 0    ; triangle wave
giSaw ftgen 0, 0, 8193, 7, 1, 8192, -1      ; sawtooth wave, downward slope

instr 1

iamp = .7
aindex phasor 110      ; read table value at this index
kweightpoint expon 0.001, p3, 1 ; using the weightpoint to morph between two tables exponentially
ktabnum1 = p4          ; first wave, it morphs to
ktabnum2 = p5          ; the second wave
asig tabmorphak aindex, kweightpoint, ktabnum1,ktabnum2, giSine, giSquare, giTri, giSaw
asig = asig*iamp
outs asig, asig

endin
</CsInstruments>
<CsScore>

i1 0 5 0 1 ;from sine to square wave
i1 6 5 2 3 ;from triangle to saw
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*table*, *tabmorph*, *tabmorphi*, *tabmorpha*, *ftmorph*,

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# tabmorphi

tabmorphi — Permet le fondu enchaîné entre un ensemble de tables avec interpolation.

## Description

*tabmorphi* permet le fondu enchaîné entre un ensemble de tables de la même taille au moyen d'une moyenne pondérée entre deux tables sélectionnées dans l'ensemble.

## Syntaxe

```
kout tabmorphi kindex, kweightpoint, ktabnum1, ktabnum2, \  
    ifn1, ifn2 [, ifn3, ifn4, ... ifnN]
```

## Initialisation

*ifn1, ifn2 [, ifn3, ifn4, ..., ifnN]* -- numéros des tables de fonction. C'est un ensemble de tables choisies par l'utilisateur pour le fondu enchaîné. Toutes les tables doivent avoir la même longueur. Il faut être conscient que seulement deux de ces tables peuvent être choisies en même temps pour le fondu enchaîné. Comme il est possible d'utiliser des nombres non-entiers pour les arguments *ktabnum1* et *ktabnum2*, le fondu enchaîné est le résultat d'une interpolation entre des tables adjacentes consécutives de l'ensemble.

## Exécution

*kout* -- la valeur retournée pour l'indice *kindex*, résultant du fondu enchaîné de deux tables (voir ci-dessous).

*kindex* -- indice principal de la table résultant du fondu enchaîné. L'intervalle va de 0 à la longueur de la table (exclue).

*kweightpoint* -- le poids de l'influence d'une paire de tables sélectionnées dans le fondu enchaîné. Cet argument est compris entre 0 et 1. 0 provoque la sortie de la première table inchangée, 1 provoque la sortie de la seconde table de la paire inchangée. Toutes les valeurs intermédiaires entre 0 et 1 déterminent la gradation du fondu enchaîné entre les deux tables de la paire.

*ktabnum1* -- la première table choisie pour le fondu enchaîné. Ce nombre n'exprime pas directement le numéro de la table mais la position de celle-ci dans la séquence de l'ensemble (de 0 à N-1). Si ce nombre est entier, la table correspondante est choisie inchangée. S'il contient une partie fractionnaire, alors une interpolation avec la table adjacente suivante a lieu.

*ktabnum2* -- la deuxième table choisie pour le fondu enchaîné. Ce nombre n'exprime pas directement le numéro de la table mais la position de celle-ci dans la séquence de l'ensemble (de 0 à N-1). Si ce nombre est entier, la table correspondante est choisie inchangée. S'il contient une partie fractionnaire, alors une interpolation avec la table adjacente suivante a lieu.

La famille d'opcodes *tabmorphi* est semblable à la famille *table*, mais elle permet un fondu enchaîné entre deux tables choisies dans un ensemble de tables. D'abord, l'utilisateur doit fournir un ensemble de tables d'égale longueur (*ifn1, ifn2 [, ifn3, ifn4, ..., ifnN]*). Ensuite, il peut choisir une paire de tables dans l'ensemble afin d'effectuer le fondu enchaîné : *ktabnum1* et *ktabnum2* reçoivent des nombres (0 représente la première table dans l'ensemble, 1 la seconde, 2 la troisième et ainsi de suite). Puis il détermine le fondu enchaîné entre les deux tables choisies, avec le paramètre *kweightpoint*. Après cela, la table résultante peut être indexée avec le paramètre *kindex* comme pour un opcode *table* normal. Si la valeur de ce paramètre dépasse la longueur de table (qui doit être la même pour toutes les tables), elle est repliée cycliquement.

*tabmorphi* est identique à *tabmorph*, mais il effectue une interpolation linéaire pour les valeurs non entières de *kindex*, à l'instar de *tablei*.

## Exemples

Voici un exemple de l'opcode *tabmorphi*. Il utilise le fichier *tabmorphi.csd* [examples/tabmorphi.csd].

### Exemple 1012. Exemple de l'opcode *tabmorphi*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tabmorphi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine    ftgen 0, 0, 8193, 10, 1      ;sine wave
giSquare  ftgen 0, 0, 8193, 7, 1, 4096, 1, 0, -1, 4096, -1 ;square wave
giTri     ftgen 0, 0, 8193, 7, 0, 2048, 1, 4096, -1, 2048, 0 ;triangle wave
giSaw     ftgen 0, 0, 8193, 7, 1, 8192, -1      ;sawtooth wave, downward slope

instr     1

iamp      = .7
kindex    phasor 440      ;read table value at this index
kindex = kindex*8192      ;for all 8192 index points
kweightpoint = 0.5      ;set weightpoint
ktabnum1  line 0, p3, 3    ;morph through all tables
ktabnum2  = 2      ;set to triangle wave
ksig      tabmorphi kindex, kweightpoint, ktabnum1, ktabnum2, giSine, giSquare, giTri, giSaw
ksig = ksig*iamp
asig      interp ksig      ;convert to audio signal
outs      asig, asig

endin
</CsInstruments>
<CsScore>

i1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*table*, *tabmorph*, *tabmorpha*, *tabmorphak*, *ftmorf*,

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans version 5.06



# tabplay

tabplay — Restitution de signaux de contrôle.

## Description

Restitution de signaux au taux de contrôle sur la base d'une temporisation à déclenchement.

## Syntaxe

```
tabplay ktrig, knumtics, kfn, kout1 [,kout2,..., koutN]
```

## Exécution

*ktrig* -- commence à jouer s'il est différent de zéro.

*knuntics* -- stoppe l'enregistrement ou réinitialise à zéro le pointeur de lecture si le nombre de tics défini par cet argument est atteint.

*kfn* -- table dans laquelle les signaux de taux-k sont enregistrés.

*kout1,...,koutN* -- signaux restitués en sortie.

Les opcodes *tabplay* et *tabrec* permettent d'enregistrer/restituer des signaux de contrôle sur la base d'une temporisation à déclenchement.

*tabplay* restitue un groupe de signaux de taux-k, préalablement enregistrés par *tabrec* dans une table. Chaque fois que l'argument *ktrig* est activé, un compteur interne est augmenté d'une unité. Après que *knuntics* impulsions de déclenchement aient été reçues par l'argument *ktrig*, le compteur interne est mis à zéro et la restitution recommence depuis le début, en boucle.

Ces opcodes peuvent être utilisés comme une sorte de mémoire à "moyen-terme" qui se "souvient" des signaux générés. Une telle mémoire peut être utilisée pour fournir à de la musique générative une structure de composition itérative cohérente.

## Exemples

Pour un exemple d'utilisation voir l'exemple dans l'opcode *tabrec*.

## Voir aussi

*tabrec*

## Crédits

Ecrit par Gabriel Maldonado.

# tabrec

tabrec — Enregistrement de signaux de contrôle.

## Description

Enregistre des signaux au taux de contrôle sur la base d'une temporisation à déclenchement.

## Syntaxe

```
tabrec   ktrig_start, ktrig_stop, knumtics, kfn, kin1 [,kin2,...,kinN]
```

## Exécution

*ktrig\_start* -- commence à enregistrer s'il est différent de zéro.

*ktrig\_stop* -- stoppe l'enregistrement lorsque *knumtics* impulsions de déclenchement ont été reçues par cet argument d'entrée.

*knumtics* -- stoppe l'enregistrement ou réinitialise à zéro le pointeur de lecture si le nombre de tics défini par cet argument est atteint.

*kfn* -- table dans laquelle les signaux de taux-k sont enregistrés.

*kin1,...,kinN* -- signaux d'entrée à enregistrer.

Les opcodes *tabrec* et *tabplay* permettent d'enregistrer/restituer des signaux de contrôle sur la base d'une temporisation à déclenchement.

L'opcode *tabrec* enregistre un groupe de signaux de taux-k en les mémorisant dans la table *kfn*. Chaque fois que *ktrig\_start* est activé, *tabrec* remet à zéro le pointeur de la table et commence à enregistrer. La phase d'enregistrement s'arrête après que *knumtics* impulsions de déclenchement aient été reçues par l'argument *ktrig\_stop*.

Ces opcodes peuvent être utilisés comme une sorte de mémoire à "moyen-terme" qui se "souvient" des signaux générés. Une telle mémoire peut être utilisée pour fournir à de la musique générative une structure de composition itérative cohérente.

## Exemples

voici un exemple de l'opcode tabrec. Il utilise le fichier *oscil.csd* [examples/tabrec.csd].

### Exemple 1013. Exemple de l'opcode tabrec.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
```

```
; For Non-realtime ouput leave only the line below:
; -o oscil.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gifn ftgen 0,0,1049576,2,0

; record control signals
instr 1
ktrig_start init 1
koct rspline 7,10,1,2
kpan rspline 7,10,0.1,0.9
ktrig_stop = 1
knumtics = kr*p3
tabrec ktrig_start,ktrig_stop,ksmps,ksmps,ksmps,ksmps,ksmps
ktrig_start = 0
endin

; play control signals
instr 2
koc, kpan init 0
ktrig init 1
ksmps = kr*p3
tabplay ktrig,ksmps,ksmps,ksmps,ksmps,ksmps,ksmps

ktrig = 0

asig poscil 0.1, cpsoct(koc)
aL,aR pan2 asig,kpan
outs aL,aR
endin

</CsInstruments>

<CsScore>
i1 0 10
i2 2 10
i2 4 10
e

</CsScore>

</CsoundSynthesizer>
```

## Voir aussi

*tabplay*

## Crédits

Ecrit par Gabriel Maldonado.

Exemple écrit par Iain McCurdy

# tabrowlin

tabrowlin — Copie une ligne d'une ftable dans une autre avec interpolation entre les lignes.

## Description

Opcode du greffon beosc.

Cette opcode suppose l'utilisation d'une ftable, qui est un simple tableau 1D, contenant une matrice 2D avec une longueur de ligne donnée. En supposant qu'une telle matrice 2D contient plusieurs lignes de flots échantillonnés (par exemple les amplitudes d'un ensemble d'oscillateurs échantillonnés à intervalles réguliers), cet opcode extrait une ligne (ou un bout d'une ligne) de ces données avec interpolation linéaire entre lignes adjacentes (si la ligne n'est pas un nombre rond) et place le résultat dans une autre ftable.

## Syntaxe

```
tabrowlin krow, ifnsrc, ifndest, inumcols [, ioffset, istart, iend, istep ]
```

## Initialisation

*ifnsrc* -- Le numéro de la table source.

*ifndest* -- Le numéro de la table de destination. Cette table doit pouvoir contenir une ligne de données.

*inumcols* -- Le nombre de colonnes croisées par une ligne dans la table source.

*ioffset* -- Le décalage de la position à laquelle commencent les données 2D (utilisé pour ignorer un possible en-tête). 0 par défaut.

*istart* -- La position du début de la lecture (par rapport à la ligne). 0 par défaut.

*iend* -- La position (non inclusive) de la fin de la lecture (ne doit pas dépasser *inumcols*).

*istep* -- Pas de lecture dans la ligne (1 par défaut).

## Exécution

*krow* -- La ligne à lire (peut être un nombre fractionnaire, auquel cas il y aura interpolation linéaire avec la ligne suivante).

## Exemples

Voici un exemple de l'opcode tabrowlin. Il utilise le fichier *tabrowlin.csd* [examples/tabrowlin.csd].

### Exemple 1014. Exemple de l'opcode tabrowlin.

```
<CsoundSynthesizer>
<CsOptions>
-odac      ;;;realtime audio out
</CsOptions>
<CsInstruments>

/*

This is the example file for tabrowlin
```

```
tabrowlin
=====
```

*This opcode assumes the use of a table, which is a simple 1D array, to hold a 2D matrix with a given row length. Assuming such a 2D table containing multiple rows of sampled streams (for instance, the amplitudes of a set of oscillators, sampled at a regular interval), this opcode can extract one row of that data with linear interpolation between adjacent rows (if row is not a whole number) and place the result in another table*

```
Syntax
=====
```

```
tabrowlin krow, ifnsrc, ifndest, inumcols, ioffset=0, istart=0, iend=0, istep=1
```

```
krow      : the row to read (can be a fractional number, in which case interpolation
            with the next row is performed)
ifnsrc     : index of the source table
ifndest    : index of the dest table
inumcols   : the number of columns a row has, in the source table
ioffset    : an offset to where the data starts (used to skip a header, if present)
istart     : start index to read from the row (not the start index of the table)
iend       : end index to read from the row (not inclusive)
istep      : step used to read the along the row
```

*If reading out of bounds a PerformanceError will be raised. Because we interpolate between rows, the last row that can be read is*

```
maxrow = (ftlen(ifnsrc)-ioffset)/inumcols - 2
```

```
*/
```

```
sr = 44100
ksmps = 128
nchnls = 1
0dbfs = 1
```

```
instr 1
; just a simple test of the bare functionality
; generate a 4x3 table
isource ftgentmp 0, 0, -12, -2, \
    0, 1, 2, 3, \
    10, 11, 12, 13, \
    20, 21, 22, 23
; create an empty table able to hold one row (4 elements)
idest ftgentmp 0, 0, -4, -2, 0
print ftlen(isource)
; we exceed the max. row to show what happens (the row is clipped
; to the max row possible and a message is printed to show the error)
krow linseg 0, p3, 2.05
printk2 krow, 20
tabrowlin krow, isource, idest, 4
ftprint idest, -1
endin
```

```
</CsInstruments>
<CsScore>
i 1 0 2
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*getrowlin, fislice, slicearray, copyf2array, tab2array, tablecopy, tableicopy,*

## Crédits

Auteur : Eduardo Moguillansky 2018

Nouveau greffon dans la version 6.12

# tabsum

tabsum — Addition des valeurs dans un intervalle d'une table.

## Description

Fait la somme des valeurs d'une ftable dans un intervalle contigu.

## Syntaxe

```
kr tabsum ifn[[, kmin] [, kmax]]
```

## Initialisation

*ifn* -- numéro de la table.

## Exécution

*kr* -- signal retourné.

*kmin*, *kmax* -- intervalle de la table à sommer. S'il est omis ou si les arguments sont nuls, il couvre par défaut les valeurs allant de 0 à la longueur de la table.

## Exemples

Voici un exemple de l'opcode tabsum. Il utilise le fichier *tabsum.csd* [examples/tabsum.csd].

### Exemple 1015. Exemple de l'opcode tabsum.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;;realtime audio out
; For Non-realtime ouput leave only the line below:
; -o tab.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 2205
nchnls = 1
0dbfs = 1

instr 1      ;;; Give a value to the increment
  kmax = 256
  knorm tabsum 1, 0, kmax
  gkinc = knorm/10
endin

instr 2
  kmax = 256
```

```

kx = rnd(kmax)
krnd tabsum 1, 0, kx
knorm tabsum 1, 0, kmax
kvar = krnd / knorm      ;;; now n [0,1] range
asig oscil kvar, p4, 2
    out asig
;;; Make randomness give 1 more often
kc tab 0, 1
    tablew kc+gkinc, 0, 1
endin
</CsInstruments>

<CsScore>
f1 0 256 21 1
f2 0 4096 10 1
i1 0 0.1
i2 0.1 3 440
e

</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

Vectorial opcodes

## Crédits

Auteur : John ffitch  
 Codemist Ltd  
 2009

Nouveau dans la version 5.11



# tab2array

tab2array — Copie un bout d'une ftable dans un tableau.

## Description

Opcodes du greffon emugens.

L'opcode *tab2array* prend une ftable et en copie un bout dans un tableau, en l'allouant ou en changeant sa taille si nécessaire.

## Syntaxe

```
kout[] tab2array ifn [, kstart, kend, kstep ]  
iout[] tab2array ifn [, istart, iend, istep ]
```

## Initialisation

*ifn* -- Le numéro de la table source.

## Exécution

*kstart* / *istart* -- La position à partir de laquelle copier. 0 par défaut.

*kend* / *iend* -- La position à partir de laquelle la copie est stoppée. Elle n'est PAS inclusive. 0 indique de copier jusqu'à la fin de la table. Par défaut = la fin de la table.

*kstep* / *istep* -- Le nombre d'éléments à ignorer. 1 par défaut.

## Exemples

Voici une exemple de l'opcode tab2array. Il utilise le fichier *tab2array.csd* [examples/tab2array.csd].

### Exemple 1016. Exemple de l'opcode tab2array.

```
<CsoundSynthesizer>  
<CsOptions>  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 128  
nchnls = 2  
0dbfs = 1.0  
  
; Example file tab2array  
/*  
  
tab2array: copy a slice of a table to an array  
  
kout[] tab2array ifn, kstart=0, kend=0, kstep=1
```

```
iout[] tab2array ifn, istart=0, iend=0, istep=1

ifn: the table index to copy from
start: the index to start copying from
end: the end index to stop copying. This is NOT inclusive. 0=end of table
step: how many elements to skip

*/

instr 1
  ifn ftgentmp 0,0,-13,-2, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
  ; copy everything at i-time, similar to copyf2array, but no need
  ; to predefine the array
  islice[] tab2array ifn
  printarray islice, "", "islice"

  ; copy the slice [1::2] to an array, at k-time
  kslice[] tab2array ifn, 1, 0, 2
  printarray kslice, 1, "", "kslice"

  ; copy into a predefined array. If the number of elements to copy
  ; exceeds the capacity of the array, the array is enlarged
  kxs[] init 3
  kxs tab2array ifn, 0, 10
  printarray kxs, 1, "", "kxs"
  turnoff
endin

</CsInstruments>
<CsScore>
i 1 0 0.1

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*copyf2array, ftslice slicearray*

## Crédits

Auteur : Eduardo Moguillansky 2018

Nouveau greffon dans la version 6.12

# tab2pvs

`tab2pvs` — Copie des donnée spectrales depuis des tableaux de taux-k (ou variables-t). Aussi appelé `pvs-fromarray`.

## Description

Copie une trame pvs depuis une variable-t ou un tableau de taux-k. Pour le moment, seuls les formats AMP+FREQ et AMP+PHASE sont produits. Cet opcode nécessite que le type-t soit défini, si bien qu'il ne fonctionne qu'avec le nouveau parseur basé sur bison/flex.

## Syntaxe

```
fsig tab2pvs tvar|karr[[,ihopsize, iwinsize, iwintype]
fsig tab2pvs kmags[], kfregs[[,ihopsize, iwinsize, iwintype]
```

## Exécution

*tvar* -- tableau de taux-k (ou variable-t) contenant l'entrée. Elle est produite à chaque période-k, mais peut ne pas contenir une nouvelle trame, les trames pvs étant produites selon leur propre rythme qui est indépendant de *kr*. La taille de ce vecteur détermine la taille de TFR,  $N = \text{taille} - 2$ .

*kmags[]*, *kfregs[]* -- tableaux de taux-k contenant les valeurs de magnitude et de fréquence en entrée. La sortie est générée à chaque période-k, mais elle peut ne pas contenir de nouvelle trame, les trames de pvs étant produites selon leur propre taux qui est indépendant de *kr*. Ces vecteurs doivent avoir la même taille qui détermine la taille de TFR,  $N = \text{taille} - 2$ .

*fsig* -- fsig de sortie recevant la copie.

*iolap* -- taille de recouvrement de l'analyse, par défaut *isize*/4.

*iwinsize* -- taille de la fenêtre d'analyse, par défaut *isize*.

*iwintype* -- type de la fenêtre d'analyse, 1 par défaut, Hanning.

## Exemples

### Exemple 1017. Exemple

```
karr[] init      1026
...
fsig1 tab2pvs karr
aout  pvsynth fsig1
```

## Crédits

Auteur : Victor Lazzarini  
Octobre 2011

Nouveau greffon dans la version 5.14

Octobre 2011.

# tambourine

tambourine — Modèle semi-physique d'un son de tambourin.

## Description

*tambourine* est un modèle semi-physique d'un son de tambourin. Il fait partie des opcodes de percussion de PhISEM. PhISEM (Physically Informed Stochastic Event Modeling) est une approche algorithmique pour simuler les collisions de multiples objets indépendants produisant des sons.

## Syntaxe

```
ares tambourine kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \  
    [, ifreq1] [, ifreq2]
```

## Initialisation

*idettack* -- période de temps durant laquelle tous les sons sont stoppés.

*inum* (facultatif) -- le nombre de perles, de dents, de cloches, de tambourins, etc. S'il vaut zéro, il prend la valeur par défaut de 32.

*idamp* (facultatif) -- le facteur d'amortissement, intervenant dans l'équation :

$\text{damping\_amount} = 0,9985 + (\text{idamp} * 0,002)$

La valeur par défaut de *damping\_amount* est 0,9985 ce qui signifie que la valeur par défaut de *idamp* est 0. Le maximum de *damping\_amount* est 1,0 (pas d'amortissement). La valeur maximale de *idamp* est donc 0,75.

L'intervalle recommandé pour *idamp* se situe d'habitude sous les 75% de la valeur maximale.

*imaxshake* (facultatif, 0 par défaut) -- quantité d'énergie à réinjecter dans le système. La valeur doit être comprise entre 0 et 1.

*ifreq* (facultatif) -- la fréquence de résonance principale. La valeur par défaut est 2300.

*ifreq1* (facultatif) -- la première fréquence de résonance. La valeur par défaut est 5600.

*ifreq2* (facultatif) -- la deuxième fréquence de résonance. La valeur par défaut est 8100.

## Exécution

*kamp* -- Amplitude de la sortie. Note : comme ces instruments sont stochastiques, ce n'est qu'une approximation.

## Exemples

Voici un exemple de l'opcode *tambourine*. Il utilise le fichier *tambourine.csd* [examples/tambourine.csd].

### Exemple 1018. Exemple de l'opcode *tambourine*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tambourine.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

idamp = p4
asig  tambourine .8, 0.01, 30, idamp, 0.4
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 .2 0
i 1 + .2 >
i 1 + 1 .7
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*bamboo, dripwater, guiro, sleighbells*

## Crédits

Auteur : Perry Cook, fait partie de PhISEM (Physically Informed Stochastic Event Modeling)

Adapté par John ffitich

Université de Bath, Codemist Ltd.

Bath, UK

Nouveau dans la version 4.07 de Csound

Notes ajoutées par Rasmus Ekman en mai 2002.

# tan

tan — Calcule une fonction tangente.

## Description

Retourne tangente de  $x$  ( $x$  en radians).

## Syntaxe

`tan(x)` (pas de restriction de taux)

`tan(k/i[])` (k- ou i-tableau)

## Exemples

Voici un exemple de l'opcode tan. Il utilise le fichier *tan.csd* [examples/tan.csd].

### Exemple 1019. Exemple de l'opcode tan.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tan.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 25
  i1 = tan(irad)

  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = -0.134
```

## Voir aussi

*cos*, *cosh*, *cosinv*, *sin*, *sinh*, *sininv*, *tan*, *taninv*

## Crédits

Ecrit par John ffitch.

Nouveau dans la version 3.47

Exemple écrit par Kevin Conder.



# tanh

tanh — Calcule une fonction tangente hyperbolique.

## Description

Retourne tangente hyperbolique de  $x$ .

## Syntaxe

**tanh**( $x$ ) (pas de restriction de taux)

**tanh**( $k/i[]$ ) ( $k$ - ou  $i$ -tableau)

## Exemples

Voici un exemple de l'opcode tanh. Il utilise le fichier *tanh.csd* [examples/tanh.csd].

### Exemple 1020. Exemple de l'opcode tanh.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tanh.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

; Instrument #1.
instr 1
  asig1 vco 1, 440, 2, 0.4, 1
  asig2 vco 1, 800, 3, 0.5, 1
  asig = asig1+asig2      ;; will go out of range
                        out tanh(asig)      ;; but tanh is a limiter
endin

</CsInstruments>
<CsScore>

f1 0 65536 10 1
; Play Instrument #1 for one second.
i 1 0 1
e
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*cos, cosh, cosinv, sin, sinh, sininv, tan, taninv*

## Crédits

Auteur : John ffitch

Nouveau dans la version 3.47

# taninv

taninv — Calcule une fonction arctangente.

## Description

Retourne arctangente de  $x$  ( $x$  en radians).

## Syntaxe

`taninv(x)` (pas de restriction de taux)

`taninv(k/i[])` (k- ou i-tableau)

## Exemples

Voici une exemple de l'opcode taninv. Il utilise le fichier *taninv.csd* [examples/taninv.csd].

### Exemple 1021. Exemple de l'opcode taninv.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o taninv.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  irad = 0.5
  i1 = taninv(irad)

  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra cette ligne :

```
instr 1:  i1 = 0.464
```

## Voir aussi

*cos*, *cosh*, *cosinv*, *sin*, *sinh*, *sininv*, *tan*, *tanh*, *taninv2*

## Crédits

Auteur : John ffitch

Nouveau dans la version 3.48

Exemple écrit par Kevin Conder.

# taninv2

taninv2 — Retourne une tangente inverse (arctangente).

## Description

Retourne arctangente de  $iy/ix$ ,  $ky/kx$ , ou  $ay/ax$ .

## Syntaxe

```
ares taninv2 ay, ax
ires taninv2 iy, ix
kres taninv2 ky, kx
```

Retourne arctangente de  $iy/ix$ ,  $ky/kx$ , ou  $ay/ax$ . Si  $y$  vaut zéro, *taninv2* retourne zéro quelque soit la valeur de  $x$ . Si  $x$  vaut zéro, la valeur de retour est :

- $\pi/2$ , si  $y$  est positif.
- $-\pi/2$ , si  $y$  est négatif.
- 0, si  $y$  vaut 0.

## Initialisation

*iy*, *ix* -- valeurs à transformer

## Exécution

*ky*, *kx* -- signaux de taux de contrôle à transformer

*ay*, *ax* -- signaux de taux audio à transformer

## Exemples

Voici un exemple de l'opcode taninv2. Il utilise le fichier *taninv2.csd* [examples/taninv2.csd].

### Exemple 1022. Exemple de l'opcode taninv2.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o taninv2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  ; Returns the arctangent for 1/2.
  i1 taninv2 1, 2

  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra cette ligne :

```
instr 1: i1 = 0.464
```

## Voir aussi

*taninv*

## Crédits

Auteur : John ffitch  
 Université de Bath/Codemist Ltd.  
 Bath, UK  
 Avril 1998

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.48 de Csound

Corrigé en mai 2002, grâce à Istvan Varga.

# tbvcf

tbvcf — Modélise quelques caractéristiques du filtre contrôlé en tension du TB303 de Roland.

## Description

Cet opcode est un essai de modélisation de quelques caractéristiques du filtre contrôlé en tension du TB303 de Roland. On utilise la méthode d'Euler pour obtenir une approximation du système, plutôt que les méthodes traditionnelles des filtres. La fréquence de coupure, Q, et la distorsion sont tous interdépendants. Des méthodes empiriques ont été utilisées pour essayer de les séparer, ce qui a pour effet de rendre la fréquence approximative. La résolution future de certains problèmes de cet opcode pourrait rendre inopérants les orchestres existants qui utilisent cette version de *tbvcf*.

## Syntaxe

```
ares tbvcf asig, xfco, xres, kdist, kasym [ , iskip]
```

## Initialisation

*iskip* (facultatif, 0 par défaut) -- s'il est non nul, l'initialisation du filtre est ignorée. (Nouveau dans les versions 4.23f13 et 5.0 de Csound).

## Exécution

*asig* -- signal d'entrée. Doit être normalisé à  $\pm 1$ .

*xfco* -- fréquence de coupure du filtre. L'intervalle optimal va de 10000 à 1500. Les valeurs inférieures à 1000 peuvent poser problème.

*xres* -- résonance ou Q. Typiquement compris entre 0 et 2.

*kdist* -- quantité de distorsion. Une valeur typique est 2. Si *kdist* s'écarte de 2 de manière significative, il peut y avoir des interactions bizarres entre *xfco* et *xres*.

*kasym* -- asymétrie de la résonance. Typiquement comprise entre 0 et 1.

## Exemples

Voici un exemple de l'opcode *tbvcf*. Il utilise le fichier *tbvcf.csd* [examples/tbvcf.csd].

### Exemple 1023. Exemple de l'opcode *tbvcf*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tbvcf.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

;-----
; TBVCF Test
; Coded by Hans Mikelson December, 2000
;-----

  sr = 44100 ; Sample rate
  ksmps = 10 ; Samples/Kontrol period
  nchnls = 2 ; Normal stereo
  odbfs = 1

  instr 10

    idur = p3 ; Duration
    iamp = p4 ; Amplitude
    ifqc = cpspch(p5) ; Pitch to frequency
    ipanl = sqrt(p6) ; Pan left
    ipanr = sqrt(1-p6) ; Pan right
    iq = p7
    idist = p8
    iasym = p9

    kdclck linseg 0, .002, 0.9, idur-.004, 0.9, .002, 0 ; Declick envelope
    kfco expseg 10000, idur, 1000 ; Frequency envelope
    ax vco 1, ifqc, 2, 0.5 ; Square wave
    ay tbvcf ax, kfco, iq, idist, iasym ; TB-VCF
    ay buthlp ay/1, 100 ; Hi-pass

    outs ay*iamp*ipanl*kdclck, ay*iamp*ipanr*kdclck
    endin

</CsInstruments>
<CsScore>

f1 0 65536 10 1

; TeeBee Test
; Sta Dur Amp Pitch Pan Q Dist1 Asym
i10 0 0.2 0.5 7.00 .5 0.0 2.0 0.0
i10 0.3 0.2 0.5 7.00 .5 0.8 2.0 0.0
i10 0.6 0.2 0.5 7.00 .5 1.6 2.0 0.0
i10 0.9 0.2 0.5 7.00 .5 1.7 2.0 0.0
i10 1.2 0.2 0.5 7.00 .5 1.8 2.0 0.0
i10 1.8 0.2 0.5 7.00 .5 0.0 2.0 0.25
i10 2.1 0.2 0.5 7.00 .5 0.8 2.0 0.25
i10 2.4 0.2 0.5 7.00 .5 1.6 2.0 0.25
i10 2.7 0.2 0.5 7.00 .5 1.8 2.0 0.25
i10 3.0 0.2 0.5 7.00 .5 1.9 2.0 0.25
i10 3.3 0.2 0.5 7.00 .5 2.0 2.0 0.25
i10 3.6 0.2 0.5 7.00 .5 0.0 2.0 0.5
i10 3.9 0.2 0.5 7.00 .5 0.8 2.0 0.5
i10 4.2 0.2 0.5 7.00 .5 1.6 2.0 0.5
i10 4.5 0.2 0.5 7.00 .5 1.8 2.0 0.5
i10 4.8 0.2 0.5 7.00 .5 1.9 2.0 0.5
i10 5.1 0.2 0.5 7.00 .5 2.0 2.0 0.5
i10 5.4 0.2 0.5 7.00 .5 0.0 2.0 0.75
i10 5.7 0.2 0.5 7.00 .5 0.8 2.0 0.75
i10 6.0 0.2 0.5 7.00 .5 1.6 2.0 0.75
i10 6.3 0.2 0.5 7.00 .5 1.8 2.0 0.75
i10 6.6 0.2 0.5 7.00 .5 1.9 2.0 0.75
i10 6.9 0.2 0.5 7.00 .5 2.0 2.0 0.75
i10 7.2 0.2 0.5 7.00 .5 0.0 2.0 1.0
i10 7.5 0.2 0.5 7.00 .5 0.8 2.0 1.0
i10 7.8 0.2 0.5 7.00 .5 1.6 2.0 1.0
i10 8.1 0.2 0.5 7.00 .5 1.8 2.0 1.0
i10 8.4 0.2 0.5 7.00 .5 1.9 2.0 1.0
i10 8.7 0.2 0.5 7.00 .5 2.0 2.0 1.0

```



e

```
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : Hans Mikelson  
Décembre, 2000 -- Janvier, 2001

Nouveau dans Csound 4.10

# tempest

tempest — Estime le tempo de motifs de pulsation dans un signal de contrôle.

## Description

Estime le tempo de motifs de pulsation dans un signal de contrôle.

## Syntaxe

```
ktemp tempest kin, iprd, imindur, imemdur, ihp, ithresh, ihtim, ixfdbak, \  
istartempo, ifn [, idisprd] [, itweek]
```

## Initialisation

*iprd* -- durée entre les analyses (en secondes). Typiquement autour de 0.02 secondes.

*imindur* -- durée minimale (en secondes) pour servir d'unité de tempo. Typiquement autour de 0.2 secondes.

*imemdur* -- durée (en secondes) du tampon de mémoire à court-terme *kin* parcouru pour trouver des motifs périodiques. Typiquement autour de 3 secondes.

*ihp* -- point à mi-puissance (en Hz) d'un filtre passe-bas utilisé pour lisser l'entrée *kin* avant tout autre traitement. Cela tend à supprimer l'activité beaucoup plus rapide. Typiquement 2 Hz.

*ithresh* -- seuil d'intensité autour duquel le signal *kin* filtré est centré et tronqué avant d'être placé dans le tampon à court-terme comme donnée pertinente pour le tempo. Typiquement au niveau du bruit de fond du signal entrant.

*ihtim* -- mi-durée (en secondes) d'un filtre interne masque de précédence qui masque les nouvelles données de *kin* en présence de données récentes plus fortes. Typiquement autour de 0.005 secondes.

*ixfdbak* -- proportion de *valeur anticipée* de cette unité à mélanger avec le signal entrant *kin* avant tout autre traitement. Typiquement autour de 0.3.

*istartempo* -- tempo initial (en pulsations par minute). Typiquement 60.

*ifn* -- numéro de table d'une fonction stockée (dessinée de gauche à droite) par laquelle la mémoire à court-terme est atténuée au court du temps.

*idisprd* (facultatif) -- s'il est différent de zéro, les tampons à court-terme passé et futur sont affichés toutes les *idisprd* secondes (normalement un multiple de *iprd*). La valeur par défaut est 0 (pas d'affichage).

*itweek* (facultatif) -- réglage fin de cette unité afin qu'elle reste stable durant l'analyse d'évènements contrôlés par sa propre sortie. La valeur par défaut est 1 (pas de changement).

## Exécution

*tempest* recherche dans *kin* une périodicité d'amplitude et estime le tempo courant. L'entrée passe d'abord par un filtre passe-bas, puis elle est centrée et tronquée et le résultat est placé dans un tampon de mémoire à court-terme (atténué dans le temps) où il est analysé à la recherche de périodicité, au moyen d'une forme d'autocorrélation. La période, exprimée comme un *tempo* en pulsations par minute, est retournée dans

*ktemp*. La période est aussi utilisée en interne pour prédire les motifs d'amplitude futurs, et ceux-ci sont placés dans un tampon adjacent à celui de l'entrée. On peut afficher périodiquement les deux tampons adjacents et les valeurs prédites facultativement mélangées avec le signal entrant pour simuler les valeurs attendues.

Cette unité est utile pour détecter les caractéristiques métriques d'un signal de taux-k (par exemple la valeur quadratique moyenne d'un signal audio ou la dérivée seconde d'un geste conducteur), avant de l'envoyer à une instruction *tempo*.

## Exemples

Voici un exemple de l'opcode *tempest*. Il utilise les fichiers *tempest.csd* [examples/tempest.csd] et *beats.wav* [examples/beats.wav].

### Exemple 1024. Exemple de l'opcode *tempest*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tempest.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Use the "beats.wav" sound file.
asig soundin "beats.wav"
; Extract the pitch and the envelope.
kcps, krms pitchamdf asig, 150, 500, 200

iprd = 0.01
imindur = 0.1
imemdur = 3
ihp = 1
ithresh = 30
ihtim = 0.005
ixfdbak = 0.05
istartempo = 110
ifn = 1

; Estimate its tempo.
k1 tempest krms, iprd, imindur, imemdur, ihp, ithresh, ihtim, ixfdbak, istartempo, ifn
printk2 k1

out asig
endin

</CsInstruments>
<CsScore>
```

```
; Table #1, a declining line.  
f 1 0 128 16 1 128 1  
  
; Play Instrument #1 for two seconds.  
i 1 0 2  
e  
  
</CsScore>  
</CsoundSynthesizer>
```

Le tempo du fichier audio « beats.wav » est de 120 pulsations par minute. Dans cet exemple, *tempest* imprimera sa meilleure estimation durant la lecture du fichier. Sa sortie contiendra des lignes comme celles-ci :

```
. i1 118.24654  
. i1 121.72949
```

# tempo

tempo — Contrôle le tempo d'une partition non interprétée.

## Description

Contrôle le tempo d'une partition non interprétée.

## Syntaxe

```
tempo ktempo, istartempo
```

## Initialisation

*istartempo* -- tempo initial (en pulsations par minute). Typiquement 60.

## Exécution

*ktempo* -- le tempo auquel la partition sera ajustée.

*tempo* permet de contrôler depuis un orchestre la vitesse d'exécution des événements de partition de Csound. Il n'opère qu'en présence de l'option *-t* de Csound. Quand cette option est positionnée, les événements de partition sont exécutés à partir de leurs paramètres p2 et p3 (pulsation) non interprétés, initialement au tempo donné sur la ligne de commande. Lorsqu'une instruction *tempo* est activée dans n'importe quel instrument (*ktempo* > 0.), le tempo courant est ajusté à *ktempo* pulsations par minute. Il peut y avoir n'importe quel nombre d'instructions *tempo* dans un orchestre, mais il vaut mieux éviter les activations simultanées.

## Exemples

Voici une exemple de l'opcode tempo. Se rappeler qu'il ne fonctionne que si l'on utilise l'option *-t* avec Csound. L'exemple utilise le fichier *tempo.csd* [examples/tempo.csd].

### Exemple 1025. Exemple de l'opcode tempo.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc    -t60 ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tempo.wav -W -t60 ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```

; Instrument #1.
instr 1
  kval tempoval

  printk 0.1, kval

  ; If the fourth p-field is 1, increase the tempo.
  if (p4 == 1) kgoto speedup
    kgoto playit

speedup:
  ; Increase the tempo to 150 beats per minute.
  tempo 150, 60

playit:

  a1 oscil 10000, 440, 1
  out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; p4 = plays at a faster tempo (when p4=1).
; Play Instrument #1 at the normal tempo, repeat 3 times.
r3
i 1 00.00 00.25 0
i 1 00.25 00.25 0
i 1 00.50 00.25 0
i 1 00.75 00.25 0
s

; Play Instrument #1 at a faster tempo, repeat 3 times.
r3
i 1 00.00 00.25 1
i 1 00.25 00.25 0
i 1 00.50 00.25 0
i 1 00.75 00.25 0
s

e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*tempoval*

## Crédits

Exemple écrit par Kevin Conder.

# temposcal

temposcal — Traitement par vocodeur à verrouillage de phase avec détection/traitement d'attaque et "pondération du tempo".

## Description

*temposcal* implémente un traitement par vocodeur à verrouillage de phase utilisant des tables de fonction qui contiennent des sources sonores échantillonnées avec *GEN01*. Il acceptera ainsi des tables à allocation différée.

Cet opcode permet une pondération indépendante du temps et de la fréquence. Le temps progresse en interne mais il est contrôlé par un paramètre de pondération du tempo ; lorsqu'une attaque est détectée, l'échelonnement du temps est momentanément interrompue pour éviter le brouillage des attaques. La qualité de l'effet est généralement améliorée avec le verrouillage de phase activé.

*temposcal* pondère aussi la hauteur, indépendamment de la fréquence, en utilisant un facteur de transposition (taux-k).

## Syntaxe

```
asig temposcal ktimescal, kamp, kpitch, ktab, klock [,ifftsize, idecim, ithresh]
```

## Initialisation

*ifftsize* -- taille de la TFR (puissance de deux), vaut par défaut 2048.

*idecim* -- décimation, 4 par défaut (ce qui signifie que *hopsiz*e = *fftsiz*e/4).

*idbthresh* -- seuil basé sur le rapport du spectre de puissance en dB entre deux fenêtres successives. Un rapport détecté au-dessus du seuil annule momentanément l'échelonnement du temps, pour éviter le brouillage des attaques (vaut 1 par défaut).

## Exécution

*ktimescal* -- rapport d'échelonnement du temps, < 1 étirement, > 1 contraction.

*kamp* -- pondération de l'amplitude.

*kpitch* -- pondération de la hauteur de grain (1=hauteur normale, < 1 plus grave, > 1 plus aigu ; négatif, à l'envers).

*klock* -- 0 ou 1, pour désactiver/activer le verrouillage de phase.

*ktab* -- table de fonction du signal source. Les tables à allocation différée (voir *GEN01*) sont acceptées, mais l'opcode attend une source mono. On peut changer de table au taux-k.

## Exemples

Voici un exemple de l'opcode temposcal. Il utilise le fichier *temposcal.csd* [examples/temposcal.csd].

## Exemple 1026. Exemple de l'opcode temposcal.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o temposcal.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ilock = p4
itab = 1
ipitch = 1
iamp = 0.8
ktime linseg 0.3, p3/2, 0.8, p3/2, 0.3
asig temposcal ktime, iamp, ipitch, itab, ilock
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 0 1 "fox.wav" 0 4 0

i 1 0 3.8 0 ;no locking
i 1 4 3.8 1 ;locking
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
Février 2010

Nouveau greffon dans la version 5.13

Février 2005.



# tempoval

tempoval — Lit la valeur courante du tempo.

## Description

Lit la valeur courante du tempo.

## Syntaxe

```
kres tempoval
```

## Exécution

*kres* -- la valeur du tempo. Si l'on utilise une valeur positive avec l'option *-t* de la ligne de commande, *tempoval* retourne le pourcentage d'accroissement/diminution par rapport au tempo original de 60 pulsations par minute. Sinon, sa valeur sera 60 (pour 60 pulsations par minute).

## Exemples

Voici un exemple de l'opcode *tempoval*. Se rappeler qu'il ne fonctionne que si l'on utilise l'option *-t* avec Csound. Il utilise le fichier *tempoval.csd* [exemples/tempoval.csd].

### Exemple 1027. Exemple de l'opcode *tempoval*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      -t60 ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o tempoval.wav -W -t60 ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Adjust the tempo to 120 beats per minute.
tempo 120, 60

; Get the tempo value.
kval tempoval

printks "kval = %f\\n", 0.1, kval
endin
```

```
</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e
```

```
</CsScore>
</CsoundSynthesizer>
```

Comme 120 pulsations par minute représente un accroissement de 50% de l'original à 60 pulsations par minute, sa sortie contiendra cette ligne :

```
kval = 0.500000
```

## Voir aussi

*tempo* and *miditempo*

## Crédits

Exemple écrit par Kevin Conder.

Nouveau dans la version 4.15

Décembre 2002. Merci à Drake Wilson pour avoir fait remarquer que la documentation n'était pas claire.

# tigoto

tigoto — Transfère le contrôle lors de la phase d'initialisation si la nouvelle note est liée à la précédente note tenue.

## Description

Semblable à *igoto* mais ne fonctionne que lors d'une phase d'initialisation concernant une nouvelle note « liée » à une note précédente tenue. (Voir l'*instruction i*). Ca ne fonctionne pas s'il n'y a pas de liaison. Permet à un instrument d'ignorer l'initialisation de ses unités si une liaison a été proposée avec succès. (Voir aussi *tival*).

## Syntaxe

```
tigoto label
```

où *label* se trouve dans le même bloc d'instrument et n'est pas une expression.

## Exemples

Voici un exemple de l'opcode *tigoto*. Il utilise le fichier *tigoto.csd* [examples/tigoto.csd].

### Exemple 1028. Exemple de l'opcode *tigoto*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

idur = abs(p3)    ;make p3 positive even if p3 is negative in score
itiv  tival
i1    = -1    ;assume this is tied note, so keep fase of oscili
        tigoto slur    ;no reinitialisation on tied notes
i1    = 0    ;first note, so reset phase
aatt  line p4, idur, 0    ;primary envelope

slur:
    if itiv==0 kgoto note    ;no expression on first and second note
    aslur linseg 0, idur*.3, p4, idur*.7, 0 ;envelope for slurred note
    aatt = aatt + aslur
```

```
note:
asig oscili aatt, p5, 1, i1
     outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 4096 10 1 ;sine wave

i1 0 -5 .8 451 ;p3 = 5 seconds
i1 1.5 -1.5 .1 512
i1 3 2 .7 440 ;3 notes together--> duration = 5 seconds

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*cigoto, goto, if, igoto, kgoto, timeout*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/2005fall/tiedNotes.html> [http://www.csoundjournal.com/2005fall/tiedNotes.html], écrit par Steven Yi.

# timedseq

timedseq — Séquenceur à variation temporelle.

## Description

Un séquenceur d'évènements dans lequel le temps peut être contrôlé par un pointeur. Les données de la séquence sont stockées dans une table.

## Syntaxe

```
ktrig timedseq ktmpnt, ifn, kp1 [,kp2, kp3, ...,kpN]
```

## Initialisation

*ifn* -- numéro de la table contenant les données de la séquence.

## Exécution

*ktri* -- signal de déclenchement en sortie.

*ktmpnt* -- pointeur de temps dans le fichier de la séquence, en secondes.

*kp1,...,kpN* -- p-champs des notes retournés en sortie. *kp2* est la date relative et *kp3* est la durée des notes en secondes.

*timedseq* est un séquenceur qui permet de programmer des notes venant d'une séquence de l'utilisateur et dépendant d'une base de temps externe donnée par un pointeur de temps (l'argument *ktmpnt*). L'utilisateur doit remplir la table *ifn* avec une liste de notes, qui peuvent provenir d'un fichier texte externe lu par *GEN23*, ou en les tapant directement dans le fichier d'orchestre (ou de partition) avec *GEN02*. Le format du fichier texte contenant la séquence comprend simplement des lignes qui contiennent plusieurs nombres séparés par des espaces (comme dans une partition normale de Csound). La première valeur de chaque ligne doit être une valeur positive ou nulle, sauf dans un cas spécial qui sera expliqué ci-dessous. Cette première valeur sert normalement à définir le numéro d'instrument correspondant à cette note particulière (comme dans une partition normale). La seconde valeur de chaque ligne doit contenir la date de la note correspondante et la troisième valeur sa durée. Voici un exemple :

```
0 0      0.25 1  93
0 0.25  0.25 2  63
0 0.5   0.25 3  91
0 0.75  0.25 4  70
0 1     0.25 5  83
0 1.25  0.25 6  75
0 1.5   0.25 7  78
0 1.75  0.25 8  78
0 2     0.25 9  83
0 2.25  0.25 10 70
0 2.5   0.25 11 54
0 2.75  0.25 12 80
-1 3    -1   -1 -1 ;; dernière ligne de la séquence
```

Dans cet exemple, la première valeur de chaque ligne est toujours zéro (c'est une valeur sans signification, mais ce p-champ peut servir, par exemple, pour donner un canal MIDI ou un numéro d'instrument), sauf dans la dernière ligne qui commence par -1. Cette valeur (-1) est une valeur spéciale qui indique la fin de

la séquence. Elle a elle-même une date car les séquences peuvent être lues en boucle. Ainsi la séquence précédente a une durée par défaut de 3 secondes, 3 étant la dernière date de la séquence.

Il est important que TOUTES les lignes contiennent le même nombre de valeurs (dans l'exemple, toutes les lignes contiennent exactement 5 valeurs). Le nombre de valeurs contenues dans chaque ligne DOIT être égal au nombre d'arguments *kpXX* de sortie (noter que même si *kp1*, *kp2*, etc. sont placés à la droite de l'opcode, ce sont des arguments de sortie, pas des arguments d'entrée).

L'argument *ktimpnt* fournit la temporisation réelle de la séquence. Actuellement, le déroulement du temps dans la séquence est spécifié par *ktimpnt* lui-même, qui représente le temps en secondes. *ktimpnt* doit toujours être positif, mais il ne peut pas avancer ou reculer dans le temps, être stationnaire ou discontinu, comme un pointeur dans un fichier séquentiel à la manière de *pvoc* ou de *lpread*. Lorsque *ktimpnt* atteint la date d'une note, un signal de déclenchement est envoyé sur l'argument de sortie *ktrig*, et les arguments *kp1*, *kp2*, ..., *kpN* sont mis à jour avec les valeurs de chaque note. Cette information peut ensuite être utilisée par *schedkwhen* pour activer des événements de note. Noter que les données *kp1*, ..., *kpn* peuvent être traitées (par exemple retardées avec *delayk*, transposées, etc.) avant d'être passées à *schedkwhen*.

*ktimpnt* peut être contrôlé par un signal linéaire, par exemple :

```
ktimpnt line      0, p3, 3 ; la durée originale de la séquence était de 3 sec
ktrig  timedseq   ktimpnt, 1, kp1, kp2, kp3, kp4, kp5
      schedkwhen ktrig, 105, 2, 0, kp3, kp4, kp5
```

Dans ce cas la séquence complète (avec sa durée originale de 3 secondes) sera jouée en *p3* secondes.

On peut faire boucler une séquence en la contrôlant avec un phaseur :

```
kphs  phasor      1/3
ktimpnt =          kphs * 3
ktrig  timedseq    ktimpnt, 1, kp1, kp2, kp3, kp4, kp5
      schedkwhen ktrig, 105, 2, 0, kp3, kp4, kp5
```

Il est évident que l'on peut ne jouer qu'un fragment de la séquence, la lire à l'envers, et avoir un accès non-linéaire à ses données de la même manière que les opcodes *pvoc* et *lpread*.

Avec l'opcode *timedseq*, on peut faire presque tout ce que l'on fait dans une partition normale, excepté les limitations suivantes :

1. On ne peut pas avoir deux notes commençant exactement à la même date ; actuellement deux notes doivent être séparées d'au moins un k-cycle (sinon le mécanisme de *schedkwhen* en escamote une des deux).
2. Toutes les notes de la séquence doivent avoir le même nombre de p-champs (même si elles activent différents instruments).

On peut remédier à ces limitations en complétant avec des valeurs sans signification les notes des instruments qui ont moins de p-champs que les autres.

## Exemples

Voici un exemple complet de l'opcode *timedseq*. Il utilise le fichier *timedseq.csd* [examples/timedseq.csd].

### Exemple 1029. Exemple de l'opcode *timedseq*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o timedseq.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giseq ftgen 0,0,128,-2, 2, 0, 0.5, 8.00,\ ;first note
                2, 1, 0.5, 8.02,\ ;second note
                2, 2, 0.5, 8.04,\ ;third
                2, 3, 0.5, 8.05,\ ;fourth
                2, 4, 0.5, 8.07,\ ;fifth
                2, 5, 0.5, 8.09,\ ;sixth
                2, 6, 0.5, 8.11,\ ;seventh
                2, 7, 0.5, 9.00,\ ;eight note
                2, 8, 0.5, 8.00,\ ;due to a quirk in the opcode, it needs an extra note - a
                -1, 8, -1, -1 ;last line is a dummy event that indicates to timedseq when t

instr 1

ibeads = 8      ;lengths of sequence in beats
itempo = p4      ;tempo
iBPS = itempo/60 ;beats per second
kphase phasor iBPS/ibeads ;phasor to move through table
kpointer = kphase*ibeads ;multiply phase (range 0 - 1) by the number of beats contained within the s
kp1 init 0
kp2 init 0
kp3 init 0
kp4 init 0
ktrigger timedseq kpointer, giseq, kp1, kp2,kp3, kp4
schedkwhen ktrigger, 0, 0, 2, 0, kp3/abs(iBPS), kp4 ;p3 values have been scaled according to tempo so t
endin ;abs(iBPS)(absolute value) is used because the tempo provided by the fourth note of the sco
; Durations here should be positive, because negative values for duration would indicate a held n
instr 2

aenv linseg 0,0.01,1,p3-0.01,0 ;amplitude envelope
asig vco2 0.4, cpspch(p4), 4, 0.5
outs asig*aenv, asig*aenv
endin

</CsInstruments>
<CsScore>
i 1 0 4 120
i 1 + . 240
i 1 + . 480
i 1 + . -480 ;when negative it plays backwards
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

GEN02, GEN23, seqtime, seqtime2, trigseq

## Crédits

Auteur : Gabriel Maldonado

# timeinstk

timeinstk — Lit le temps absolu en cycles de taux-k.

## Description

Lit le temps absolu en cycles de taux-k, depuis le démarrage d'une instance d'un instrument. Appelé aussi bien lors de la phase d'initialisation que pendant la phase d'exécution.

## Syntaxe

```
kres timeinstk
```

## Exécution

*timeinstk* donne le temps en cycles de taux-k. Ainsi avec :

```
sr      = 44100
kr      = 6300
ksmps   = 7
```

après une demi-seconde, l'opcode *timeinstk* retournera 3150. Il retourne toujours un nombre entier.

*timeinstk* produit une variable de taux-k en sortie. Il n'y a pas de paramètres d'entrée.

*timeinstk* est semblable à *timek* sauf qu'il retourne le temps écoulé depuis le démarrage de cette instance de l'instrument.

## Exemples

Voici un exemple de l'opcode *timeinstk*. Il utilise le fichier *timeinstk.csd* [examples/timeinstk.csd].

### Exemple 1030. Exemple de l'opcode *timeinstk*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o timeinstk.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
```



```
instr 1
; Print out the value from timeinstk every half-second.
k1 timeinstk
printks "k1 = %f samples\\n", 0.5, k1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
k1 = 1.000000 samples
k1 = 2205.000000 samples
k1 = 4410.000000 samples
k1 = 6615.000000 samples
k1 = 8820.000000 samples
```

## Voir aussi

*timeinsts, timek, times*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Exemple écrit par Kevin Conder.

# timeinsts

timeinsts — Lit le temps absolu en secondes.

## Description

Lit le temps absolu en secondes, depuis le démarrage d'une instance d'un instrument.

## Syntaxe

```
kres timeinsts
```

## Exécution

Le temps en secondes est donné par *timeinsts*. Il retournera 0.5 après une demi-seconde.

*timeinsts* produit une variable de taux-k en sortie. Il n'y a pas de paramètres d'entrée.

*timeinsts* est semblable à *times* sauf qu'il retourne le temps écoulé depuis le démarrage de cette instance de l'instrument.

## Exemples

Voici un exemple de l'opcode *timeinsts*. Il utilise le fichier *timeinsts.csd* [examples/timeinsts.csd].

### Exemple 1031. Exemple de l'opcode *timeinsts*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o timeinsts.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 2^10, 10, 1

instr 1

kvib init 1
ktim timeinsts      ;read time

if ktim > 2 then      ;do something after 2 seconds
    kvib oscili 2, 3, giSine ;make a vibrato
endif
```

```
asig poscil .5, 600+kvib, giSine ;add vibrato
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*timeinstk, timek, times*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

# timek

timek — Lit le temps absolu en cycles de taux-k.

## Description

Lit le temps absolu en cycles de taux-k, depuis le début de l'exécution.

## Syntaxe

```
ires timek
```

```
kres timek
```

## Exécution

*timek* donne le temps en cycles de taux-k. Ainsi avec :

```
sr      = 44100  
kr      = 6300  
ksmps   = 7
```

après une demi-seconde, l'opcode *timek* retournera 3150. Il retourne toujours un nombre entier.

*timek* produit une variable de taux-k en sortie. Il n'y a pas de paramètres d'entrée.

*timek* peut aussi opérer seulement au démarrage de l'instance de l'instrument. Il produit alors une variable de taux-i (préfixée par *i* ou *gi*) en sortie.

## Exemples

Voici un exemple de l'opcode *timek*. Il utilise le fichier *timek.csd* [exemples/timek.csd].

### Exemple 1032. Exemple de l'opcode *timek*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
; Audio out  Audio in  
-odac          -iadc      ;;RT audio I/O  
; For Non-realtime ouput leave only the line below:  
; -o timek.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
; Initialize the global variables.  
sr = 44100  
kr = 4410  
ksmps = 10  
nchnls = 1
```

```

; Instrument #1.
instr 1
; Print out the value from timek every half-second.
k1 timek
printks "k1 = %f samples\\n", 0.5, k1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

k1 = 1.000000 samples
k1 = 2205.000000 samples
k1 = 4410.000000 samples
k1 = 6615.000000 samples
k1 = 8820.000000 samples

```

## Voir aussi

*timeinstk, timensts, times*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.47

Exemple écrit par Kevin Conder.

# times

times — Lit le temps absolu en secondes.

## Description

Lit le temps absolu en secondes, depuis le début de l'exécution.

## Syntaxe

```
ires times
```

```
kres times
```

## Exécution

Le temps en secondes est donné par *times*. Il retournera 0.5 après une demi-seconde.

*times* produit une variable de taux-k en sortie. Il n'y a pas de paramètres d'entrée.

*times* peut aussi opérer au démarrage de l'instance de l'instrument. Il produit alors une variable de taux-i (préfixée par *i* ou *gi*) en sortie.

## Exemples

Voici un exemple de l'opcode times. Il utilise le fichier *times\_complex.csd* [examples/times\_complex.csd].

### Exemple 1033. Exemple de l'opcode times.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o times_complex.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
;by joachim heintz and rory walsh
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giWave   ftgen      0, 0, 1024, 10, 1, .5, .25

instr again

instance =          p4
;reset the duration of this instance
iDur      rnd31      5, 3      ;shorter values are more probable
iDur      =          abs(iDur) + 0.2
p3       =          iDur
;trigger the effect instrument of this instance
```

```

        event_i      "i", "fx_processor", 0, iDur, instance
;print the status quo
kTime      times
prints      "instance = %d, start = %f, duration = %f\n", instance, i(kTime), iDur
;make sound
iamp       active    1      ;scale amplitudes
iOct       random    5, 10   ;find pitch
aEnv       transeg    0, 0.02, 0, 1/iamp, p3-0.02, -6, 0 ;output envelope
aSend      poscil     aEnv, cpsoct(iOct), giWave ;audio signal
;send signal to effect instrument
Sbus       sprintf    "audio_%d", instance ;create unique software bus
           chnset      aSend/2, Sbus ;send audio on this bus
;get the last k-cycle of this instance and trigger the successor in it
kLast      release
           schedkwhen kLast, 0, 0, "again", 0, 1, instance+1
endin

instr fx_processor
;apply feedback delay to the above instrument
iwhich     =          p4      ;receive instance number ...
Sbus       sprintf    "audio_%d", iwhich ; ... and related software bus
audio      chnget      Sbus ;receive audio on this bus
irvbtim    random     1, 5    ;find reverb time
p3         =          p3+irvbtim ;adjust instrument duration
iltptmL    random     .1, .5   ;find looptime left ...
iltptmR    random     .1, .5   ;...and right
ipan       random     0, 1     ; pan and ...
imix       random     0, 1     ;... mix audio
aL,aR      pan2        audio, ipan ;create stereo
awetL      comb        aL, irvbtim, iltptmL ;comb filter
awetR      comb        aR, irvbtim, iltptmR
aoutL      ntrpol       aL, awetL, imix ;wet-dry mix
aoutR      ntrpol       aR, awetR, imix
           outs         aoutL/2, aoutR/2
endin

</CsInstruments>
<CsScore>
i "again" 0 1 1

e 3600
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

new alloc for instr again:
instance = 1, start = 0.000000, duration = 0.650439
new alloc for instr fx_processor:
instance = 2, start = 0.650884, duration = 0.411043
new alloc for instr fx_processor:
instance = 3, start = 1.061587, duration = 0.231085
new alloc for instr fx_processor:
instance = 4, start = 1.292336, duration = 0.543473
new alloc for instr fx_processor:
instance = 5, start = 1.835828, duration = 1.777097

```

## Voir aussi

*timeinstk, timeinsts, timek*

## Crédits

Auteur : Robin Whittle

Australie  
Mai 1997



# timeout

timeout — Branchement conditionnel durant l'exécution en fonction de la durée de la note qui s'est déjà écoulée.

## Description

Branchement conditionnel durant l'exécution en fonction de la durée de la note qui s'est déjà écoulée. *istrt* et *idur* sont exprimés en secondes. Le branchement vers *label* aura lieu à partir de l'instant *istrt*, et restera actif pendant *idur* secondes. Noter que *timeout* peut être réinitialisé pour des activations multiples dans une seule note (voir l'exemple de *reinit*).

## Syntaxe

```
timeout istrt, idur, label
```

où *label* se trouve dans le même bloc d'instrument et n'est pas une expression.

## Exemples

Voici un exemple de l'opcode timeout. Il utilise le fichier *timeout.csd* [examples/timout.csd].

### Exemple 1034. Exemple de l'opcode timeout.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o timeout.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

indx = 0
itim = p4      ;change time for one step

clock:
    timeout 0, itim, time
    reinit clock

time:
    itmp table indx, 2, 0, 0, 1
    if itmp == 1 then
        print itmp
        event_i "i",2, 0, .1      ;event has duration of .1 second
```

```

endif
indx = indx+1

endin

instr 2 ;play it

kenv transeg 0.01, p3*0.25, 1, 1, p3*0.75, .5, 0.01
asig oscili kenv*.4, 400, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 1024 10 1 ;sine
f 2 0 16 2 1 0 0 1 0 1 0 1 0 1 0 0 0 0 0 0 ;the rythm table

i1 0 10 .1
i1 + 10 .05
i1 + 10 .01
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*goto, if, igoto, kgoto, tigo*

# tival

tival — Met la valeur du drapeau interne de « liaison » de l'instrument dans la variable de taux i.

## Syntaxe

ir tival

## Description

Met la valeur du drapeau interne de « liaison » de l'instrument dans la variable de taux i.

## Initialisation

Met la valeur du drapeau interne de « liaison » de l'instrument dans la variable de taux i. Affecte 1 si la note est « liée » à une note tenue précédente (voir l'*instruction i*) ; affecte 0 s'il n'y a pas de liaison. (Voir aussi *tigoto*.)

## Exemples

Voici un exemple de l'opcode tival. Il utilise le fichier *tival.csd* [examples/tival.csd].

### Exemple 1035. Exemple de l'opcode tival.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tival.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

idur = abs(p3)    ;make p3 positive even if p3 is negative in score
itiv tival
i1 = -1    ;assume this is tied note, so keep fase of oscili
tigoto slur    ;no reinitialisation on tied notes
i1 = 0    ;first note, so reset phase
aatt line p4, idur, 0    ;primary envelope

slur:
if itiv==0 kgoto note    ;no expression on first and second note
aslur linseg 0, idur*.3, p4, idur*.7, 0 ;envelope for slurred note
aatt = aatt + aslur
```

```
note:
asig oscili aatt, p5, 1, i1
    outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 4096 10 1 ;sine wave

i1 0 -5 .8 451 ;p3 = 5 seconds
i1 1.5 -1.5 .1 512
i1 3 2 .7 440 ;3 notes together--> duration = 5 seconds

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*=, divz, init*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/2005fall/tiedNotes.html> [http://www.csoundjournal.com/2005fall/tiedNotes.html], écrit par Steven Yi.

# tlineto

tlineto — Génère des glissandi déclenchés par un signal de contrôle.

## Description

Génère des glissandi déclenchés par un signal de contrôle.

## Syntaxe

```
kres tlineto ksig, ktime, ktrig
```

## Exécution

*kres* -- Signal de sortie.

*ksig* -- Signal d'entrée.

*ktime* -- Durée du glissando en secondes.

*ktrig* -- Signal de déclenchement.

*tlineto* est semblable à *lineto* mais on peut l'appliquer à n'importe quelle sorte de signal (pas seulement des signaux en escalier) sans produire de discontinuités. La dernière valeur de chaque segment est échantillonnée et bloquée à partir du signal d'entrée chaque fois que la valeur de *ktrig* est différente de zéro. Normalement le signal *ktrig* est constitué d'une suite de zéros (voir l'opcode *trigger*).

L'effet de glissando est assez différent de celui de *port*. En effet, ici, les lignes sont droites. De plus, le contexte d'utilisation est différent.

## Exemples

Voici un exemple de l'opcode *tlineto*. Il utilise le fichier *tlineto.csd* [examples/tlineto.csd].

### Exemple 1036. Exemple de l'opcode *tlineto*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tlineto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
```

```

giSine ftgen 0, 0, 2^10, 10, 1

instr 1

kmtr lfo 1, .5, 1 ;produce trigger signal
ktr trigger kmtr, .5, 0 ;with triangle wave

ktime = p4
kfreq randh 1000, 3, .2, 0, 500 ;generate random values
kfreq tlineto kfreq, ktime, ktr ;different glissando times
aout poscil .4, kfreq, giSine
outs aout, aout

endin
</CsInstruments>
<CsScore>

i 1 0 10 .2 ;short glissando
i 1 11 10 .8 ;longer glissande
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*lineto*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.13

# tone

tone — Un filtre passe-bas récursif du premier ordre avec une réponse en fréquence variable.

## Description

Un filtre passe-bas récursif du premier ordre avec une réponse en fréquence variable.

*tone* est filtre RII à un terme. Sa formule est :

$$y_n = c1 * x_n + c2 * y_{n-1}$$

où

- $b = 2 - \cos(2 \pi \text{ hp/sr})$ ;
- $c2 = b - \sqrt{b^2 - 1.0}$
- $c1 = 1 - c2$

## Syntaxe

```
ares tone asig, khp [, iskip]
```

## Initialisation

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*ares* -- le signal audio de sortie.

*asig* -- le signal audio en entrée.

*khp* -- le point à mi-puissance de la courbe de réponse, en Hertz. La mi-puissance est définie par puissance maximale / racine de 2.

*tone* implémente un filtre passe-bas récursif du premier ordre dans lequel la variable *khp* (en Hz) détermine le point à mi-puissance de la courbe de réponse. La mi-puissance est définie par puissance maximale / racine de 2.

## Exemples

Voici un exemple de l'opcode *tone*. Il utilise le fichier *tone.csd* [examples/tone.csd].

### Exemple 1037. Exemple de l'opcode *tone*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tone.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1

asig diskin2 "beats.wav", 1
outs asig, asig
endin

instr 2

kton line 10000, p3, 0 ;all the way down to 0 Hz
asig diskin2 "beats.wav", 1
asig tone asig, kton ;half-power point at 500 Hz
outs asig, asig
endin

</CsInstruments>
<CsScore>

i 1 0 2
i 2 2 2

e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*areson, aresonk, atone, atonek, port, portk, reson, resonk, tonek*



# tonek

tonek — Un filtre passe-bas récursif du premier ordre avec une réponse en fréquence variable.

## Description

Un filtre passe-bas récursif du premier ordre avec une réponse en fréquence variable.

## Syntaxe

```
kres tonek ksig, khp [, iskip]
```

## Initialisation

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*kres* -- le signal de sortie au taux de contrôle.

*ksig* -- le signal d'entrée au taux de contrôle.

*khp* -- le point à mi-puissance de la courbe de réponse, en Hertz. La mi-puissance est définie par puissance maximale / racine de 2.

*tonek* est semblable à *tone* à part le fait que sa sortie se fait au taux de contrôle plutôt qu'au taux audio.

## Exemples

Voici un exemple de l'opcode *tonek*. Il utilise le fichier *tonek.csd* [examples/tonek.csd].

### Exemple 1038. Exemple de l'opcode *tonek*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tonek.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
```

```
gisin ftgen 0, 0, 2^10, 10, 1

instr 1

ksig randomh 400, 1800, 150
aout poscil .2, 100+ksig, gisin
outs aout, aout
endin

instr 2

ksig randomh 400, 1800, 150
khp line 1, p3, 100 ;vary high-pass
ksig tonek ksig, khp
aout poscil .2, 100+ksig, gisin
outs aout, aout
endin

</CsInstruments>
<CsScore>

i 1 0 5
i 2 5.5 5
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*areson, aresonk, atone, atonek, port, portk, reson, resonk, tone*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

# tonex

tonex — Emule une série de filtres utilisant l'opcode *tone*.

## Description

*tonex* est équivalent à un filtre constitué de plusieurs couches de filtres *tone* avec les mêmes arguments, connectés en série. L'utilisation d'une série d'un nombre important de filtres permet une pente de coupure plus raide. Ils sont plus rapides que l'équivalent obtenu à partir du même nombre d'instances d'opcodes classiques dans un orchestre Csound, car il n'y aura qu'un cycle d'initialisation et une seule passe de *k* cycles de contrôle à la fois et la boucle audio sera entièrement contenue dans la mémoire cache du processeur.

## Syntaxe

```
ares tonex  asig, khp [, inumlayer] [, iskip]
ares tonex  asig, ahp [, inumlayer] [, iskip]
```

## Initialisation

*inumlayer* (facultatif) -- nombre d'éléments dans la série de filtre. La valeur par défaut est 4.

*iskip* (facultatif, par défaut 0) -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal d'entrée

*khp/ahp* -- le point à mi-puissance de la courbe de réponse. La mi-puissance est définie par puissance maximale / racine de 2.

## Exemples

Voici un exemple de l'opcode *tonex*. Il utilise le fichier *tonex.csd* [examples/tonex.csd].

### Exemple 1039. Exemple de l'opcode *tonex*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tonex.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

instr 1

asig diskin2 "beats.wav", 1
outs asig, asig
endin

instr 2

kton line 10000, p3, 0 ;all the way down to 0 Hz
asig diskin2 "beats.wav", 1
asig tonex asig, kton, 8 ;8 filters
outs asig, asig
endin
</CsInstruments>
<CsScore>

i 1 0 2
i 2 3 2

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*atonex, resonx*

## Crédits

Auteur : Gabriel Maldonado (adapté par John ffitich)  
Italie

Nouveau dans la version 3.49 de Csound

Paramètres de taux audio introduits dans la version 6.02

Octobre 2013.

# trandom

**trandom** — Génère une suite contrôlée de nombres pseudo-aléatoires entre des valeurs minimale et maximale en fonction d'un déclencheur.

## Description

Génère au taux-*k* une suite contrôlée de nombres pseudo-aléatoires entre des valeurs minimale et maximale chaque fois que le paramètre de déclenchement est différent de 0.

## Syntaxe

kout **trandom** ktrig, kmin, kmax

## Exécution

*ktrig* -- déclencheur (l'opcode produit un nouveau nombre aléatoire chaque fois que cette valeur est différente de 0.

*kmin* -- limite inférieure de l'intervalle

*kmax* -- limite supérieure de l'intervalle

*trandom* est presque identique à l'opcode *random* sauf que *trandom* ne renouvelle sa sortie avec une nouvelle valeur aléatoire que si l'argument *ktrig* est déclenché (c-à-d chaque fois qu'il est différent de zéro).

## Exemples

Voici un exemple de l'opcode *trandom*. Il utilise le fichier *trandom.csd* [exemples/trandom.csd].

### Exemple 1040. Exemple de l'opcode *trandom*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o trandom.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

seed 0      ; every run time different values

instr 1
```

```
kmin init 0 ;random number between 0 and 220
kmax init 220
ktrig = p4
k1 trandom ktrig, kmin, kmax
printk2 k1 ;print when k1 changes
asig poscil .4, 220+k1, 1 ;if triggered, add random values to frequency
outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 4096 10 1

i 1 0 2 0 ;not triggered
i 1 + 2 1 ;triggered
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*random*

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5.06

# tradsyn

tradsyn — Synthèse additive d'un flot de suivi de partiels.

## Description

L'opcode *tradsyn* prend en entrée un flot de signal TRACKS pv (tel que généré, par exemple, par *partials*), comme décrit dans Lazzarini et al, "Time-stretching using the Instantaneous Frequency Distribution and Partial Tracking", Proc.of ICMC05, Barcelone. Il resynthétise le signal en utilisant une interpolation linéaire de l'amplitude et de la fréquence pour piloter un banc d'oscillateurs interpolants avec pondération de l'amplitude et de la hauteur.

## Syntaxe

```
asig tradsyn fin, kscal, kpitch, kmaxtracks, ifn
```

## Exécution

*asig* -- signal de sortie au taux audio.

*fin* -- flot pv d'entrée au format TRACKS.

*kscal* -- pondération d'amplitude.

*kpitch* -- pondération de hauteur.

*kmaxtracks* -- nombre maximum de canaux dans la resynthèse. En limitant ce dernier, on obtient un effet de filtrage non-linéaire en ignorant les canaux les plus récents et de fréquences hautes (les canaux sont ordonnés respectivement par date de début et par fréquence ascendante).

*ifn* -- table de fonction contenant une période de sinusoïde (sinus ou cosinus).

## Exemples

Voici un exemple de l'opcode *tradsyn*. Il utilise le fichier *tradsyn.csd* [examples/tradsyn.csd].

### Exemple 1041. Exemple de l'opcode *tradsyn*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tradsyn.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
```

```
nchnls = 2
odbfs = 1

instr 1

ipch = p4
ain disk2 "beats.wav", 1
fsl,fsi2 pvsifd ain,2048,512,1 ; ifd analysis
fst partials fsl,fsi2,.003,1,3,500 ; partial tracking
aout tradsyn fst, 1, ipch, 500, 1 ; resynthesis
outs aout, aout

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1

i 1 0 2 1.5 ;up a 5th
i 1 + 2 .5 ;octave down
e
</CsScore>
</CsoundSynthesizer>
```

L'exemple ci-dessus montre le suivi de partiels d'un signal d'analyse par distribution de fréquence instantanée et la resynthèse additive linéaire avec transposition de la hauteur.

## Crédits

Auteur : Victor Lazzarini  
Juin 2005

Nouveau greffon dans la version 5

Novembre 2004.



# transeg

transeg — Construit une enveloppe définie par l'utilisateur.

## Description

Construit une enveloppe définie par l'utilisateur.

## Syntaxe

```
ares transeg ia, idur, itype, ib [, idur2] [, itype] [, ic] ...  
kres transeg ia, idur, itype, ib [, idur2] [, itype] [, ic] ...
```

## Initialisation

*ia* -- valeur de départ.

*ib*, *ic*, etc. -- valeur après *idur* secondes.

*idur* -- durée en secondes du premier segment. Avec une valeur nulle ou négative, l'initialisation sera ignorée.

*idur2*,...*idurx* etc. -- durée en secondes de chaque segment.

*itype*, *itype2*, etc. -- s'il vaut 0, un segment de droite est produit. S'il est différent de 0, *transeg* crée la courbe suivante en *n* pas :

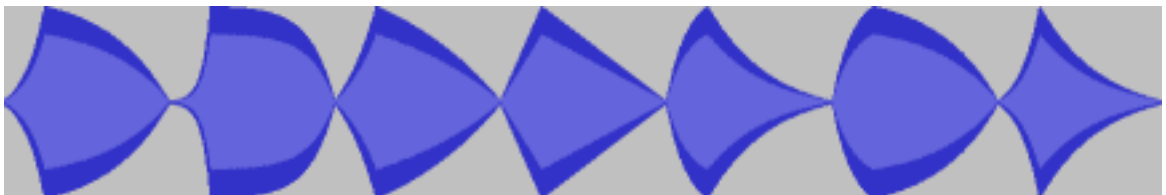
$$ibeg + (ivalue - ibeg) * (1 - \exp(i*itype/(n-1))) / (1 - \exp(itype))$$

## Exécution

Si *itype* > 0, on a une courbe montant lentement (concave) ou décroissant lentement (convexe), tandis que si *itype* < 0, la courbe monte rapidement (convexe) ou décroît rapidement (concave). Voir aussi *GEN16*.

## Exemples

Voici un exemple de l'opcode transeg. Il utilise le fichier *transeg.csd* [examples/transeg.csd]. L'exemple produit la sortie suivante :



Sortie de l'exemple de transeg.

## Exemple 1042. Exemple de l'opcode transeg.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o transeg.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2

0dbfs = 1

instr 1
;p4 and p5 determine the type of curve for each
;section of the envelope
kenv transeg 0.01, p3*0.25, p4, 1, p3*0.75, p5, 0.01
a1 oscil kenv, 440, 1
outs a1, a1
endin

</CsInstruments>
<CsScore>
; Table #1, a sine wave.
f 1 0 16384 10 1

i 1 0 2 2 2
i 1 + . 5 5
i 1 + . 1 1
i 1 + . 0 0
i 1 + . -2 -2
i 1 + . -2 2
i 1 + . 2 -2
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*expsega, expsegr, linseg, linsegr, transegr.*

## Crédits

Auteur : John ffitch  
Université de Bath, Codemist. Ltd.  
Bath, UK  
Octobre 2000

Nouveau dans la version 4.09 de Csound

Merci à Matt Gerassimoff pour avoir précisé la syntaxe correcte de la commande.

# transegb

transegb — Construit une enveloppe définie par l'utilisateur en temps absolu.

## Description

Construit une enveloppe définie par l'utilisateur en temps absolu.

## Syntaxe

```
ares transegb ia, itim, itype, ib [, itim2] [, itype] [, ic] ...  
kres transegb ia, itim, itype, ib [, itim2] [, itype] [, ic] ...
```

## Initialisation

*ia* -- valeur initiale.

*ib*, *ic*, etc. -- valeur après *itim* secondes.

*itim* -- date en secondes de la fin du premier segment.

*itim2*,... *itimx* etc. -- date en secondes de la fin des segments suivants.

*itype*, *itype2*, etc. -- si 0, un segment de droite est généré. Si différent de 0, *transegb* crée la courbe suivante sur *n* pas :

$$\text{ibeg} + (\text{ivalue} - \text{ibeg}) * (1 - \exp(i * \text{itype} / (n - 1))) / (1 - \exp(\text{itype}))$$

## Exécution

Si *itype* > 0, on a une courbe montant lentement (concave) ou décroissant lentement (convexe), tandis que si *itype* < 0, la courbe monte rapidement (convexe) ou décroît rapidement (concave). Voir aussi *GEN16*.

## Exemples

Voici un exemple de l'opcode transegb. Il utilise le fichier *transegb.csd* [examples/transegb.csd]. L'exemple produit la sortie suivante :

Sortie de l'exemple de transegb.

### Exemple 1043. Exemple de l'opcode transegb.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform
```

```

; Audio out   Audio in
-odac         -iadc   ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o transeg.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2

0dbfs = 1

instr 1
;p4 and p5 determine the type of curve for each
;section of the envelope
kenv transegb 0.01, p3*0.25, p4, 1, p3, p5, 0.01
a1 oscil kenv, 440, 1
outs a1, a1
endin

</CsInstruments>
<CsScore>
; Table #1, a sine wave.
f 1 0 16384 10 1

i 1 0 2 2 2
i 1 + . 5 5
i 1 + . 1 1
i 1 + . 0 0
i 1 + . -2 -2
i 1 + . -2 2
i 1 + . 2 -2
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*expseg, expsega, expsega, expsegr, linseg, linsegb, linsegr, transeg transegr*

## Crédits

Auteur : John ffitch  
 Université de Bath, Codemist. Ltd.  
 Bath, UK  
 Juin 2011

Nouveau dans la version 5.14 de Csound.

# transegr

transegr — Construit une enveloppe définissable par l'utilisateur prolongée par un segment de relâchement.

## Description

Construit une enveloppe définissable par l'utilisateur. Semblable à *transeg*, avec un segment de relâchement en prolongement.

## Syntaxe

```
ares transegr ia, idur, itype, ib [, idur2] [, itype] [, ic] ...
```

```
kres transegr ia, idur, itype, ib [, idur2] [, itype] [, ic] ...
```

## Initialisation

*ia* -- valeur de départ.

*ib*, *ic*, etc. -- valeur après *idur* secondes.

*idur* -- durée en secondes du premier segment. Avec une valeur nulle ou négative toute initialisation sera ignorée.

*idur2*,... *idurx* etc. -- durée de segment en secondes.

*itype*, *itype2*, etc. -- s'il vaut 0, un segment de droite est produit. S'il est non nul, alors *transegr* crée la courbe suivante pour *n* pas :

$$\text{ibeg} + (\text{ivalue} - \text{ibeg}) * (1 - \exp(-i * \text{itype} / (n-1))) / (1 - \exp(\text{itype}))$$

## Exécution

Si *itype* > 0, il y a une courbe croissant lentement (concave) ou décroissant lentement (convexe), tandis que si *itype* < 0, la courbe est à croissance rapide (convexe) ou à décroissance rapide (concave). Voir aussi *GEN16*.

Cet opcode est le même que *transeg* avec un segment de relâchement additionnel déclenché par un événement MIDI noteoff, un *événement de note* avec p1 négatif dans la partition ou un opcode *turnoff2*.

## Exemples

Voici un exemple de l'opcode transegr. Il utilise le fichier *transegr.csd* [exemples/transegr.csd].

### Exemple 1044. Exemple de l'opcode transegr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0   ;;realtime audio out and realtime midi in
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o transegr.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

icps cpsmidi
iamp ampmidi .2
;          st,dur1,typ1,val,dur2,typ2,end
kenv transegr 0, .2, 2, .5, 1, - 3, 0
asig pluck kenv*iamp, icps, icps, 1, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 4096 10 1 ;sine

f0 30 ;runs 30 seconds
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*expsega, expsegr, linseg, linsegr, transeg*

## Crédits

Auteur : John ffitch  
Janvier 2010

Nouveau dans la version 5.12 de Csound.

# trcross

trcross — Synthèse croisée à flot de suivi de partiels.

## Description

L'opcode *trcross* prend deux entrées contenant des flots de signal TRACKS pv (tels que générés, par exemple, par *partials*) et en fait une synthèse croisée dans un flot TRACKS unique. Deux modes d'opération différents sont utilisés : mode 0, synthèse croisée par multiplication de l'amplitude des deux entrées et mode 1, synthèse croisée par remplacement des amplitudes de l'entrée 1 par celles de l'entrée 2. Les fréquences et les phases de l'entrée 1 sont conservées dans la sortie. La synthèse croisée est réalisée en assortissant les canaux entre les deux entrées en utilisant un "intervalle de recherche". L'algorithme d'assortiment cherche dans la seconde entrée les canaux qui se trouvent dans l'intervalle de recherche entourant chaque canal de la première entrée. On peut changer cet intervalle au taux de contrôle. Plus les intervalles sont larges et plus on trouve d'assortiments.

## Syntaxe

```
fsig trcross fin1, fin2, ksearch, kdepth [, kmode]
```

## Exécution

*fsig* -- flot pv de sortie au format TRACKS.

*fin1* -- premier flot pv d'entrée au format TRACKS.

*fin2* -- second flot pv d'entrée au format TRACKS.

*ksearch* -- rapport de l'intervalle de recherche définissant une "zone de recherche" autour de chaque canal de la première entrée pour l'assortiment.

*kdepth* -- importance de l'effet (entre 0 et 1).

*kmode* -- mode de synthèse croisée. 0, multiplication des amplitudes (filtrage), 1, remplacement des amplitudes de l'entrée 1 par celles de l'entrée 2 (comme pour le vocodeur). Vaut 0 par défaut.

## Exemples

Voici un exemple de l'opcode trcross. Il utilise le fichier *trcross.csd* [examples/trcross.csd].

### Exemple 1045. Exemple de l'opcode trcross.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o trcross.wav -W ;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

  sr = 44100
  ksmpr = 32
  nchnls = 2
  odbfs = 1

  instr 1

  ain1 disk2 "beats.wav", 1, 0, 1
  ain2 disk2 "fox.wav", 1

  imode = p4
  fs1,fsi2 pvsifd ain1, 2048, 512, 1 ; ifd analysis
  fst      partials fs1, fsi2, .01, 1, 3, 500 ; partial tracking

  fs11,fsi12 pvsifd ain2, 2048, 512, 1 ; ifd analysis (second input)
  fst1      partials fs11, fsi12, .01, 1, 3, 500 ; partial tracking (second input)

  fcr      trcross fst, fst1, 1.05, 1, imode ; cross-synthesis (mode 0 and mode 1)
  aout      tradsyn fcr, 1, 1, 500, 1 ; resynthesis of tracks
           outs aout*3, aout*3

  endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1

i 1 0 3 0
i 1 5 3 1

e
</CsScore>
</CsoundSynthesizer>

```

L'exemple ci-dessus montre le suivi de partiels de deux signaux d'analyse par distribution de fréquence instantanée, la synthèse croisée suivie du remixage des deux parties du spectre et de la resynthèse.

## Crédits

Auteur : Victor Lazzarini  
Février 2006

Nouveau dans Csound 5.01



# trfilter

trfilter — Filtrage d'un flot de suivi de partiels.

## Description

L'opcode *trfilter* prend en entrée un flot de signal TRACKS pv (tel que généré, par exemple, par *partials*) et le filtre en utilisant la courbe de réponse des amplitudes stockée dans une table de fonction. La table de fonction peut avoir n'importe quelle taille (aucune restriction aux puissances de deux). La table est consultée avec interpolation linéaire. Il est possible de créer des courbes de filtrage variant dans le temps en mettant à jour la table de la réponse des amplitudes avec un opcode d'écriture de table.

## Syntaxe

```
fsig trfilter fin, kamnt, ifn
```

## Exécution

*fsig* -- flot pv de sortie au format TRACKS.

*fin* -- flot pv d'entrée au format TRACKS.

*kamnt* -- importance du filtrage (entre 0 et 1)

*ifn* -- numéro de la table de fonction. Celle-ci contient une courbe de réponse des amplitudes, de 0 Hz à la fréquence de Nyquist (table indexée entre 0 et N). Toutes les tailles sont permises. Plus la table est grande et plus la courbe de réponse est lisse. La consultation de la table se fait avec interpolation linéaire.

## Exemples

Voici un exemple de l'opcode *trfilter*. Il utilise le fichier *trfilter.csd* [examples/trfilter.csd].

### Exemple 1046. Exemple de l'opcode *trfilter*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o trfilter.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
sr = 44100
ksmps = 32
nchnls = 2
odbfs = 1
```

```
gifn ftgen 2, 0, -22050, 5, 1, 1000, 1, 4000, 0.000001, 17050, 0.000001 ; low-pass filter curve of 22050
```

```
instr 1

kam line 1, p3, p4
ain diskint2 "beats.wav", 1, 0, 1
fsl,fsi2 pvsifd ain, 2048, 512, 1 ; ifd analysis
fst partials fsl, fsi2, .003, 1, 3, 500 ; partial tracking
fsc1 trfilter fst, kam, gifn ; filtering using function table 2
aout tradsyn fsc1, 1, 1, 500, 1 ; resynthesis
outs aout, aout

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1

i 1 0 4 1
i 1 5 4 0 ;reduce filter effect
e
</CsScore>
</CsoundSynthesizer>
```

L'exemple ci-dessus montre le suivi de partiels d'un signal d'analyse par distribution de fréquence instantanée et la resynthèse additive linéaire avec filtrage passe-bas.

## Crédits

Auteur : Victor Lazzarini  
Février 2006

Nouveau dans Csound 5.01

# trhighest

trhighest — Extrait le canal de fréquence le plus haut d'un flot de suivi de partiels.

## Description

L'opcode *trhighest* prend en entrée un flot de signal TRACKS pv (tel que généré, par exemple, par *partials*) et ne restitue que le canal le plus haut. De plus, il fournit en sortie deux signaux de taux-k, correspondant à la fréquence et à l'amplitude du signal de canal le plus haut.

## Syntaxe

```
fsig, kfr, kamp trhighest finl, kscal
```

## Exécution

*fsig* -- flot pv de sortie au format TRACKS.

*kfr* -- fréquence (en Hz) du canal de fréquence le plus haut.

*kamp* -- amplitude du canal de fréquence le plus haut.

*fin* -- flot pv d'entrée au format TRACKS.

*kscal* -- pondération d'amplitude de la sortie.

## Exemples

Voici un exemple de l'opcode *trhighest*. Il utilise le fichier *trhighest.csd* [examples/trhighest.csd].

### Exemple 1047. Exemple de l'opcode trhighest.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o trhighest.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ain      diskin2 "fox.wav", 1
fs1,fsi2 pvsifd ain, 2048, 512, 1 ; ifd analysis
fst partials fs1, fsi2, .1, 1, 3, 500 ; partial tracking
```

```
fhi,kfr,kamp trhighest fst, 1 ; highest freq-track
aout tradsyn fhi, 1, 1, 1, 1 ; resynthesis of highest frequency
outs aout*40, aout*40 ; compensate energy loss

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1 ;sine wave

i 1 0 3

e
</CsScore>
</CsoundSynthesizer>
```

L'exemple ci-dessus montre le suivi de partiels d'un signal d'analyse par distribution de fréquence instantanée, l'extraction de la fréquence la plus haute et la resynthèse.

## Crédits

Auteur : Victor Lazzarini  
Février 2006

Nouveau dans Csound 5.01

# trigger

trigger — Informe quand un signal de taux-k traverse un seuil.

## Description

Informe quand un signal de taux-k traverse un seuil.

## Syntaxe

kout **trigger** ksig, kthreshold, kmode

## Exécution

*ksig* -- signal d'entrée

*kthreshold* -- seuil de déclenchement

*kmode* -- peut valoir 0, 1 ou 2

Normalement *trigger* retourne des zéros : *trigger* retourne 1 chaque fois que *ksig* traverse *kthreshold*. Il y a trois modes d'utilisation de *ktrig* :

- *kmode* = 0 - (bas-haut) *ktrig* retourne 1 lorsque la valeur courante de *ksig* est supérieure à *kthreshold*, alors que l'ancienne valeur de *ksig* était égale ou inférieure à *kthreshold*.
- *kmode* = 1 - (haut-bas) *ktrig* retourne 1 lorsque la valeur courante de *ksig* est inférieure à *kthreshold* alors que l'ancienne valeur de *ksig* était égale ou supérieure à *kthreshold*.
- *kmode* = 2 - (les deux) *ktrig* retourne 1 dans les deux cas précédents.

## Exemples

Voici un exemple de l'opcode trigger. Il utilise le fichier *trigger.csd* [exemples/trigger.csd].

### Exemple 1048. Exemple de l'opcode trigger.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o trigger.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
```

```

instr 1

kmtr lfo 1, 1, 1 ;triangle wave
kmode = p4
ktr trigger kmtr, .5, kmode
printk2 ktr
schedkwhen ktr, 0, 3, 2, 0, .3

endin

instr 2

aenv linseg 0,p3*.1,1,p3*.3,1,p3*.6,0 ;envelope
a1 poscil .3*aenv, 1000, 1
outs a1, a1

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine

i 1 0 3 0 ;down-up
i 1 4 3 2 ;down-up & up=down

e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Gabriel Maldonado  
 Italie

Nouveau dans la version 3.49 de Csound.

# trigseq

trigseq — Accepte un signal déclencheur en entrée et retourne un groupe de valeurs.

## Description

Accepte un signal déclencheur en entrée et retourne un groupe de valeurs.

## Syntaxe

```
trigseq ktrig_in, kstart, kloop, kinitndx, kfn_values, kout1 [, kout2] [...]
```

## Exécution

*ktrig\_in* -- signal de déclenchement en entrée.

*kstart* -- indice du début de la section en boucle.

*kloop* -- indice de la fin de la section en boucle.

*kinitndx* -- indice initial.



### Note

Bien que *kinitndx* soit renseigné au taux-k, l'accès ne s'y fait qu'au taux d'initialisation. Ainsi, si l'on utilise un argument de taux-k, son affectation doit se faire avec *init*.

*kfn\_values* -- numéro d'une table contenant une suite de groupes de valeurs.

*kout1* -- valeurs retournées

*kout2*, ... (facultatif) -- plus de valeurs retournées

Cet opcode traite des suites temporelles de groupes de valeurs stockées dans une table.

*trigseq* accepte un signal déclencheur (*ktrig\_in*) en entrée et retourne un groupe de valeurs (contenues dans la table *kfn\_values*) chaque fois que *ktrig\_in* admet une valeur différente de zéro. Chaque fois qu'un groupe de valeurs est déclenché, le pointeur de la table est avancé du nombre de positions correspondant au nombre d'éléments de ce groupe, afin de pointer vers le groupe suivant de valeurs. Le nombre d'éléments des groupes est déterminé par le nombre d'arguments *koutX*.

Il est possible de démarrer la séquence depuis une valeur différente de la première, en affectant à *kinitndx* un indice différent de zéro (qui correspond à la première valeur de la table). Normalement la séquence est bouclée, et le début et la fin de la boucle peuvent être ajustés en modifiant les arguments *kstart* et *kloop*. L'utilisateur doit s'assurer que les valeurs de ces arguments (ainsi que celle de *kinitndx*) correspondent à des indices de table valides, sinon Csound plantera (car il n'y a aucun test sur ces indices).

Il est possible de désactiver la boucle (mode à une passe) en affectant la même valeur aux arguments *kstart* et *kloop*. Dans ce cas, le dernier élément lu sera celui correspondant à la valeur de ces arguments. La table peut être lue à l'envers en affectant une valeur négative à *kloop*.

*trigseq* est conçu pour être utilisé avec les opcodes *seqtime* ou *trigger*.

### Exemple 1049. Exemple de l'opcode trigseq.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o trigseq.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giTimes ftgen 91, 0, 128, -2,      1, 1/2, 1/2, 1/8, 1/8, 1/2,1/2, 1/16, 1/16, 1/16, 1/16, 1/16, 1/16, 1
giSeq ftgen 90, 0, 128, -2,      1, 2,      .5, 3,      .25, 4,      .10, 5,      .05, 6 ;** sequence amp

instr 1

icps init p4
iamp init .3

kloop init p5
initndx init p6
kloop2 init p7
initndx2 init p8
kdur init p9
iminTime init p10
imaxTime init p11
kampratio init 1
kfregratio init 1

ktime_unit expseg iminTime,p3/8,iminTime,p3* 3/4,imaxTime,p3/8,imaxTime

; **ktrig seqtime ktime_unit, kstart, kloop, initndx, kfn_times
;ktrig seqtime 1/ktime_unit, 0,      15, 0,      giTimes

ktrig metro ktime_unit

;*** trigseq ktrig_in, kstart, kloop, initndx, kfn_values, kout1 [, kout2, kout3, ..., koutN]
trigseq ktrig, 0, kloop2,initndx2, giSeq,      kampratio, kfregratio

;atrig = ktrig*10000
schedkwhen ktrig, -1, -1, 3, 0, kdur, kampratio*iamp, kfregratio*icps
; schedkwhen ktrig, -1, -1, 2, 0, ktrig, kampratio*iamp, kfregratio*icps
endin

instr 2

icps init p4
iamp init .2
```



```

kloop init p5
initndx init p6
kloop2 init p7
initndx2 init p8
kdur init p9
iminTime init p10
imaxTime init p11
kampratio init 1
kfregratio init 1

ktime_unit expseg iminTime,p3/8,iminTime,p3* 3/4,imaxTime,p3/8,imaxTime

; **ktrig seqtime ktime_unit, kstart, kloop, initndx, kfn_times
ktrig seqtime 1/ktime_unit, 0, 15, 0, giTimes

; ktrig metro ktime_unit

; **** trigseq ktrig_in, kstart, kloop, initndx, kfn_values, kout1 [, kout2, kout3, ..., koutN]
trigseq ktrig, 0, kloop2, initndx2, giSeq, kampratio, kfregratio
printk2 ktrig
; atrig = ktrig*10000
; schedkwhen ktrig, -1, -1, 2, 0, kdur, kampratio*iamp, kfregratio*icps
schedkwhen ktrig, -1, -1, 3, 0, ktrig, kampratio*iamp, kfregratio*icps
endin

instr 3

print p3
kenv expseg 1.04, p3,.04
a1 foscili p4*a(kenv-0.04), p5,1,1,kenv*5, 2
outs a1, a1
endin

</CsInstruments>
<CsScore>
f2 0 8192 10 1

; icps unused unused kloop2 initndx2 kdur iminTime imaxTime

s

i1 0 6 100 0 0 5 0 .2 3 15
i1 8 6 150 0 0 4 1 .1 4 30
i1 16 6 200 0 0 5 3 .25 8 50
i1 24 6 300 0 0 3 0 .1 1 30

i2 32 6 100 0 0 5 0 .2 1 1
i2 40 6 150 0 0 4 1 .1 .5 .5
i2 48 6 200 0 0 5 3 .25 3 .5
i2 56 6 300 0 0 5 0 .1 1 8

e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*seqtime, trigger*

## Crédits

Auteur : Gabriel Maldonado

Novembre 2002. Note sur le paramètre *kinitndx* ajoutée grâce à Rasmus Ekman.

Janvier 2003. J'ai corrigé les crédits grâce à une note de Øyvind Brandtsegg.

Nouveau dans la version 4.06

# trim

trim — Ajuste la taille d'un tableau unidimensionnel.

## Description

Augmente ou réduit un tableau unidimensionnel.

## Syntaxe

```
trim_i iarray, ilen
trim xarray, klen
```

## Exécution

*iarray* -- un tableau unidimensionnel de taux-i

*xarray* -- un tableau unidimensionnel

*klen* -- la taille désirée

Donne au tableau la taille désirée soit en le tronquant soit en le complétant avec des valeurs nulles.

## Exemples

Voici un exemple de l'opcode trim. Il utilise le fichier *trim.csd* [examples/trim.csd].

### Exemple 1050. Exemple de l'opcode trim.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-n
</CsOptions>
<CsInstruments>
instr 1
  kA1[] fillarray 0, 1, 2, 3, 4, 5, 6, 7 ; <-- 8 elements
  printf "lenarray(kA1) before slicearray: %d\n", 1, lenarray:k(kA1)
  kA1 slicearray kA1, 1, 4 ; <-- 4 elements
  printf "lenarray(kA1) AFTER slicearray: %d\n", 1, lenarray:k(kA1)
  trim kA1, 4
  printks "kA1 after trim: { ", 0
  kCnt = 0
  while (kCnt < lenarray:k(kA1)) do
    printf "%d ", kCnt + 1, kA1[kCnt]
    kCnt += 1
  od
  printks "}\n", 0
  turnoff
endin
```

```
</CsInstruments>

<CsScore>
i1  0 0.1
e
</CsScore>

</CsoundSynthesizer>
```

## Voir aussi

*slicearray*

## Crédits

Auteur : John ffitch 2018

Nouveau dans la version 6.12

# trirand

trirand — Générateur de nombres aléatoires de distribution triangulaire.

## Description

Générateur de nombres aléatoires de distribution triangulaire. C'est un générateur de bruit de classe x.

## Syntaxe

```
ares trirand krange  
ires trirand krange  
kres trirand krange
```

## Exécution

*krange* -- l'intervalle des nombres aléatoires (*-krange* à *+krange*).

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode trirand. Il utilise le fichier *trirand.csd* [exemples/trirand.csd].

### Exemple 1051. Exemple de l'opcode trirand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac      ;;realtime audio out  
;-iadc      ;;uncomment -iadc if RT audio input is needed too  
; For Non-realtime ouput leave only the line below:  
; -o trirand.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
ksmps = 32  
nchnls = 2  
0dbfs = 1  
  
instr 1    ; every run time same values  
  
ktri trirand 100  
  printk .2, ktri    ; look
```

```

aout oscili 0.8, 440+ktri, 1 ; & listen
outs aout, aout
endin

instr 2 ; every run time different values

seed 0
ktri trirand 100
printk .2, ktri ; look
aout oscili 0.8, 440+ktri, 1 ; & listen
outs aout, aout
endin

</CsInstruments>
<CsScore>
; sine wave
f 1 0 16384 10 1

i 1 0 2
i 2 3 2
e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

i 1 time 0.00067: -4.97993
i 1 time 0.20067: 1.20909
i 1 time 0.40067: 17.45873
i 1 time 0.60067: 52.55409
i 1 time 0.80067: -1.92888
i 1 time 1.00000: -11.01149
i 1 time 1.20067: 9.79521
i 1 time 1.40067: 26.98504
i 1 time 1.60067: 24.67405
i 1 time 1.80000: -67.59846
i 1 time 2.00000: 64.24861
WARNING: Seeding from current time 521999639
i 2 time 3.00067: 3.28969
i 2 time 3.20067: 54.98986
i 2 time 3.40067: -33.84788
i 2 time 3.60000: -41.93523
i 2 time 3.80067: -6.61742
i 2 time 4.00000: 39.67097
i 2 time 4.20000: 2.95123
i 2 time 4.40067: 45.59255
i 2 time 4.60067: 16.57259
i 2 time 4.80067: -18.80273
i 2 time 5.00000: -2.01697

```

## Voir aussi

*betarand, bexprnd, cauchy, exprand, gauss, linrand, pcauchy, poisson, unirand, weibull*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

# trlowest

trlowest — Extrait le canal de fréquence le plus bas d'un flot de suivi de partiels.

## Description

L'opcode *trlowest* prend en entrée un flot de signal TRACKS pv (tel que généré, par exemple, par *partials*) et ne restitue que le canal le plus bas. De plus, il fournit en sortie deux signaux de taux-k, correspondant à la fréquence et à l'amplitude du signal de canal le plus bas.

## Syntaxe

```
fsig, kfr, kamp trlowest finl, kscal
```

## Exécution

*fsig* -- flot pv de sortie au format TRACKS.

*kfr* -- fréquence (en Hz) du canal de fréquence le plus bas.

*kamp* -- amplitude du canal de fréquence le plus bas.

*fin* -- flot pv d'entrée au format TRACKS.

*kscal* -- pondération d'amplitude de la sortie.

## Exemples

Voici un exemple de l'opcode *trlowest*. Il utilise le fichier *trlowest.csd* [examples/trlowest.csd].

### Exemple 1052. Exemple de l'opcode trlowest.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o trlowest.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ain      diskin2 "beats.wav", 1
fs1,fsi2 pvsifd ain, 2048, 512, 1 ; ifd analysis
fst partials fs1, fsi2, .003, 1, 3, 500 ; partial tracking
```

```
flow,kfr,kamp trlowest fst, 1 ; lowest freq-track
aout tradsyn flow, 1, 1, 1, 1 ; resynthesis of lowest frequency
outs aout*2, aout*2

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1 ;sine wave

i 1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

L'exemple ci-dessus montre le suivi de partiels d'un signal d'analyse par distribution de fréquence instantanée, l'extraction de la fréquence la plus basse et la resynthèse.

## Crédits

Auteur : Victor Lazzarini  
Février 2006

Nouveau dans Csound 5.01



# trmix

trmix — Mixage de flots de suivi de partiels.

## Description

L'opcode *trmix* prend deux entrées contenant des flots de signal TRACKS pv (tels que générés, par exemple, par *partials*) et les mixe en un flot TRACKS unique. Les canaux sont mixés dans l'espace disponible (défini par le nombre original de bins de TFR dans les signaux analysés). Si la somme des canaux en entrée dépasse cet espace, les canaux d'ordre plus élevé dans la seconde entrée sont ignorés.

## Syntaxe

```
fsig trmix fin1, fin2
```

## Exécution

*fsig* -- flot pv de sortie au format TRACKS.

*fin1* -- premier flot pv d'entrée au format TRACKS.

*fin2* -- second flot pv d'entrée au format TRACKS.

## Exemples

Voici un exemple de l'opcode *trmix*. Il utilise le fichier *trmix.csd* [exemples/trmix.csd].

### Exemple 1053. Exemple de l'opcode *trmix*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o trmix.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ain diskin2 "fox.wav", 1
fs1,fsi2 pvsifd ain, 2048, 512, 1 ; ifd analysis
fst partials fs1, fsi2, .003, 1, 3, 500 ; partial tracking
fslo,fshi trsplit fst, 1000 ; split partial tracks at 1000 Hz
fsc1 trscale fshi, 1.3 ; shift the upper tracks
fmix trmix fslo,fsc1 ; mix the shifted and unshifted tracks
```

```
aout tradsyn fmix, 1, 1, 500, 1 ; resynthesis of tracks
outs      aout, aout

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1 ;sine wave

i 1 0 3
e
</CsScore>
</CsoundSynthesizer>
```

L'exemple ci-dessus montre le suivi de partiels d'un signal d'analyse par distribution de fréquence instantanée, la séparation en fréquence et la transposition de hauteur de la partie haute du spectre, suivies du remixage des deux parties du spectre et de la resynthèse.

## Crédits

Auteur : Victor Lazzarini  
Février 2006

Nouveau dans Csound 5.01

# trscale

trscale — Pondération en fréquence d'un flot de suivi de partiels.

## Description

L'opcode *trscale* prend en entrée un flot de signal TRACKS pv (tel que généré, par exemple, par *partials*) et pondère toutes les fréquence d'une quantité de taux-k. Il peut aussi, facultativement, pondérer le gain du signal par une quantité de taux-k (1 par défaut). Le résultat est une transposition de hauteur des canaux d'entrée.

## Syntaxe

```
fsig trscale fin, kpitch [, kgain]
```

## Exécution

*fsig* -- flot pv de sortie au format TRACKS.

*fin* -- flot pv d'entrée au format TRACKS.

*kpitch* -- pondération de fréquence.

*kgain* -- pondération d'amplitude (1 par défaut).

## Exemples

Voici un exemple de l'opcode trscale. Il utilise le fichier *trscale.csd* [examples/trscale.csd].

### Exemple 1054. Exemple de l'opcode trscale.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o trscale.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kpitch = p4
ain diskin2 "fox.wav", 1
fs1,fsi2 pvsifd ain, 2048, 512, 1 ; ifd analysis
fst partials fs1, fsi2, .003, 1, 3, 500 ; partial tracking
```

```
fscl trscale fst, kpitch ; frequency scale
aout tradsyn fscl, 1, 1, 500, 1 ; resynthesis
    outs      aout, aout

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1

i 1 0 3 1.5 ;up a 5th
i 1 3 3 3 ;two octaves higher
e
</CsScore>
</CsoundSynthesizer>
```

L'exemple ci-dessus montre le suivi de partiels d'un signal d'analyse par distribution de fréquence instantanée et la resynthèse additive linéaire avec transposition de hauteur.

## Crédits

Auteur : Victor Lazzarini  
Février 2006

Nouveau dans Csound 5.01

# trshift

trshift — Pondération en fréquence d'un flot de suivi de partiels.

## Description

L'opcode *trshift* prend en entrée un flot de signal TRACKS pv (tel que généré, par exemple, par *partials*) et décale toutes les fréquence d'une fréquence de taux-k. Il peut aussi, facultativement, pondérer le gain du signal par une quantité de taux-k (1 par défaut). Le résultat est un décalage en fréquence des canaux d'entrée.

## Syntaxe

```
fsig trshift fin, kpsift [, kgain]
```

## Exécution

*fsig* -- flot pv de sortie au format TRACKS.

*fin* -- flot pv d'entrée au format TRACKS.

*kshift* -- décalage de fréquence en Hz.

*kgain* -- pondération d'amplitude (1 par défaut).

## Exemples

Voici un exemple de l'opcode trshift. Il utilise le fichier *trshift.csd* [examples/trshift.csd].

### Exemple 1055. Exemple de l'opcode trshift.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o trshift.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kpsft = p4
ain diskin2 "fox.wav", 1
fs1, fsi2 pvsifd ain, 2048, 512, 1          ; ifd analysis
fst      partials fs1, fsi2, 0.003, 1, 3, 500 ; partial tracking
```

```

fsc1    trshift    fst, kpsft          ; frequency shift
aout    tradsyn    fsc1, 1, 1, 500, 1  ; resynthesis
        outs aout, aout

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1 ;sine

i 1 0 3 150 ;adds 150Hz to all tracks
i 1 + 3 500 ;adds 500Hz to all tracks
e
</CsScore>
</CsoundSynthesizer>

```

L'exemple ci-dessus montre le suivi de partiels d'un signal d'analyse par distribution de fréquence instantanée et la resynthèse additive linéaire avec décalage de fréquence.

## Crédits

Auteur : Victor Lazzarini  
Février 2006

Nouveau dans Csound 5.01

# trsplit

trsplit — Séparation en fréquence d'un flot de suivi de partiels.

## Description

L'opcode *trsplit* prend en entrée un flot de signal TRACKS pv (tel que généré, par exemple, par *partials*) et le sépare en deux signaux selon un "point de séparation" en fréquence variant au taux-k. La première sortie contiendra tous les canaux de 0 Hz à la fréquence de séparation et la seconde sortie contiendra les canaux de la fréquence de séparation à la fréquence de Nyquist. Il peut aussi, facultativement, pondérer le gain des signaux de sortie par une quantité de taux-k (1 par défaut). Le résultat est constitué de deux signaux ne contenant chacun qu'une partie du spectre original.

## Syntaxe

```
fsiglow, fsighi trsplit fin, ksplit [, kgainlow, kgainhigh]
```

## Exécution

*fsiglow* -- flot pv de sortie au format TRACKS contenant les canaux sous le point de séparation.

*fsighi* -- flot pv de sortie au format TRACKS contenant les canaux au-dessus de et incluant le point de séparation.

*fin* -- flot pv d'entrée au format TRACKS.

*ksplit* -- point de séparation des fréquences en Hz.

*kgainlow*, *kgainhig* -- pondération d'amplitude de chaque sortie (1 par défaut).

## Exemples

Voici un exemple de l'opcode trsplit. Il utilise le fichier *trsplit.csd* [examples/trsplit.csd].

### Exemple 1056. Exemple de l'opcode trsplit.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o trsplit.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
```

```
instr 1

ain diskin2 "beats.wav", 1
fs1,fsi2 pvsifd ain, 2048, 512, 1 ; ifd analysis
fst partials fs1, fsi2, .003, 1, 3, 500 ; partial tracking
fslo,fshi trsplit fst, 1500 ; split partial tracks at 1500 Hz
aout tradsyn fshi, 1, 1, 500, 1 ; resynthesis of tracks above 1500Hz
      outs aout, aout

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1 ;sine

i 1 0 2
e
</CsScore>
</CsoundSynthesizer>
```

L'exemple ci-dessus montre le suivi de partiels d'un signal d'analyse par distribution de fréquence instantanée et la resynthèse additive linéaire de la partie supérieure du spectre (à partir de 1500 Hz).

## Crédits

Auteur : Victor Lazzarini  
Février 2006

Nouveau dans Csound 5.01



# turnoff

turnoff — Permet à un instrument de s'arrêter lui-même.

## Description

Permet à un instrument de s'arrêter lui-même ou d'arrêter une instance d'un autre instrument.

## Syntaxe

**turnoff**

**turnoff** inst

**turnoff** knst

## Initialisation

*inst* -- descripteur d'instance d'un instrument à arrêter (obtenu d'un opcode *nstance*).

## Exécution

*turnoff* -- sans paramètre cette instruction de la phase d'exécution permet à un instrument de s'arrêter lui-même. Quelle soit de durée finie ou « tenue », la note en cours d'exécution par l'instrument est immédiatement enlevée de la liste des notes actives. Aucune autre note n'est affectée.

*knst* -- descripteur d'instance d'un instrument à arrêter (obtenu d'un opcode *nstance*).

## Exemples

L'exemple suivant utilise l'opcode *turnoff*. Il provoque la fin d'une note lorsqu'un signal de contrôle dépasse un certain seuil (ici la fréquence de Nyquist). Il utilise le fichier *turnoff.csd* [examples/turnoff.csd].

### Exemple 1057. Exemple de l'opcode *turnoff*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o turnoff.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr  = 44100
kr  = 4410
ksmps = 10
nchnls = 1
```

```

; Instrument #1.
instr 1
  k1 expon 440, p3/10,880      ; begin gliss and continue
  if k1 < sr/2  kgoto contin  ; until Nyquist detected
    turnoff ; then quit

contin:
  a1 oscil 10000, k1, 1
  out a1
endin

</CsInstruments>
<CsScore>

; Table #1: an ordinary sine wave.
f 1 0 32768 10 1

; Play Instrument #1 for 4 seconds.
i 1 0 4
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*ihold turnoff2, turnon*

# turnoff2

turnoff2 — Arrête une ou des instances d'autres instruments pendant la phase d'exécution.

## Description

Arrête une ou des instances d'autres instruments pendant la phase d'exécution.

## Syntaxe

**turnoff2** *kinsno*, *kmode*, *krelease*

## Exécution

*kinsno* -- instrument à arrêter (peut-être fractionnaire). S'il vaut zéro ou est négatif, aucun instrument n'est arrêté.

*kmode* -- somme des valeurs suivantes :

- 0, 1, ou 2 : arrête toutes les instances (0), seulement les plus anciennes (1), ou seulement les plus récentes (2)
- 4 : n'arrête que les notes dont la partie fractionnaire du numéro d'instrument correspond à *kinsno*, plutôt que d'ignorer la partie fractionnaire.
- 8 : n'arrête que les notes dont la durée est indéfinie ( $p3 < 0$  ou MIDI).

*krelease* -- s'il est non nul, les instances arrêtées peuvent avoir une période d'extinction (release), sinon elles sont désactivées immédiatement (avec possible émission de clics).

Il faut respecter le principe d'arrêter des instruments ayant un numéro plus élevé que celui de l'instrument duquel *turnoff2* est appelé, sinon il peut y avoir des problèmes d'initialisation.

## Exemples

L'exemple suivant, écrit par Lou Cohen, utilise l'opcode *turnoff2*.

### Exemple 1058. Exemple de l'opcode turnoff2.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o sin.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

sr      =      48000      ;samples per second for sound
kr      =      4800
nchnls  =      1
0dbfs   =      32767

gisine    ftgen    1, 0, 131073, 9, 1, 1, 0      ;single sine tone

;-----
instr 2 ;start
    ktrigger      init      0
    if (ktrigger = 0) then
        kMultiple  =      1.1
        kHz        =      440
        kAmp       =      (0dbfs/10)
;startup four instances of instrument 200
        event      "i", 200, 0, 3000, kAmp, kHz
        kAmp      =      kAmp * 0.75
        kHz       =      kHz * kMultiple

        event      "i", 200, 0, 3000, kAmp, kHz
        kAmp      =      kAmp * 0.75
        kHz       =      kHz * kMultiple

        event      "i", 200, 0, 3000, kAmp, kHz
        kAmp      =      kAmp * 0.75
        kHz       =      kHz * kMultiple

        event      "i", 200, 0, 3000, kAmp, kHz
        kAmp      =      kAmp * 0.75
        kHz       =      kHz * kMultiple
        ktrigger   =      1
    endif
endin
;-----
instr 3 ;after 10 seconds, turn off the instruments
    ktrigger init 1
    if (ktrigger==1) then
        turnoff2 200, 1, 1      ;turn off must recently started instrument instance
        kactive  active 200      ;find out how many are still active
        printk2 kactive        ;print mainly to show progress

        turnoff2 200, 0, 1      ;turn off all the rest of the instruments
        kactive  active 200      ;find out how many are still active
        printk2 kactive, 10     ;print to show progress
    endif
endin
;-----
instr 200 ;play the tone
    kEnv    linen    1, 0.1, p3, 0.1
    ar      oscil    kEnv*p4, p5, 1
    out     ar
    print    p4, p5

endin
</CsInstruments>
<CsScore>

i2 0 0.1
i3 10 0.1
</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

*turnoff*

## Crédits

Auteur : Istvan Varga  
2005

Nouveau dans Csound 5.00

# turnon

turnon — Active un instrument pour une durée indéfinie.

## Description

Active un instrument pour une durée indéfinie.

## Syntaxe

```
turnon insnum [, itime]
```

## Initialisation

*insnum* -- numéro de l'instrument à activer

*itime* (facultatif, 0 par défaut) -- délai, en secondes, après lequel l'instrument *insnum* sera activé. Vaut 0 par défaut.

## Exécution

*turnon* active l'instrument *insnum* après un délai de *itime* secondes, ou immédiatement si *itime* n'est pas spécifié. L'instrument reste actif jusqu'à ce qu'il soit explicitement arrêté. (Voir *turnoff*)

## Voir aussi

*turnoff*, *turnoff2*

# tvconv

tvconv — Un opcode de convolution dynamique (filtre RIF).

## Description

Un opcode prenant deux signaux en entrée et interprétant l'un deux comme les coefficients à variation temporelle linéaire d'un filtre à réponse impulsionnelle finie. Réalisé via une convolution directe (pour des tailles de partition d'un échantillon) ou par une convolution partitionnée basée sur la TFD. Les signaux peuvent être "gelés" (les coefficients du filtre sont maintenus constants) à n'importe quel moment, au taux-*a* ou au taux-*k*.

## Syntaxe

```
ares tvconv asig1, asig2, xfreez1,  
          xfreez2, iparts, ifils
```

## Initialisation

*iparts* -- taille de partition, pour des tailles > 1, une convolution partitionnée basée sur la TFD est utilisée. Sinon une ligne à retard à RIF est réalisée dans le domaine temporel. Les tailles de partition > 1 sont arrondies à la puissance de deux la plus proche.

*ifils* -- taille du filtre. Pour des tailles de partition > 1, la taille du filtre est arrondie à la puissance de deux la plus proche. Lorsque le taille de partition vaut 1, comme une convolution directe est utilisée, les filtres peuvent avoir n'importe quelle taille.

## Exécution

*ares* -- sortie audio.

*asig1*, *asig2* -- entrées audio.

*xfreez1* -- indicateur de gel de *asig1*. Les coefficients ne sont mis à jour (le signal passe dans la convolution) que si *xfreez1* > 0. Cette entrée peut prendre un signal audio ou de taux-*k*, ou une constante.

*xfreez2* -- indicateur de gel de *asig2*. Fonctionne comme *xfreez1*.

## Exemples

Voici un exemple de l'opcode tvconv. Il utilise le fichier *tvconv.csd* [examples/tvconv.csd].

### Exemple 1059. Exemple de l'opcode tvconv.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
-odac  
</CsOptions>
```

```
<CsInstruments>

instr 1
  asig disk "fox.wav",1,0,1
  air disk "beats.wav",1,0,1
  air buthp air/0dbfs,1000
  k1 linseg 0,p3/3,0,0,1,2*p3/3,1
  a1 oscili k1, 0.5, 1
  a2 oscili k1, 0.6, 1
  asig tvconv asig,air,1-a2,1-a1,256,1024
  asig clip asig,1,0dbfs
  out asig
endin

</CsInstruments>
<CsScore>
f1 0 1024 7 0 512 0 1 1 511 1
i1 0 30
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pconvolve, convolve, fconv*

## Crédits

Auteur : Victor Lazzarini  
2017

Nouveau dans la version 6.09



# unirand

unirand — Générateur de nombres aléatoires de distribution uniforme (valeurs positives seulement).

## Description

Générateur de nombres aléatoires de distribution uniforme (valeurs positives seulement). C'est un générateur de bruit de classe x.

## Syntaxe

```
ares unirand krange
```

```
ires unirand krange
```

```
kres unirand krange
```

## Exécution

*krange* -- l'intervalle des nombres aléatoires (0 - *krange*).

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusets: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode unirand. Il utilise le fichier *unirand.csd* [examples/unirand.csd].

### Exemple 1060. Exemple de l'opcode unirand.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o unirand.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1    ; every run time same values

ktri unirand 100
```

```

    printk .2, ktri    ; look
    aout oscili 0.8, 440+ktri, 1    ; & listen
    outs aout, aout
    endin

    instr 2    ; every run time different values

    seed 0
    ktri unibrand 100
    printk .2, ktri    ; look
    aout oscili 0.8, 440+ktri, 1    ; & listen
    outs aout, aout
    endin

</CsInstruments>
<CsScore>
; sine wave
f 1 0 16384 10 1

i 1 0 2
i 2 3 2
e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

i 1 1 time      0.00067:      81.47237
i 1 1 time      0.20067:      41.72671
i 1 1 time      0.40067:       5.96189
i 1 1 time      0.60067:     91.59912
i 1 1 time      0.80067:     85.07127
i 1 1 time      1.00000:     92.50948
i 1 1 time      1.20067:     98.79347
i 1 1 time      1.40067:     98.91449
i 1 1 time      1.60067:     50.37808
i 1 1 time      1.80000:     72.02497
i 1 1 time      2.00000:     52.94362

WARNING: Seeding from current time 4007444022

i 2 2 time      3.00067:     91.86294
i 2 2 time      3.20067:     94.68759
i 2 2 time      3.40067:      1.05825
i 2 2 time      3.60000:     78.57628
i 2 2 time      3.80067:     27.67408
i 2 2 time      4.00000:     76.46347
i 2 2 time      4.20000:     77.10071
i 2 2 time      4.40067:     34.28921
i 2 2 time      4.60067:     37.72286
i 2 2 time      4.80067:     54.96646
i 2 2 time      5.00000:     11.67566
B 3.000 .. 5.000 T 5.000 TT 5.000 M: 0.80000 0.80000
Score finished in csoundPerform().

```

## Voir aussi

*seed, betarand, bexpnrnd, cauchy, exprand, gauss, linrand, pcauchy, poisson, trirand, weibull*

## Crédits

Auteur: Paris Smaragdis  
MIT, Cambridge

1995

# until

until — Une construction syntactique de boucle.

## Description

Une construction syntactique de boucle.

## Syntaxe

```
until condition do
... od
```

## Exécution

Les instructions entre *do* et *od* forment le corps d'une boucle qui est exécutée jusqu'à ce que la condition devienne vraie.

## Exemples

Voici un exemple de la construction until. Il utilise le fichier *until.csd* [examples/until.csd].

### Exemple 1061. Exemple de la construction until.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ifthen.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
lab99:
if p4<0 goto lab100
p4 = p4-1
print p4
goto lab99
lab100:
endin

instr 2
until p4<0 do
p4 = p4-1
print p4
od
```

```

endin
</CsInstruments>
<CsScore>
i 1 1 1 4
i 2 2 1 4
e

</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

B 0.000 .. 1.000 T 1.000 TT 1.000 M: 0.0
new alloc for instr 1:
instr 1: p4 = 3.000
instr 1: p4 = 2.000
instr 1: p4 = 1.000
instr 1: p4 = 0.000
instr 1: p4 = -1.000
B 1.000 .. 2.000 T 2.000 TT 2.000 M: 0.0
new alloc for instr 2:
instr 2: p4 = 3.000
instr 2: p4 = 2.000
instr 2: p4 = 1.000
instr 2: p4 = 0.000
instr 2: p4 = -1.000
B 2.000 .. 3.000 T 3.000 TT 3.000 M: 0.0

```

## Voir aussi

*loop\_ge*, *loop\_gt*, *loop\_le*, *loop\_lt* et *while*.

## Crédits

John ffitch.

Nouveau dans la version 5.14 de Csound avec le nouveau parseur.

# unwrap

unwrap — Applique une opération de dépliement à un tableau de valeurs de phase.

## Description

Applique une opération de dépliement à un vecteur de phases stockées dans un tableau. La sortie est un tableau avec les phases dans l'intervalle  $[-\pi, \pi)$ .

## Syntaxe

```
kout[] unwrap kin[]
```

## Exécution

*kout[]* -- tableau de sortie contenant les phases dépliées. Crée s'il n'existe pas.

*kin[]* -- tableau en entrée contenant le vecteur d'entrée.

## Exemples

Voici un exemple de l'opcode unwrap. Il utilise le fichier *unwrap.csd* [examples/unwrap.csd].

### Exemple 1062. Exemple de l'opcode unwrap.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
ksmps = 64

ifn1 ftgen 1, 0, 512, 7, 0, 512, 0
ifn2 ftgen 2, 0, 512, 7, 0, 512, 0

opcode PVA,k[]k[]k,aii
  asig, isize, ihop xin
  iolaps init isize/ihop
  kcnt init 0
  krow init 1
  kIn[] init isize
  kOlph[] init isize/2 + 1
  ifac = (sr/(ihop*2*$M_PI));
  iscal = (2*$M_PI*ihop/isize);
  kfl = 0
  kIn shiftin asig
  if kcnt == ihop then
    kWin[] window kIn,krow*ihop
    kSpec[] rfft kWin
    kMags[] mags kSpec
    kPha[] phs kSpec
    kDelta[] = kPha - kOlph
    kOlph = kPha
  kk = 0
```

```

    kDelta unwrap kDelta
    while kk < isize/2 do
        kPha[kk] = (kDelta[kk] + kk*iscal)*ifac
        kk += 1
    od
    krow = (krow+1)%iolaps
    kcnt = 0
    kfl = 1
endif
xout kMags,kPha,kfl
    kcnt += ksmps
endop

opcode PVS,a,k[]k[]kii
    kMags[],kFr[],kfl,isize,ihop xin
    iolaps init isize/ihop
    ifac = ihop*2*$M_PI/sr;
    iscal = sr/isize
    krow init 0
    kOla[] init isize
    kOut[][] init iolaps,isize
    kPhs[] init isize/2+1
    if kfl == 1 then
        kk = 0
        while kk < isize/2 do
            kFr[kk] = (kFr[kk] - kk*iscal)*ifac
            kk += 1
        od
        kPhs = kFr + kPhs
        kSpec[] pol2rect kMags,kPhs
        kRow[] riff kSpec
        kWin[] window kRow, krow*ihop
        kOut setrow kWin, krow
        kOla = 0
        kk = 0
        until kk == iolaps do
            kRow getrow kOut, kk
            kOla = kOla + kRow
            kk += 1
        od
        krow = (krow+1)%iolaps
    endif
    xout shiftout(kOla)/iolaps
endop

instr 1
    isi = 1024
    ihop = 128

    a1 diskin2 "fox.wav",1,0,1 ; audio input
    kMags[],kPhs[],kflg PVA a1,isi,ihop
    a2 PVS kMags,kPhs,kflg,isi,ihop
        out a2

endin

</CsInstruments>
<CsScore>
i1 0 10
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux.

## Crédits

Auteur : Victor Lazzarini  
NUI Maynooth  
2014

Nouveau dans la version 6.04



# upsamp

upsamp — Modifie un signal par sur-échantillonnage.

## Description

Modifie un signal par sur-échantillonnage.

## Syntaxe

```
ares upsamp ksig
```

## Exécution

*upsamp* convertit un signal de contrôle en signal audio. Cela est réalisé par simple répétition de la kval. *upsamp* est une forme légèrement plus efficace de l'affectation *asig = ksig*.

## Exemples

Voici un exemple de l'opcode upsamp. Il utilise le fichier *upsamp.csd* [examples/upsamp.csd].

### Exemple 1063. Exemple de l'opcode upsamp.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o upsamp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
;;with code from Steven Cook / David Akbari, Menno Knevel and Joachim Heintz

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

seed      0

opcode Decimator, a, akk ;UDO Sample rate / Bit depth reducer
;see http://www.csounds.com/udo/displayOpcode.php?opcode\_id=73
    setksmps 1
ain, kbit, ksrate xin

kbits    =      2^kbit                ;bit depth (1 to 16)
kfold    =      (sr/ksrate)           ;sample rate
kin       =      downsamp ain          ;convert to kr
kin       =      (kin+0dbfs)           ;add DC to avoid (-)
kin       =      kin*(kbits/(0dbfs*2)) ;scale signal level
kin       =      int(kin)              ;quantise
aout      =      upsamp kin            ;convert to sr
aout      =      aout*(2/kbits)-0dbfs  ;rescale and remove DC
a0ut      =      fold aout, kfold      ;resample
xout      =      a0ut
```

```

endop

instr 1 ;avoid playing this too loud

kbit      =      p4
ksr        =      44100
asig       diskin  "fox.wav", 1
aout       Decimator asig, kbit, ksr
           printks "bitrate = %d, ", 3, kbit
           printks "with samplerate = %d\\n", 3, ksr
           outs    aout*.7, aout*.7

endin

instr 2 ;moving randomly between different bit values (1 - 6)

kbit       randomi 1, 6, .5, 1
asig       diskin  "fox.wav", 1, 0, 1 ;loop play
aout       Decimator asig, kbit, 44100
           printks "bitrate = %f\\n", .3, kbit
           outs    aout*.7, aout*.7

endin

</CsInstruments>
<CsScore>
i 1 0 3 16 ;sounds allright but
i 1 + 3 5 ;it's getting worse
i 1 + 3 2 ;and worse...
i 2 9 22 ;or quality moves randomly
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*diff, downsamp, integ, interp, samphold*

# urandom

urandom — Opcodes de nombres vraiment aléatoires dans un intervalle contrôlable.

## Description

Opcodes du greffon urandom.

Opcodes de nombres vraiment aléatoires dans un intervalle contrôlable. Ces unités ne fonctionnent que sous système de type Unix et utilisent /dev/urandom pour construire les valeurs aléatoires de Csound.

## Syntaxe

```
ax urandom [imin, imax]
ix urandom [imin, imax]
kx urandom [imin, imax]
```

## Initialisation

*ix* -- valeur de sortie au taux-i.

*imin* -- valeur minimale de l'intervalle ; -1 par défaut.

*imax* -- valeur maximale de l'intervalle ; +1 par défaut.



### Notes

L'algorithme produit  $2^{64}$  valeurs possibles différentes qui sont mises à l'échelle pour s'inscrire dans l'intervalle demandé. Le hasard vient de la méthode usuelle /dev/urandom de Linux/OSX. Il n'y a aucune garantie que ce soit vraiment aléatoire, mais il y a de grandes chances. Il ne produit pas de valeurs cycliques.

## Exécution

*ax* -- valeur de sortie au taux-a.

*kx* -- valeur de sortie au taux-k.

## Exemples

Voici un exemple de l'opcode urandom au taux-a. Il utilise le fichier *urandom.csd* [examples/urandom.csd].

### Exemple 1064. Exemple de l'opcode urandom au taux-a.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
```

```

; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rnd31.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create random numbers at a-rate in the range -2 to 2
aur urandom -2, 2

; Use the random numbers to choose a frequency.
afreq = aur * 500 + 100

a1 oscil 30000, afreq, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>

```

Voici un exemple de l'opcode urandom au taux-k. Il utilise le fichier *urandom\_krate.csd* [exemples/urandom\_krate.csd].

### Exemple 1065. Exemple de l'opcode urandom au taux-k.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o rnd31_krate.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
; Create random numbers at k-rate in the range -1 to 1

```

```
; with a uniform distribution.
k1 urandom

printks "k1=%f\\n", 0.1, k1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
e

</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
k1=0.229850
k1=-0.077047
k1=-0.199339
k1=-0.620577
k1=-0.119447
k1=-0.596258
k1=0.525800
k1=-0.171583
k1=-0.017196
k1=-0.974613
k1=-0.036276
```

## Crédits

Auteur : John ffitch

Nouveau dans la version 5.13

# urd

**urd** — Un générateur de nombres aléatoires de distribution discrète définie par l'utilisateur que l'on peut utiliser comme une fonction.

## Description

Un générateur de nombres aléatoires de distribution discrète définie par l'utilisateur que l'on peut utiliser comme une fonction.

## Syntaxe

```
aout = urd(ktableNum)
iout = urd(itableNum)
kout = urd(ktableNum)
```

## Initialisation

*itableNum* -- numéro d'une table contenant la fonction de la distribution aléatoire. Cette table est générée par l'utilisateur. Voir GEN40, GEN41 et GEN42. La longueur de la table peut être différente d'une puissance de 2.

## Exécution

*ktableNum* -- numéro d'une table contenant la fonction de la distribution aléatoire. Cette table est générée par l'utilisateur. Voir GEN40, GEN41 et GEN42. La longueur de la table peut être différente d'une puissance de 2.

*urd* est le même opcode que *duserrnd*, mais on peut l'utiliser à la manière d'une fonction.

Pour un tutoriel sur les histogrammes et les fonctions de distribution aléatoires consulter :

- D. Lorrain. "A panoply of stochastic cannons". In C. Roads, ed. 1989. Music machine. Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode *urd*. Il utilise le fichier *urd.csd* [examples/urd.csd].

### Exemple 1066. Exemple de l'opcode *urd*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
```

```

; -o urd.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ktab = 1 ;ftable 1
kurd = urd(ktab)
ktrig metro 5 ;triggers 5 times per second
kres samphold kurd, ktrig ;sample and hold value of kurd
      printk2 kres ;print it
asig poscil .5, 220+kres, 2
      outs asig, asig
endin

instr 2

seed 0 ;every run new values

ktab = 1 ;ftable 1
kurd = urd(ktab)
ktrig metro 5 ;triggers 5 times per second
kres samphold kurd, ktrig ;sample and hold value of kurd
      printk2 kres ;print it
asig poscil .5, 220+kres, 2
      outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 -20 -42 10 20 .3 100 200 .7 ;30% choose between 10 and 20 and 70% between 100 and 200
f2 0 8192 10 1 ;sine wave

i 1 0 5
i 2 6 5
e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

i1 184.61538
i1 130.76923
i1 169.23077
i1 12.00000
.....

WARNING: Seeding from current time 3751086165

i2 138.46154
i2 12.00000
i2 123.07692
i2 161.53846
i2 123.07692
i2 153.84615
.....

```

## Voir aussi

*cuserrnd, duserrnd*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.16



# vactrol

vactrol — Générateur unitaire suiveur d'enveloppe.

## Description

Opcode du greffon buchla.

Générateur unitaire suiveur d'enveloppe simulant un Vactrole Perkin Elmer VTL5C3/2.

## Syntaxe

```
ares vactrol asig [iup, idown]
```

## Initialisation

*iup* -- Le temps de montée du filtre qui vaut 20 par défaut.

*idown* -- Le temps de chute du filtre qui vaut 3000 par défaut.

## Exécution

*asig* -- Le signal dont on extrait l'enveloppe.

## Exemples

Voici un exemple de l'opcode vactrol. Il utilise le fichier *vactrol.csd* [examples/vactrol.csd].

### Exemple 1067. Exemple de l'opcode vactrol.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>

<CsInstruments>
    nchnls = 2
    0dbfs = 1

    instr 1
        a1 lfo 0.3, 1, 4
        a2 vactrol a1
        a3 oscili 2, 440
        out a1*a3,a2*a3
    endin

</CsInstruments>

<CsScore>
    i1 0 3
    e
</CsScore>

</CsoundSynthesizer>
```

Cet opcode peut réduire le bruit de frottement produit par l'extraction d'une enveloppe complexe, car il adoucit les transitions.

## Crédits

Auteur : John ffitch  
D'après Julian Parker  
Nouveau dans la version 6.04

# vadd

**vadd** — Ajoute une valeur scalaire à un vecteur dans une table.

## Description

Ajoute une valeur scalaire à un vecteur dans une table.

## Syntaxe

```
vadd ifn, kval, kelements [, kdstoffset] [, kverbose]
```

## Initialisation

*ifn* - numéro de la table hébergeant le signal vectoriel à traiter.

## Exécution

*kval* - valeur scalaire à ajouter.

*kelements* - nombre de composantes du vecteur.

*kdstoffset* - décalage d'indexation pour la table de destination (facultatif, vaut 0 par défaut).

*kverbose* - Indique si les avertissements sont affichés (vaut 0 par défaut).

**vadd** ajoute la valeur de *kval* à chaque composante du vecteur contenu dans la table *ifn*, à partir de l'index de table *kdstoffset*. Cela permet de traiter une section particulière d'une table en spécifiant le décalage et le nombre d'éléments à traiter. Le décalage est compté à partir de 0, si bien que si aucun décalage n'est spécifié (ou s'il est fixé à 0), la table est modifiée depuis le début.

Noter que cet opcode est exécuté au taux-k si bien que la valeur de *kval* est ajoutée à chaque période de contrôle. A utiliser avec précaution si l'on ne veut pas finir avec des nombres très grands (ou utiliser *vadd\_i*).

Ces opcodes (*vadd*, *vmult*, *vpow* et *vexp*) réalisent des opérations numériques entre un signal vectoriel de contrôle (hébergé par la table *ifn*), et un signal scalaire (*kval*). Le résultat est un nouveau vecteur qui écrase les anciennes valeurs de *ifn*. Tous ces opcodes travaillent au taux-k.

Les valeurs négatives sont valides pour *kdstoffset*. Les composantes du vecteur se trouvant en dehors de la table sont alors ignorées, et elles ne sont pas repliées autour de la table.

Si l'argument facultatif *kverbose* est différent de 0, l'opcode affichera des messages d'avertissement à chaque passe-k si les longueurs de table sont dépassées.

Dans tous ces opcodes, les vecteurs résultants sont stockés dans *ifn*, écrasant les vecteurs initiaux. Si l'on veut garder le vecteur initial, il faut utiliser *vcopy* ou *vcopy\_i* pour le copier dans une autre table. Tous ces opérateurs sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc. Ils peuvent aussi être utiles en conjonction avec les opcodes spectraux *pvsftw* et *pvsftr*.



## Note

Prière de noter que l'argument *elements* a changé dans la version 5.03 du taux-i au taux-k. Cela change le comportement de l'opcode dans le cas inhabituel où la variable de taux-i *ielements* est modifiée à l'intérieur de l'instrument, par exemple dans :

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

## Exemples

Voici un exemple de l'opcode vadd. Il utilise le fichier *vadd.csd* [examples/vadd.csd].

### Exemple 1068. Exemple de l'opcode vadd.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vadd ifn1, ival, ielements, idstoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
turnoff
endif

kcount = kcount + 1
endin

</CsInstruments>
```

```
<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 5 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 8 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1 10 12
i2 1.6 0.2 1
e

</CsScore>

</CsoundSynthesizer>
```

## Voir aussi

*vadd\_i*, *vmult*, *vpow* et *vexp*.

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vadd\_i

`vadd_i` — Ajoute une valeur scalaire à un vecteur dans une table.

## Description

Ajoute une valeur scalaire à un vecteur dans une table.

## Syntaxe

```
vadd_i ifn, ival, ielements [, idstoffset]
```

## Initialisation

*ifn* - numéro de la table hébergeant le signal vectoriel à traiter.

*ival* - valeur scalaire à ajouter.

*ielements* - nombre de composantes du vecteur.

*idstoffset* - décalage d'indexation pour la table de destination (facultatif, vaut 0 par défaut).

## Exécution

`vadd_i` ajoute la valeur de *ival* à chaque composante du vecteur contenu dans la table *ifn*, en partant de l'index de table *idstoffset*. Cela permet de traiter une section particulière d'une table en spécifiant le décalage et le nombre d'éléments à traiter. Le décalage est compté à partir de 0, si bien que si aucun décalage n'est spécifié (ou s'il est fixé à 0), la table est modifiée depuis le début.

Cet opcode n'est exécuté qu'à l'initialisation. Il y a une version de taux-k de cet opcode appelée `vadd`.

Les valeurs négatives sont valides pour *idstoffset*. Les composantes du vecteur se trouvant en dehors de la table sont alors ignorées, et elles ne sont pas repliées autour de la table.

Dans tous ces opcodes, les vecteurs résultants sont stockés dans *ifn*, écrasant les vecteurs initiaux. Si l'on veut garder le vecteur initial, il faut utiliser `vcopy` ou `vcopy_i` pour le copier dans une autre table. Tous ces opérateurs sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que `vcella`, `adsynt`, `adsynt2`, etc. Ils peuvent aussi être utiles en conjonction avec les opcodes spectraux `pvsftw` et `pvsftr`.

## Exemples

Voici un exemple de l'opcode `vadd_i`. Il utilise le fichier `vadd_i.csd` [exemples/vadd\_i.csd].

### Exemple 1069. Exemple de l'opcode `vadd_i`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>
```

```

; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vadd_i ifn1, ival, ielements, idstoffset
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
  turnoff
endif

kcount = kcount + 1
endin

</CsInstruments>

<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
e

</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

*vadd*, *vmult\_i*, *vpow\_i* et *vexp\_i*.

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vaddv

vaddv — Addition entre deux signaux vectoriels de contrôle.

## Description

Addition entre deux signaux vectoriels de contrôle.

## Syntaxe

```
vaddv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
```

## Initialisation

*ifn1* - numéro de la table hébergeant le premier vecteur à traiter.

*ifn2* - numéro de la table hébergeant le second vecteur à traiter.

## Exécution

*kelements* - nombre de composantes des deux vecteurs.

*kdstoffset* - décalage d'indexation pour la table de destination *ifn1* (vaut 0 par défaut).

*ksrcoffset* - décalage d'indexation pour la table source *ifn2* (vaut 0 par défaut).

*kverbose* - Indique si les avertissements sont affichés (vaut 0 par défaut).

*vaddv* additionne deux signaux vectoriels de contrôle, chaque composante du premier vecteur n'étant traitée qu'avec la composante correspondante de l'autre vecteur. Chaque signal vectoriel est hébergé dans une table (*ifn1* et *ifn2*). Le nombre de composantes de chaque vecteur doit être identique.

Le résultat est un nouveau signal vectoriel de contrôle qui écrase les anciennes valeurs de *ifn1*. Si l'on veut garder l'ancien vecteur *ifn1*, il faut utiliser l'opcode *vcopy\_i* pour le copier dans une autre table. On peut utiliser *kdstoffset* et *ksrcoffset* pour spécifier des vecteurs à n'importe quelle position dans les tables.

Des valeurs négatives pour *kdstoffset* et *ksrcoffset* sont acceptables. Si *kdstoffset* est négatif, la partie du vecteur hors-limites est ignorée. Si *ksrcoffset* est négatif, les éléments hors-limites seront supposés valoir 0 (c'est-à-dire que les éléments de destination ne seront pas changés). Si des éléments pour le vecteur de destination sont au-delà de la taille de la table (point de garde inclus), ces éléments sont ignorés (les éléments ne sont pas repliés autour des tables). Si des éléments pour le vecteur source sont au-delà de la longueur de la table, ces éléments sont supposés valoir 0 (le vecteur de destination ne sera pas changé pour ces éléments).



### Avertissement

L'utilisation de la même table comme source et comme destination dans les versions antérieures à la 5.04 peut induire un comportement imprévu. A utiliser avec précaution.

Cet opcode travaille au taux-k (cela signifie qu'à chaque passe-k les vecteurs sont additionnés). Il y a une version de taux-i de cet opcode appelée *vaddv\_i*.





## Note

Prière de noter que l'argument *elements* a changé dans la version 5.03 du taux-i au taux-k. Cela change le comportement de l'opcode dans le cas inhabituel où la variable de taux-i *ielements* est modifiée à l'intérieur de l'instrument, par exemple dans :

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

Tous ces opérateurs (*vaddv*, *vsubv*, *vmultv*, *vddivv*, *vpowv*, *vexpv*, *vcopy* et *vmap*) sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2* etc.

## Exemples

Voici un exemple de l'opcode *vaddv*. Il utilise le fichier *vaddv.csd* [examples/vaddv.csd].

### Exemple 1070. Exemple de l'opcode *vaddv*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc          -nm0   ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o cigoto.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

opcode TableDumpSimp, 0, ijo
;prints the content of a table in a simple way
ifn, iprec, ippr xin; function table, float precision while printing (default = 3), parameters per row
iprec = (iprec == -1 ? 3 : iprec)
ippr = (ippr == 0 ? 10 : ippr)
iend = ftlen(ifn)
indx = 0
Sformat sprintf "%%.%df\t", iprec
Sdump = ""
loop:
ival tab_i indx, ifn
Snew sprintf Sformat, ival
Sdump strcat Sdump, Snew
indx = indx + 1
imod = indx % ippr
if imod == 0 then
puts Sdump, 1
Sdump = ""
endif
if indx < iend igoto loop
```

```

    puts Sdump, 1
endop

instr 1
ifn1 = p4
ifn2 = p5
ielements = p6
idstoffset = p7
isrcoffset = p8
kval init 25
vaddv ifn1, ifn2, ielements, idstoffset, isrcoffset, 1
turnoff
endin

instr 2
TableDumpSimp p4, 3, 16
endin

</CsInstruments>
<CsScore>

f 1 0 16 -7 1 15 16

f 2 0 16 -7 1 15 2

i2 0.0 0.2 1
i2 0.2 0.2 2
i1 0.4 0.01 1 2 5 3 8
i2 0.8 0.2 1
i1 1.0 0.01 1 2 5 10 -2
i2 1.2 0.2 1
i1 1.4 0.01 1 2 8 14 0
i2 1.6 0.2 1
i1 1.8 0.01 1 2 8 0 14
i2 2.0 0.2 1
i1 2.2 0.002 1 1 8 5 2
i2 2.4 0.2 1
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vaddv\_i

vaddv\_i — Addition entre deux signaux vectoriels de contrôle à l'initialisation.

## Description

Addition entre deux signaux vectoriels de contrôle à l'initialisation.

## Syntaxe

```
vaddv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
```

## Initialisation

*ifn1* - numéro de la table hébergeant le premier vecteur à traiter.

*ifn2* - numéro de la table hébergeant le second vecteur à traiter.

*ielements* - nombre de composantes des deux vecteurs.

*idstoffset* - décalage d'indexation pour la table de destination *ifn1* (vaut 0 par défaut).

*isrcoffset* - décalage d'indexation pour la table source *ifn2* (vaut 0 par défaut).

## Exécution

*vaddv\_i* additionne deux signaux vectoriels de contrôle, chaque composante du premier vecteur n'étant traitée qu'avec la composante correspondante de l'autre vecteur. Chaque signal vectoriel est hébergé dans une table (*ifn1* et *ifn2*). Le nombre de composantes de chaque vecteur doit être identique.

Le résultat est un nouveau signal vectoriel de contrôle qui écrase les anciennes valeurs de *ifn1*. Si l'on veut garder l'ancien vecteur *ifn1*, il faut utiliser l'opcode *vcopy\_i* pour le copier dans une autre table. On peut utiliser *idstoffset* et *isrcoffset* pour spécifier des vecteurs à n'importe quelle position dans les tables.

Des valeurs négatives pour *idstoffset* et *isrcoffset* sont acceptables. Si *idstoffset* est négatif, la partie du vecteur hors-limites est ignorée. Si *isrcoffset* est négatif, les éléments hors-limites seront supposés valoir 0 (c'est-à-dire que les éléments de destination ne seront pas changés). Si des éléments pour le vecteur de destination sont au-delà de la taille de la table (point de garde inclus), ces éléments sont ignorés (les éléments ne sont pas repliés autour des tables). Si des éléments pour le vecteur source sont au-delà de la longueur de la table, ces éléments sont supposés valoir 0 (le vecteur de destination ne sera pas changé pour ces éléments).



### Avertissement

L'utilisation de la même table comme source et comme destination dans les versions antérieures à la 5.04 peut induire un comportement imprévu. A utiliser avec précaution.

Cet opcode travaille au taux-i. Il y a une version de taux-k de cet opcode appelée *vaddv*.

Tous ces opérateurs (*vaddv\_i*, *vsubv\_i*, *vmultv\_i*, *vdivv\_i*, *vpowv\_i*, *vexpv\_i*, *vcopy* et *vmap*) sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2* etc.

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vaget

vaget — Accès aux valeurs du tampon courant d'une variable de taux-a par indexation.

## Description

Accès aux valeurs du tampon courant d'une variable de taux-a par indexation. Utile pour effectuer des manipulations échantillon par échantillon au taux-k sans recourir à *setksmps* 1.



### Note

Comme cet opcode ne vérifie pas les limites d'indexation, il faut faire attention à ne pas essayer de lire des valeurs au-delà de *ksmps* (la taille du tampon d'une variable de taux-a) en utilisant des valeurs d'indice supérieures à *ksmps*.

## Syntaxe

```
kval vaget kndx, avar
```

## Exécution

*kval* - valeur lue depuis *avar*

*kndx* - indice de l'échantillon à lire dans le tampon de la variable *avar* donnée

*avar* - variable de taux-a dont on veut lire les valeurs

## Exemples

Voici un exemple de l'opcode vaget. Il utilise le fichier *vaget.csd* [examples/vaget.csd].

### Exemple 1071. Exemple de l'opcode vaget.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o avarget.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr=44100
ksmps=16
nchnls=2

instr 1 ; Sqrt Signal
ifreq = (p4 > 15 ? p4 : cpspch(p4))
iamp = ampdb(p5)

aout init 0
```

```
ksampnum init 0

kenv linseg 0, p3 * .5, 1, p3 * .5, 0

aout1 vco2 1, ifreq
aout2 vco2 .5, ifreq * 2
aout3 vco2 .2, ifreq * 4

aout sum aout1, aout2, aout3

;Take Sqrt of signal, checking for negatives
kcount = 0

loopStart:

kval vaget kcount, aout

if (kval > .0) then
  kval = sqrt(kval)
elseif (kval < 0) then
  kval = sqrt(-kval) * -1
else
  kval = 0
endif

vaset kval, kcount, aout

loop_lt kcount, 1, ksmps, loopStart

aout = aout * kenv

aout moogladder aout, 8000, .1

aout = aout * iamp

outs aout, aout
endin

</CsInstruments>

<CsScore>

i1 0.0 2 440 80
e

</CsScore>

</CsoundSynthesizer>
```

## Voir aussi

*vaset*

## Crédits

Auteur : Steven Yi

Nouveau dans la version 5.04

Septembre 2006.

# valpass

valpass — Réverbération variable du signal en entrée avec une réponse en fréquence plate.

## Description

Réverbération variable du signal en entrée avec une réponse en fréquence plate.

## Syntaxe

```
ares valpass asig, krvt, xlpt, imaxlpt [, iskip] [, insmps]
```

## Initialisation

*imaxlpt* -- durée de boucle maximale pour *klpt*

*iskip* (facultatif, 0 par défaut) -- état initial de l'espace de données de la boucle de retard (cf. *reson*). La valeur par défaut est 0.

*insmps* (facultatif, 0 par défaut) -- valeur du retard, en nombre d'échantillons.

## Exécution

*krvt* -- la durée de réverbération (définie comme le temps en secondes pris par un signal pour décroître à 1/1000 ou 60 dB de son amplitude originale).

*xlpt* -- durée de boucle variable en secondes, comme *ilpt* dans *comb*. La durée de boucle peut aller jusqu'à *imaxlpt*.

Ce filtre répète l'entrée avec une densité d'écho déterminée par la durée de boucle *xlpt*. Le taux d'atténuation est indépendant et il est déterminé par *krvt*, la durée de réverbération (définie comme le temps en secondes pris par un signal pour décroître à 1/1000 ou 60 dB de son amplitude originale). La sortie apparaît sans retard.

## Exemples

Voici un exemple de l'opcode valpass. Il utilise le fichier *valpass.csd* [examples/valpass.csd].

### Exemple 1072. Exemple de l'opcode valpass.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o valpass.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

krvt = 1.5
klpt line p4, p3, p5
imaxlpt = .1

a1 diskin2 "fox.wav", 1
a1 valpass a1, krvt, klpt, imaxlpt
a2 valpass a1, krvt, klpt*.5, imaxlpt
  outs a1, a2

endin
</CsInstruments>
<CsScore>

i 1 0 5 .01 .2
e
</CsScore>
</CsoundSynthesizer>

```

Voici un autre exemple de l'opcode valpass. Il utilise le fichier *valpass-2.csd* [examples/valpass-2.csd].

### Exemple 1073. Second exemple de l'opcode valpass.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o valpass-2.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giSine ftgen 0, 0, 65536, 10, 1   ;sine wave

instr 1

asig diskin2 "beats.wav", 1, 0, 1
krvt line 0.01, p3, p3   ;reverb time
adepth = p4   ;sine depth
krate = 0.3   ;sine rate (speed)
adel oscil 0.5, krvt, giSine   ;delay time oscillator (LFO)
adel = ((adel+0.5)*adepth)   ;scale and offset LFO
aout valpass asig, krvt, adel*0.01, 0.5
  outs aout, aout

endin
</CsInstruments>
<CsScore>

i1 0 10 1
i1 11 10 5
e

```



```
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*alpass, comb, reverb, vcomb*

## Crédits

Auteur : William « Pete » Moss  
Université du Texas à Austin  
Austin, Texas USA  
Janvier 2002

# vaset

vaset — Ecrit une valeur dans le tampon courant d'une variable de taux-a par indexation.

## Description

Ecrit une valeur dans le tampon courant d'une variable de taux-a à la position donnée. Utile pour effectuer des manipulations échantillon par échantillon au taux-k sans recourir à *setksmps* 1.



### Note

Comme cet opcode ne vérifie pas les limites d'indexation, il faut faire attention à ne pas essayer d'écrire une valeur au-delà de *ksmps* (la taille du tampon d'une variable de taux-a) en utilisant des valeurs d'indice supérieures à *ksmps*.

## Syntaxe

```
vaset kval, kndx, avar
```

## Exécution

*kval* - valeur à écrire dans *avar*

*kndx* - indice de l'échantillon à écrire dans le tampon de la variable *avar* donnée

*avar* - variable de taux-a dans laquelle écrire

## Exemples

Voici un exemple de l'opcode vaset. Il utilise le fichier *vaset.csd* [examples/vaset.csd].

### Exemple 1074. Exemple de l'opcode vaset.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o avarset.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr=44100
ksmps=1
nchnls=2

instr 1 ; Sine Wave
ifreq = (p4 > 15 ? p4 : cpspch(p4))
iamp = ampdb(p5)

kenv adsr 0.1, 0.05, .9, 0.2
```

```

aout init 0
ksampnum init 0

kcount = 0

iperiod = sr / ifreq

i2pi = 3.14159 * 2

loopStart:

kphase = (ksampnum % iperiod) / iperiod
knewval = sin(kphase * i2pi)

vaset knewval, kcount, aout

ksampnum = ksampnum + 1

loop_lt kcount, 1, ksmps, loopStart

aout = aout * iamp * kenv

outs aout, aout
endin

</CsInstruments>

<CsScore>

i1 0.0 2 440 80
e

</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

*vaget*

## Crédits

Auteur : Steven Yi

Nouveau dans la version 5.04

Septembre 2006.

# vbap

vbap — Distribue un signal audio sur plusieurs canaux.

## Description

Distribue un signal audio sur plusieurs canaux, jusqu'à 64 dans la première forme, en nombre arbitraire dans la seconde.

## Syntaxe

```
ar1[, ar2...] vbap asig, kazim \
    kelev [, kspread] [, ilayout]

array[] vbap asig, kazim [,
    kelev] [, kspread] [, ilayout]
```

## Initialisation

*ilayout* -- indice de la disposition des haut-parleurs dans l'intervalle 0-99, correspondant à un appel à *vbaplsinit*.

## Exécution

*asig* -- signal audio à traiter.

*kazim* -- angle d'azimut de la source virtuelle.

*kelev* (facultatif) -- angle d'élévation de la source virtuelle.

*kspread* (facultatif) -- diffusion de la source virtuelle (de 0 à 100). S'il vaut 0, on a un panoramique d'amplitude conventionnel. Plus *kspread* augmente et plus le nombre de haut-parleurs utilisés dans le panoramique augmente. S'il vaut 100, le son est appliqué à tous les haut-parleurs.

*vbap* prend un signal en entrée, *asig*, et le distribue sur les sorties en fonction des contrôles *kazim* et *kelev*, et de la disposition des haut-parleurs. Si *idim* = 2, *kelev* est mis à zéro. La distribution est réalisée par Panoramique d'Amplitude sur une Base de Vecteurs (VBAP - voir référence). VBAP distribue le signal en tenant compte des données de haut-parleurs configurées avec *vbaplsinit*. Le signal est appliqué au plus à deux haut-parleurs dans les configurations 2D et à trois haut-parleurs dans les configurations 3D. Si la source virtuelle est distribuée en dehors de la région couverte par les haut-parleurs, les haut-parleurs les plus proches sont utilisés dans le panoramique.



### Avertissement

Prière de noter que tous les opcodes de panoramique *vbap* nécessitent une initialisation du système *vbap* avec *vbaplsinit*.

## Exemples

Voir l'entrée sur *vbap8* pour un exemple d'utilisation des opcodes *vbap*.

## Référence

Ville Pulkki : « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, juin 1997, Vol. 45/6, p. 456.

## Exemples

Voici un exemple de l'opcode vbap. Il utilise le fichier *vbap.csd* [examples/vbap.csd].

### Exemple 1075. Exemple de l'opcode vbap.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime output leave only the line below:
; -o vbap.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 4 ;quad
0dbfs = 1

vbaplsinit 2, 4, 0, 90, 180, 270

instr 1

asig diskin2 "beats.wav", 1, 0, 1   ;loop beats.wav
kaz line 0, p3, p4   ;come from right rear speaker &
a1,a2,a3,a4 vbap asig, 180, 100, kaz   ;change spread of soundsource
printks "spread of source = %d\n", 1, kaz ;print spread value
outq a1,a2,a3,a4

endin
</CsInstruments>
<CsScore>

i 1 0 12 100

e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
spread of source = 0
spread of source = 8
spread of source = 17
spread of source = 25
spread of source = 33
spread of source = 42
spread of source = 50
spread of source = 58
spread of source = 67
spread of source = 75
```

```
spread of source = 83  
spread of source = 92  
spread of source = 100
```

## Voir aussi

*vbapmove, vbap16, vbap16move, vbap4, vbap4move, vbap8, vbap8move, vbaplsinit, vbapz, vbapzmove*

## Crédits

Auteur : Ville Pulkki  
Sibelius Academy Computer Music Studio  
Laboratoire d'Acoustique et de Traitement du Signal Audio  
Helsinki, Université de Technologie  
Helsinki, Finlande  
Mai 2000  
Auteur : John ffitch  
Juillet 2012, septembre 2013

Nouveau dans la version 5.17.13 de Csound.

Variante pour tableau ajoutée dans la version 6.01

# vbapmove

vbapmove — Distribue un signal audio sur plusieurs canaux avec des sources virtuelles en mouvement.

## Description

Distribue un signal audio sur un maximum de 64 canaux avec des sources virtuelles en mouvement.

## Syntaxe

```
ar1[, ar2...] vbapmove asig, idur, ispread, ifldnum, ifld1 \
    [, ifld2] [...]
```

```
aarray[] vbapmove asig, idur, ispread, ifldnum, ifld1 \
    [, ifld2] [...]
```

## Initialisation

*idur* -- durée pendant laquelle le mouvement a lieu.

*ispread* -- diffusion de la source virtuelle (de 0 à 100). S'il vaut 0, on a un panoramique d'amplitude conventionnel. Plus *ispread* augmente et plus le nombre de haut-parleurs utilisés dans le panoramique augmente. S'il vaut 100, le son est appliqué à tous les haut-parleurs.

*ifldnum* -- nombre de champs (sa valeur absolue doit être supérieure ou égale à 2). Si *ifldnum* est positif, le mouvement de la source virtuelle est une ligne brisée spécifiée par les directions données. Chaque transition est exécutée durant un intervalle de même durée. Si *ifldnum* est négatif, les vitesses angulaires spécifiées sont appliquées à la source virtuelle durant les intervalles de temps spécifiés correspondants (voir ci-dessous).

*ifld1*, *ifld2*, ... -- angles d'azimut ou vitesses angulaires et durées correspondantes des phases du mouvement (voir ci-dessous).

## Exécution

*asig* -- signal audio à traiter.

*vbapmove* permet l'utilisation de sources virtuelles en mouvement. Si *ifldnum* est positif, les champs représentent les directions de la source virtuelle durant des intervalles de temps égaux, *iazi1*, [*iele1*,] *iazi2*, [*iele2*,], etc. La position de la source virtuelle est interpolée entre ces directions en partant de la première direction et en terminant à la dernière. Chaque intervalle est interpolé durant une fraction de la durée de l'évènement sonore égale à *durée\_totale* / *nombre\_intervalles*.

Si *ifldnum* est négatif, les champs représentent les vitesses angulaires à intervalles réguliers. Le premier champ est cependant la direction de départ, *iazi1*, [*iele1*,] *iazi\_vel1*, [*iele\_vel1*,] *iazi\_vel2*, [*iele\_vel2*,] ... Chaque vitesse est appliquée à la note qui occupe la fraction *durée\_totale* / *nombre\_de\_vitesses* de la durée de l'évènement sonore. Si l'élévation de la source virtuelle dépasse 90 degrés ou devient inférieure à 0 degré, la polarité de la vitesse angulaire change. Ainsi l'élévation angulaire produit une source virtuelle qui monte et descend entre 0 et 90 degrés.



### Avertissement

Prière de noter que tous les opcodes de panoramique *vbap* nécessitent une initialisation du système *vbap* avec *vbaplsinit*.

## Exemples

Voir l'entrée sur *vbap8move* pour un exemple d'utilisation des opcodes *vbapXmove*.

## Référence

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, juin 1997, Vol. 45/6, p. 456.

## Exemples

Voici un exemple de l'opcode *vbapmove*. Il utilise le fichier *vbapmove.csd* [examples/vbapmove.csd].

### Exemple 1076. Exemple de l'opcode *vbapmove*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vbapmove.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 4 ;quad
0dbfs = 1

vbaplsinit 2, 4, 0, 90, 180, 270

instr 1

asig diskin2 "beats.wav", 1, 0, 1 ;loop beats.wav
a1,a2,a3,a4 vbapmove asig, p3, 1, 2, 310, 180 ;change movement of soundsource in
        outq a1,a2,a3,a4 ;the rear speakers

endin
</CsInstruments>
<CsScore>

i 1 0 5

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*vbap*, *vbap16*, *vbap16move*, *vbap4*, *vbap4move*, *vbap8*, *vbap8move*, *vbaplsinit*, *vbapz*, *vbapzmove*

## Crédits

Auteur : Ville Pulkki



Sibelius Academy Computer Music Studio  
Laboratoire d'Acoustique et de Traitement du Signal Audio  
Helsinki, Université de Technologie  
Helsinki, Finlande  
Mai 2000  
Auteur : John ffitch  
Juillet 2012, septembre 2013

Nouveau dans la version 5.17.13 de Csound.

Sortie sur tableau ajoutée dans la version 6.01

# vbapg

vbapg — Calcule les gains pour un positionnement du son entre des canaux multiples.

## Description

Calcule les gains pour un positionnement du son entre plusieurs canaux, jusqu'à 64.

## Syntaxe

```
k1[, k2...] vbapg kazim [,kelev] [, kspread] [, ilayout]  
karray[] vbapg kazim [,kelev] [, kspread] [, ilayout]
```

## Initialisation

*ilayout* -- indice de la disposition des haut-parleurs dans l'intervalle 0-99, correspondant à un appel à *vbaplsinit*. Vaut 0 par défaut.

## Exécution

*kazim* -- angle d'azimut de la source virtuelle.

*kelev* (facultatif) -- angle d'élévation de la source virtuelle.

*kspread* (facultatif) -- diffusion de la source virtuelle (de 0 à 100). S'il vaut 0, on a un panoramique d'amplitude conventionnel. Plus *kspread* augmente et plus le nombre de haut-parleurs utilisés dans le panoramique augmente. S'il vaut 100, le son est appliqué à tous les haut-parleurs.

*vbapg* calcule les gains qu'un signal d'entrée aurait entre plusieurs haut-parleurs en fonction des contrôles *kazim* et *kelev*, et de la disposition des haut-parleurs *ilayout*. Si *idim* = 2, *kelev* est mis à zéro. La distribution est réalisée par Panoramique d'Amplitude sur une Base de Vecteurs (VBAP - voir référence). VBAP distribue le signal en tenant compte des données de haut-parleurs configurées avec *vbaplsinit*. Le signal est appliqué au plus à deux haut-parleurs dans les configurations 2D et à trois haut-parleurs dans les configurations 3D. Si la source virtuelle est distribuée en dehors de la région couverte par les haut-parleurs, les haut-parleurs les plus proches sont utilisés dans le panoramique.



### Avertissement

Prière de noter que tous les opcodes de panoramique *vbap* nécessitent une initialisation du système *vbap* avec *vbaplsinit*.

## Exemples

## Référence

Ville Pulkki : « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, juin 1997, Vol. 45/6, p. 456.

## Exemples

Voici un exemple de l'opcode *vbapg*. Il utilise le fichier *vbapg.csd* [examples/vbapg.csd].

## Exemple 1077. Exemple de l'opcode vbapg.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vbap4.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 4 ;quad
0dbfs = 1

vbaplsinit 2.01, 4, 0, 90, 180, 270

instr 1

asig diskin2 "beats.wav", 1, 0, 1 ;loop beats.wav
kaz line 0, p3, p4 ;come from right rear speaker &
k1,k2,k3,k4 vbapg 180, 100, kaz, 1 ;change spread of soundsource
printks "spread of source = %d\n", 1, kaz ;print spread value
outq asig*k1,asig*k2,asig*k3,asig*k4

endin
</CsInstruments>
<CsScore>

i 1 0 12 100

e
</CsScore>
</CsoundSynthesizer>
```

La sortie contiendra des lignes comme celles-ci :

```
spread of source = 0
spread of source = 8
spread of source = 17
spread of source = 25
spread of source = 33
spread of source = 42
spread of source = 50
spread of source = 58
spread of source = 67
spread of source = 75
spread of source = 83
spread of source = 92
spread of source = 100
```

## Voir aussi

*vbap*, *vbapmove*, *vbap4*, *vbap4move*, *vbap8*, *vbap8move*, *vbap16*, *vbap16move*, *vbaplsinit*, *vbapz*, *vbapz-move*

## Credits

Auteur : Ville Pulkki  
Sibelius Academy Computer Music Studio  
Laboratoire d'Acoustique et de Traitement du Signal Audio  
Helsinki, Université de Technologie  
Helsinki, Finlande  
Mai 2000  
Auteur : John ffitch  
Juillet 2012, septembre 2013

Nouveau dans la version 5.17.13 de Csound.

Variante pour tableau ajoutée dans la version 6.01

# vbapgmmove

**vbapgmmove** — Calcule les gains pour positionner un son entre plusieurs canaux avec des sources virtuelles en mouvement.

## Description

Calcule les gains pour positionner un son entre plusieurs canaux avec des sources virtuelles en mouvement.

## Syntaxe

```
kr1[, kr2...] vbapgmmove idur, ispread, ifldnum, ifld1 \
[, ifld2] [...]\n\nkarray[] vbapgmmove idur, ispread, ifldnum, ifld1 \
[, ifld2] [...]
```

## Initialisation

*idur* -- durée pendant laquelle le mouvement a lieu.

*ispread* -- diffusion de la source virtuelle (de 0 à 100). S'il vaut 0, on a un panoramique d'amplitude conventionnel. Plus *ispread* augmente et plus le nombre de haut-parleurs utilisés dans le panoramique augmente. S'il vaut 100, le son est appliqué à tous les haut-parleurs.

*ifldnum* -- nombre de champs (sa valeur absolue doit être supérieure ou égale à 2). Si *ifldnum* est positif, le mouvement de la source virtuelle est une ligne brisée spécifiée par les directions données. Chaque transition est exécutée durant un intervalle de même durée. Si *ifldnum* est négatif, les vitesses angulaires spécifiées sont appliquées à la source virtuelle durant les intervalles de temps spécifiés correspondants (voir ci-dessous).

*ifld1, ifld2, ...* -- angles d'azimut ou vitesses angulaires et durées correspondantes des phases du mouvement (voir ci-dessous).

## Exécution

*vbapgmmove* permet l'utilisation de sources virtuelles en mouvement. Si *ifldnum* est positif, les champs représentent les directions de la source virtuelle durant des intervalles de temps égaux, *iazi1*, [*iele1*,] *iazi2*, [*iele2*,], etc. La position de la source virtuelle est interpolée entre ces directions en partant de la première direction et en terminant à la dernière. Chaque intervalle est interpolé durant une fraction de la durée de l'évènement sonore égale à *durée\_totale* / *nombre\_intervalles*.

Si *ifldnum* est négatif, les champs représentent les vitesses angulaires à intervalles réguliers. Le premier champ est cependant la direction de départ, *iazi1*, [*iele1*,] *iazi\_vel1*, [*iele\_vel1*,] *iazi\_vel2*, [*iele\_vel2*,] ... Chaque vitesse est appliquée à la note qui occupe la fraction *durée\_totale* / *nombre\_de\_vitesses* de la durée de l'évènement sonore. Si l'élévation de la source virtuelle dépasse 90 degrés ou devient inférieure à 0 degré, la polarité de la vitesse angulaire change. Ainsi l'élévation angulaire produit une source virtuelle qui monte et descend entre 0 et 90 degrés.



### Avertissement

Prière de noter que tous les opcodes de panoramique *vbap* nécessitent une initialisation du système *vbap* avec *vbaplsinit*.

## Exemples

Voici un exemple de l'opcode `vbapgmmove`. Il utilise le fichier `vbapgmmove.csd` [examples/vbapgmmove.csd].

### Exemple 1078. Exemple de l'opcode `vbapgmmove`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vbapgmmove.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 4 ;quad
0dbfs = 1

vbaplsinit 2, 4, 0, 90, 180, 270

instr 1

asig diskint2 "beats.wav", 1, 0, 1 ;loop beats.wav

k0,k1,k2,k3 vbapgmmove p3, 1, 2, 310, 180 ;change movement of soundsource in
outq k0*asig,k1*asig,k2*asig,k3*asig ;the rear speakers

endin
</CsInstruments>
<CsScore>

i 1 0 5

e
</CsScore>
</CsoundSynthesizer>
```

## Référence

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, juin 1997, Vol. 45/6, p. 456.

## Voir aussi

`vbap`, `vbap16`, `vbap16move`, `vbap4`, `vbap4move`, `vbap8`, `vbap8move`, `vbaplsinit`, `vbapz`, `vbapzmove`

## Crédits

Auteur : Ville Pulkki  
Sibelius Academy Computer Music Studio  
Laboratoire d'Acoustique et de Traitement du Signal Audio  
Helsinki, Université de Technologie

Helsinki, Finlande

Mai 2000

Auteur : John ffitch

Juillet 2012, septembre 2013

Nouveau dans la version 5.17.13 de Csound.

Sortie sur tableau ajoutée dans la version 6.01

# vbap16

vbap16 — Distribue un signal audio sur 16 canaux.

## Description

Distribue un signal audio sur 16 canaux.

## Syntaxe

```
ar1, ..., ar16 vbap16 asig, kazim [, kelev] [, kspread]
```

## Exécution

*asig* -- signal audio à traiter.

*kazim* -- angle d'azimut de la source virtuelle.

*kelev* (facultatif) -- angle d'élévation de la source virtuelle.

*kspread* (facultatif) -- diffusion de la source virtuelle (de 0 à 100). S'il vaut 0, on a un panoramique d'amplitude conventionnel. Plus *kspread* augmente et plus le nombre de haut-parleurs utilisés dans le panoramique augmente. S'il vaut 100, le son est appliqué à tous les haut-parleurs.

*vbap16* prend un signal en entrée, *asig*, et le distribue sur 16 sorties en fonction des contrôles *kazim* et *kelev*, et de la disposition des haut-parleurs. Si *idim* = 2, *kelev* est mis à zéro. La distribution est réalisée par Panoramique d'Amplitude sur une Base de Vecteurs (VBAP - voir référence). VBAP distribue le signal en tenant compte des données de haut-parleurs configurées avec *vbaplsinit*. Le signal est appliqué au plus à deux haut-parleurs dans les configurations 2D et à trois haut-parleurs dans les configurations 3D. Si la source sonore est distribuée en dehors de la région couverte par les haut-parleurs, les haut-parleurs les plus proches sont utilisés dans le panoramique.



### Avertissement

Prière de noter que tous les opcodes de panoramique *vbap* nécessitent une initialisation du système *vbap* avec *vbaplsinit*.

## Exemples

Voir l'entrée sur *vbap8* pour un exemple d'utilisation des opcodes *vbap*.

## Référence

Ville Pulkki : « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, juin 1997, Vol. 45/6, p. 456.

## Voir aussi

*vbap16move*, *vbap4*, *vbap4move*, *vbap8*, *vbap8move*, *vbaplsinit*, *vbapz*, *vbapzmove*



## Crédits

Auteur : Ville Pulkki  
Sibelius Academy Computer Music Studio  
Laboratoire d'Acoustique et de Traitement du Signal Audio  
Helsinki, Université de Technologie  
Helsinki, Finlande  
Mai 2000

Nouveau dans la Version 4.07 de Csound. Les paramètres d'entrée acceptent le taux-k depuis Csound 5.09.

# vbap16move

vbap16move — Distribue un signal audio sur 16 canaux avec des sources virtuelles en mouvement.

## Description

Distribue un signal audio sur 16 canaux avec des sources virtuelles en mouvement.

## Syntaxe

```
ar1, ..., ar16 vbap16move asig, idur, ispread, ifldnum, ifld1 \
[, ifld2] [...]
```

## Initialisation

*idur* -- durée pendant laquelle le mouvement a lieu.

*ispread* -- diffusion de la source virtuelle (de 0 à 100). S'il vaut 0, on a un panoramique d'amplitude conventionnel. Plus *ispread* augmente et plus le nombre de haut-parleurs utilisés dans le panoramique augmente. S'il vaut 100, le son est appliqué à tous les haut-parleurs.

*ifldnum* -- nombre de champs (sa valeur absolue doit être supérieure ou égale à 2). Si *ifldnum* est positif, le mouvement de la source virtuelle est une ligne brisée spécifiée par les directions données. Chaque transition est exécutée durant un intervalle de même durée. Si *ifldnum* est négatif, les vitesses angulaires spécifiées sont appliquées à la source virtuelle durant les intervalles de temps spécifiés correspondants (voir ci-dessous).

*ifld1*, *ifld2*, ... -- angles d'azimut ou vitesses angulaires et durées correspondantes des phases du mouvement.

## Exécution

*asig* -- signal audio à traiter.

*vbap16move* permet l'utilisation de sources virtuelles en mouvement. Si *ifldnum* est positif, les champs représentent les directions de la source virtuelle durant des intervalles de temps égaux, *iazi1*, [*iele1*,] *iazi2*, [*iele2*,], etc. La position de la source virtuelle est interpolée entre ces directions en partant de la première direction et en terminant à la dernière. Chaque intervalle est interpolé durant une fraction de la durée de l'évènement sonore égale à  $\text{durée\_totale} / \text{nombre\_intervalles}$ .

Si *ifldnum* est négatif, les champs représentent les vitesses angulaires à intervalles réguliers. Le premier champ est cependant la direction de départ, *iazi1*, [*iele1*,] *iazi\_vell1*, [*iele\_vell1*,] *iazi\_vell2*, [*iele\_vell2*,] ... Chaque vitesse est appliquée à la note qui occupe la fraction  $\text{durée\_totale} / \text{nombre\_de\_vitesses}$  de la durée de l'évènement sonore. Si l'élévation de la source virtuelle dépasse 90 degrés ou devient inférieure à 0 degré, la polarité de la vitesse angulaire change. Ainsi l'élévation angulaire produit une source virtuelle qui monte et descend entre 0 et 90 degrés.



### Avertissement

Prière de noter que tous les opcodes de panoramique *vbap* nécessitent une initialisation du système *vbap* avec *vbaplsinit*.

## Exemples

Voir l'entrée sur *vbap8move* pour un exemple d'utilisation des opcodes *vbapXmove*.

## Référence

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, juin 1997, Vol. 45/6, p. 456.

## Voir aussi

*vbap16*, *vbap4*, *vbap4move*, *vbap8*, *vbap8move*, *vbaplsinit*, *vbapz*, *vbapzmove*, *vbapzmove*

## Crédits

Auteur : Ville Pulkki  
Sibelius Academy Computer Music Studio  
Laboratoire d'Acoustique et de Traitement du Signal Audio  
Helsinki, Université de Technologie  
Helsinki, Finlande  
Mai 2000

Nouveau dans la Version 4.07 de Csound.

# vbap4

vbap4 — Distribue un signal audio sur 4 canaux.

## Description

Distribue un signal audio sur 4 canaux.

## Syntaxe

```
ar1, ar2, ar3, ar4 vbap4 asig, kazim [, kelev] [, kspread]
```

## Exécution

*asig* -- signal audio à traiter.

*kazim* -- angle d'azimut de la source virtuelle.

*kelev* (facultatif) -- angle d'élévation de la source virtuelle.

*kspread* (facultatif) -- diffusion de la source virtuelle (de 0 à 100). S'il vaut 0, on a un panoramique d'amplitude conventionnel. Plus *kspread* augmente et plus le nombre de haut-parleurs utilisés dans le panoramique augmente. S'il vaut 100, le son est appliqué à tous les haut-parleurs.

*vbap4* prend un signal en entrée, *asig*, et le distribue sur 4 sorties en fonction des contrôles *kazim* et *kelev*, et de la disposition des haut-parleurs. Si *idim* = 2, *kelev* est mis à zéro. La distribution est réalisée par Panoramique d'Amplitude sur une Base de Vecteurs (VBAP - voir référence). VBAP distribue le signal en tenant compte des données de haut-parleurs configurées avec *vbaplsinit*. Le signal est appliqué au plus à deux haut-parleurs dans les configurations 2D et à trois haut-parleurs dans les configurations 3D. Si la source sonore est distribuée en dehors de la région couverte par les haut-parleurs, les haut-parleurs les plus proches sont utilisés dans le panoramique.



### Avertissement

Prière de noter que tous les opcodes de panoramique *vbap* nécessitent une initialisation du système *vbap* avec *vbaplsinit*.

## Exemples

Voir l'entrée sur *vbap8* pour un exemple d'utilisation des opcodes *vbap*.

## Référence

Ville Pulkki : « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, juin 1997, Vol. 45/6, p. 456.

## Exemples

Voici un exemple de l'opcode *vbap4*. Il utilise le fichier *vbap4.csd* [examples/vbap4.csd].

## Exemple 1079. Exemple de l'opcode vbap4.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vbap4.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 4 ;quad
0dbfs = 1

vbaplsinit 2, 4, 0, 90, 180, 270

instr 1

asig diskint2 "beats.wav", 1, 0, 1 ;loop beats.wav
kaz line 0, p3, p4 ;come from right rear speaker &
a1,a2,a3,a4 vbap4 asig, 180, 100, kaz ;change spread of soundsource
printks "spread of source = %d\n", 1, kaz ;print spread value
outq a1,a2,a3,a4

endin
</CsInstruments>
<CsScore>

i 1 0 12 100

e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
spread of source = 0
spread of source = 8
spread of source = 17
spread of source = 25
spread of source = 33
spread of source = 42
spread of source = 50
spread of source = 58
spread of source = 67
spread of source = 75
spread of source = 83
spread of source = 92
spread of source = 100
```

## Voir aussi

*vbap16, vbap16move, vbap4move, vbap8, vbap8move, vbaplsinit, vbapz, vbapzmove*

## Crédits

Auteur : Ville Pulkki

Sibelius Academy Computer Music Studio  
Laboratoire d'Acoustique et de Traitement du Signal Audio  
Helsinki, Université de Technologie  
Helsinki, Finlande  
Mai 2000

Nouveau dans la version 4.06 de Csound. Les paramètres d'entrée acceptent le taux-k depuis Csound 5.09.

# vbap4move

vbap4move — Distribue un signal audio sur 4 canaux avec des sources virtuelles en mouvement.

## Description

Distribue un signal audio sur 4 canaux avec des sources virtuelles en mouvement.

## Syntaxe

```
ar1, ar2, ar3, ar4 vbap4move asig, idur, ispread, ifldnum, ifld1 \
[, ifld2] [...]
```

## Initialisation

*idur* -- durée pendant laquelle le mouvement a lieu.

*ispread* -- diffusion de la source virtuelle (de 0 à 100). S'il vaut 0, on a un panoramique d'amplitude conventionnel. Plus *ispread* augmente et plus le nombre de haut-parleurs utilisés dans le panoramique augmente. S'il vaut 100, le son est appliqué à tous les haut-parleurs.

*ifldnum* -- nombre de champs (sa valeur absolue doit être supérieure ou égale à 2). Si *ifldnum* est positif, le mouvement de la source virtuelle est une ligne brisée spécifiée par les directions données. Chaque transition est exécutée durant un intervalle de même durée. Si *ifldnum* est négatif, les vitesses angulaires spécifiées sont appliquées à la source virtuelle durant les intervalles de temps spécifiés correspondants (voir ci-dessous).

*ifld1, ifld2, ...* -- angles d'azimut ou vitesses angulaires et durées correspondantes des phases du mouvement (voir ci-dessous).

## Exécution

*asig* -- signal audio à traiter.

*vbap4move* permet l'utilisation de sources virtuelles en mouvement. Si *ifldnum* est positif, les champs représentent les directions de la source virtuelle durant des intervalles de temps égaux, *iazi1*, [*iele1*,] *iazi2*, [*iele2*,], etc. La position de la source virtuelle est interpolée entre ces directions en partant de la première direction et en terminant à la dernière. Chaque intervalle est interpolé durant une fraction de la durée de l'évènement sonore égale à  $\text{durée\_totale} / \text{nombre\_intervalles}$ .

Si *ifldnum* est négatif, les champs représentent les vitesses angulaires à intervalles réguliers. Le premier champ est cependant la direction de départ, *iazi1*, [*iele1*,] *iazi\_vell1*, [*iele\_vell1*,] *iazi\_vell2*, [*iele\_vell2*,] ... Chaque vitesse est appliquée à la note qui occupe la fraction  $\text{durée\_totale} / \text{nombre\_de\_vitesses}$  de la durée de l'évènement sonore. Si l'élévation de la source virtuelle dépasse 90 degrés ou devient inférieure à 0 degré, la polarité de la vitesse angulaire change. Ainsi l'élévation angulaire produit une source virtuelle qui monte et descend entre 0 et 90 degrés.



### Avertissement

Prière de noter que tous les opcodes de panoramique *vbap* nécessitent une initialisation du système *vbap* avec *vbaplsinit*.

## Exemples

Voir l'entrée sur *vbap8move* pour un exemple d'utilisation des opcodes *vbapXmove*.

## Référence

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, juin 1997, Vol. 45/6, p. 456.

## Exemples

Voici un exemple de l'opcode *vbap4move*. Il utilise le fichier *vbap4move.csd* [examples/vbap4move.csd].

### Exemple 1080. Exemple de l'opcode *vbap4move*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vbap4move.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 4 ;quad
0dbfs = 1

vbaplsinit 2, 4, 0, 90, 180, 270

instr 1

asig diskin2 "beats.wav", 1, 0, 1 ;loop beats.wav
a1,a2,a3,a4 vbap4move asig, p3, 1, 2, 310, 180 ;change movement of soundsource in
      outq a1,a2,a3,a4 ;the rear speakers

endin
</CsInstruments>
<CsScore>

i 1 0 5

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*vbap16*, *vbap16move*, *vbap4*, *vbap8*, *vbap8move*, *vbaplsinit*, *vbapz*, *vbapzmove*

## Crédits

Auteur : Ville Pulkki



Sibelius Academy Computer Music Studio  
Laboratoire d'Acoustique et de Traitement du Signal Audio  
Helsinki, Université de Technologie  
Helsinki, Finlande  
Mai 2000

Nouveau dans la Version 4.07 de Csound.

# vbap8

vbap8 — Distribue un signal audio sur 8 canaux.

## Description

Distribue un signal audio sur 8 canaux.

## Syntaxe

```
ar1, ..., ar8 vbap8 asig, kazim [, kelev] [, kspread]
```

## Exécution

*asig* -- signal audio à traiter.

*kazim* -- angle d'azimut de la source virtuelle

*kelev* (facultatif) -- angle d'élévation de la source virtuelle.

*kspread* (facultatif) -- diffusion de la source virtuelle (de 0 à 100). S'il vaut 0, on a un panoramique d'amplitude conventionnel. Plus *kspread* augmente et plus le nombre de haut-parleurs utilisés dans le panoramique augmente. S'il vaut 100, le son est appliqué à tous les haut-parleurs.

*vbap8* prend un signal en entrée, *asig*, et le distribue sur 8 sorties en fonction des contrôles *kazim* et *kelev*, et de la disposition des haut-parleurs. Si *idim* = 2, *kelev* est mis à zéro. La distribution est réalisée par Panoramique d'Amplitude sur une Base de Vecteurs (VBAP - voir référence). VBAP distribue le signal en tenant compte des données de haut-parleurs configurées avec *vbaplsinit*. Le signal est appliqué au plus à deux haut-parleurs dans les configurations 2D et à trois haut-parleurs dans les configurations 3D. Si la source sonore est distribuée en dehors de la région couverte par les haut-parleurs, les haut-parleurs les plus proches sont utilisés dans le panoramique.



### Avertissement

Prière de noter que tous les opcodes de panoramique *vbap* nécessitent une initialisation du système *vbap* avec *vbaplsinit*.

## Exemples

Voici un exemple simple de l'opcode *vbap8*. Il utilise le fichier *vbap8.csd* [examples/vbap8.csd].

### Exemple 1081. Exemple de l'opcode vbap8.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
;-odac      -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
-o vbap8.wav -W ;;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

    sr      =      44100
    kr      =      441
    ksmpps  =      100
    nchnls  =      4
    vbaplsinit      2, 8,  0, 45, 90, 135, 200, 245, 290, 315

    instr 1
    asig   oscil      20000, 440, 1
    a1,a2,a3,a4,a5,a6,a7,a8   vbap8   asig, p4, 0, 20 ;p4 = azimuth

    ;render twice with alternate outq statements
    ; to obtain two 4 channel .wav files:

        outq      a1,a2,a3,a4
    ;      outq      a5,a6,a7,a8
    ; or use an 8-channel output for realtime output (set nchnls to 8):
    ;      outo a1,a2,a3,a4,a5,a6,a7,a8
    ;      endin

</CsInstruments>
<CsScore>
f 1 0 8192 10 1
; Play Instrument #1 for one second.
;      azimuth
i 1 0 1      20
i 1 + .      40
i 1 + .      60
i 1 + .      80
i 1 + .     100
i 1 + .     120
i 1 + .     140
i 1 + .     160
e

</CsScore>
</CsoundSynthesizer>

```

## Référence

Ville Pulkki : « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, juin 1997, Vol. 45/6, p. 456.

## Voir aussi

*vbap16, vbap16move, vbap4, vbap4move, vbap8move, vbaplsinit, vbapz, vbapzmove*

## Crédits

Auteur : Ville Pulkki  
 Sibelius Academy Computer Music Studio  
 Laboratoire d'Acoustique et de Traitement du Signal Audio  
 Helsinki, Université de Technologie  
 Helsinki, Finlande  
 Mai 2000

Nouveau dans la Version 4.07 de Csound. Les paramètres d'entrée acceptent le taux-k depuis Csound 5.09.

# vbap8move

vbap8move — Distribue un signal audio sur 8 canaux avec des sources virtuelles en mouvement.

## Description

Distribue un signal audio sur 8 canaux avec des sources virtuelles en mouvement.

## Syntaxe

```
ar1, ..., ar8 vbap8move asig, idur, ispread, ifldnum, ifld1 \
[, ifld2] [...]
```

## Initialisation

*idur* -- durée pendant laquelle le mouvement a lieu.

*ispread* -- diffusion de la source virtuelle (de 0 à 100). S'il vaut 0, on a un panoramique d'amplitude conventionnel. Plus *ispread* augmente et plus le nombre de haut-parleurs utilisés dans le panoramique augmente. S'il vaut 100, le son est appliqué à tous les haut-parleurs.

*ifldnum* -- nombre de champs (sa valeur absolue doit être supérieure ou égale à 2). Si *ifldnum* est positif, le mouvement de la source virtuelle est une ligne brisée spécifiée par les directions données. Chaque transition est exécutée durant un intervalle de même durée. Si *ifldnum* est négatif, les vitesses angulaires spécifiées sont appliquées à la source virtuelle durant les intervalles de temps spécifiés correspondants (voir ci-dessous).

*ifld1*, *ifld2*, ... -- angles d'azimut ou vitesses angulaires et durées correspondantes des phases du mouvement (voir ci-dessous).

## Exécution

*asig* -- signal audio à traiter.

*vbap8move* permet l'utilisation de sources virtuelles en mouvement. Si *ifldnum* est positif, les champs représentent les directions de la source virtuelle durant des intervalles de temps égaux, *iazi1*, [*iele1*,] *iazi2*, [*iele2*,], etc. La position de la source virtuelle est interpolée entre ces directions en partant de la première direction et en terminant à la dernière. Chaque intervalle est interpolé durant une fraction de la durée de l'évènement sonore égale à  $\text{durée\_totale} / \text{nombre\_intervalles}$ .

Si *ifldnum* est négatif, les champs représentent les vitesses angulaires à intervalles réguliers. Le premier champ est cependant la direction de départ, *iazi1*, [*iele1*,] *iazi\_vell1*, [*iele\_vell1*,] *iazi\_vell2*, [*iele\_vell2*,] ... Chaque vitesse est appliquée à la note qui occupe la fraction  $\text{durée\_totale} / \text{nombre\_de\_vitesses}$  de la durée de l'évènement sonore. Si l'élévation de la source virtuelle dépasse 90 degrés ou devient inférieure à 0 degré, la polarité de la vitesse angulaire change. Ainsi l'élévation angulaire produit une source virtuelle qui monte et descend entre 0 et 90 degrés.



### Avertissement

Prière de noter que tous les opcodes de panoramique *vbap* nécessitent une initialisation du système *vbap* avec *vbaplsinit*.

## Exemples

Voici un exemple simple de l'opcode *vbap8move*. Il utilise le fichier *vbap8move.csd* [examples/vbap8move.csd].

### Exemple 1082. Exemple de l'opcode *vbap8move*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vbap8move.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 48000
ksmps = 10
nchnls = 8

;Example by Hector Centeno 2007

vbaplsinit      2, 8, 15, 65, 115, 165, 195, 245, 295, 345

    instr 1
ifldnum = 9
ispread = 30
idur = p3

;; Generate a sound source
kenv loopseg 10, 0, 0, 0, 0.5, 1, 10, 0
a1 pinkish 3000*kenv

;; Move circling around once all the speakers
aout1, aout2, aout3, aout4, aout5, aout6, aout7, aout8 vbap8move a1, idur, ispread, ifldnum, 15, 65, 115, 165, 195, 245, 295, 345

;; Speaker mapping
aFL = aout8 ; Front Left
aFR = aout1 ; Front Right
aMFL = aout7 ; Mid Front Left
aMFR = aout2 ; Mid Front Right
aMBL = aout6 ; Mid Back Left
aMBR = aout3 ; Mid Back Right
aBL = aout5 ; Back Left
aBR = aout4 ; Back Right

outo aFL, aFR, aMFL, aMFR, aMBL, aMBR, aBL, aBR

    endin

</CsInstruments>
<CsScore>
i1 0 30
e
</CsScore>
</CsoundSynthesizer>
```

## Référence

Ville Pulkki : « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, juin 1997, Vol. 45/6, p. 456.

## Voir aussi

*vbap16, vbap16move, vbap4, vbap4move, vbap8, vbaplsinit, vbapz, vbapzmove*

## Crédits

Auteur : Ville Pulkki  
Sibelius Academy Computer Music Studio  
Laboratoire d'Acoustique et de Traitement du Signal Audio  
Helsinki, Université de Technologie  
Helsinki, Finlande  
Mai 2000

Nouveau dans la Version 4.07 de Csound.

# vbaplsinit

vbaplsinit — Configure la sortie VBAP selon les paramètres de haut-parleur.

## Description

Configure la sortie VBAP selon les paramètres de haut-parleur.

## Syntaxe

```
vbaplsinit idim, ilsnum [, idir1] [, idir2] [...] [, idir32]
```

```
vbaplsinit idim, ilsnum, ilsarray
```

## Initialisation

*idim* -- dimension de l'espace de haut-parleurs. 2 ou 3. Si la dimension a une partie fractionnaire, celle-ci représente l'indice de la disposition créée (utilisé seulement dans *vbap*, *vbapz* et *vbapg*). La partie fractionnaire doit être comprise entre .00 et .99.

*ilsnum* -- nombre de haut-parleurs. En deux dimensions, il peut varier entre 2 et 64. En trois dimensions, il peut varier entre 3 et 64.

*idir1*, *idir2*, ..., *idir32* -- directions des haut-parleurs. Le nombre de directions doit être inférieur ou égal à 16. Dans une répartition des haut-parleurs en deux dimensions, *idirn* représente l'angle d'azimut du *n*ième canal. Dans une répartition des haut-parleurs en trois dimensions, les champs représentent les angles d'azimut et d'élévation de chaque haut-parleur (*azi1*, *ele1*, *azi2*, *ele2*, etc.).

*ilsarray* -- tableau unidimensionnel de données comme celles décrites ci-dessus.



### Note

A deux dimensions, l'angle entre deux haut-parleurs adjacents doit être inférieur à 179 degrés (170 degrés dans les versions précédentes). C'est une restriction de l'algorithme.

## Exécution

VBAP distribue le signal en tenant compte des données de haut-parleurs configurées avec *vbaplsinit*. Le signal est appliqué au plus à deux haut-parleurs dans les configurations 2D et à trois haut-parleurs dans les configurations 3D. Si la source sonore est distribuée en dehors de la région couverte par les haut-parleurs, les haut-parleurs les plus proches sont utilisés dans le panoramique.

## Exemples

Voici un exemple de l'opcode *vbaplsinit*. Il utilise le fichier *vbaplsinit.csd* [exemples/vbaplsinit.csd].

### Exemple 1083. Exemple de l'opcode *vbaplsinit*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vwaplsinit.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 8
0dbfs = 1

vwaplsinit 2, 8, 0, 45, 90, 180, 270, 0, 0, 0 ;5 speakers for 5.1 amps

instr 1

asig disk2 "beats.wav", 1, 0, 1 ;loop beats.wav
kazim line 1, p3, 355
a1,a2,a3,a4,a5,a6,a7,a8 vbap8 asig, kazim, 0, 1 ;change azimuth of soundsource
; Speaker mapping
aFL = a1 ; Front Left
aMF = a5 ; Mid Front
aFR = a2 ; Front Right
aBL = a3 ; Back Left
aBR = a4 ; Back Right
outo aFL,aFR,aBL,aBR,aMF,a6,a7,a8 ;a6, a7 and a8 are dummies

endin
</CsInstruments>
<CsScore>

i 1 0 5

e
</CsScore>
</CsoundSynthesizer>
```

Voir les différentes entrées des opcodes vbap pour d'autres exemples.

## Référence

Ville Pulkki : « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, juin 1997, Vol. 45/6, p. 456.

## Voir aussi

*vbap, vbapg, vbap4, vbap4move, vbap8, vbap8move, vbap16, vbap16move, vbapz, vbapzmove*

## Crédits

Auteur : Ville Pulkki  
Sibelius Academy Computer Music Studio  
Laboratoire d'Acoustique et de Traitement du Signal Audio  
Helsinki, Université de Technologie  
Helsinki, Finlande  
Mai 2000

Nouveau dans la Version 4.07 de Csound.



Les dispositions multiples sont nouvelles dans la version 5.17.14

# vbapz

vbapz — Ecrit un signal audio multi-canaux dans un tableau ZAK.

## Description

Ecrit un signal audio multi-canaux dans un tableau ZAK.

## Syntaxe

```
vbapz inumchnls, istartndx, asig, kazim [, kelev] [, kspread]
```

## Initialisation

*inumchnls* -- nombre de canaux à écrire dans le tableau ZA. Doit être compris entre 2 et 256.

*istartndx* -- premier indice ou position à utiliser dans le tableau ZA.

## Exécution

*asig* -- signal audio à traiter.

*kazim* -- angle d'azimut de la source virtuelle.

*kelev* angle d'azimut de la source virtuelle.

*kspread* (facultatif) -- diffusion de la source virtuelle (de 0 à 100). S'il vaut 0, on a un panoramique d'amplitude conventionnel. Plus *kspread* augmente et plus le nombre de haut-parleurs utilisés dans le panoramique augmente. S'il vaut 100, le son est appliqué à tous les haut-parleurs.

L'opcode *vbapz* est l'équivalent multi-canaux d'opcodes comme *vbap4*, travaillant sur *inumchnls* et utilisant un tableau ZAK en sortie.



### Avertissement

Prière de noter que tous les opcodes de panoramique *vbap* nécessitent une initialisation du système *vbap* avec *vbaplsinit*.

## Exemples

Voir l'entrée sur *vbap8* pour un exemple d'utilisation des opcodes *vbap*.

## Référence

Ville Pulkki : « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, juin 1997, Vol. 45/6, p. 456.

## Voir aussi

*vbap16*, *vbap16move*, *vbap4*, *vbap4move*, *vbap8*, *vbap8move*, *vbaplsinit*, *vbapzmove*

## Crédits

John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la Version 4.07 de Csound. Les paramètres d'entrée acceptent le taux-k depuis Csound 5.09.

# vbapzmove

**vbapzmove** — Ecrit un signal audio multi-canaux dans un tableau ZAK avec des sources virtuelles en mouvement.

## Description

Ecrit un signal audio multi-canaux dans un tableau ZAK avec des sources virtuelles en mouvement.

## Syntaxe

```
vbapzmove inumchnls, istartndx, asig, idur, ispread, ifldnum, ifld1, \
        ifld2, [...]
```

## Initialisation

*inumchnls* -- nombre de canaux à écrire dans le tableau ZA. Doit être compris entre 2 et 256.

*istartndx* -- premier indice ou position à utiliser dans le tableau ZA.

*idur* -- durée pendant laquelle le mouvement a lieu.

*ispread* -- diffusion de la source virtuelle (de 0 à 100). S'il vaut 0, on a un panoramique d'amplitude conventionnel. Plus *ispread* augmente et plus le nombre de haut-parleurs utilisés dans le panoramique augmente. S'il vaut 100, le son est appliqué à tous les haut-parleurs.

*ifldnum* -- nombre de champs (sa valeur absolue doit être supérieure ou égale à 2). Si *ifldnum* est positif, le mouvement de la source virtuelle est une ligne brisée spécifiée par les directions données. Chaque transition est exécutée durant un intervalle de même durée. Si *ifldnum* est négatif, les vitesses angulaires spécifiées sont appliquées à la source virtuelle durant les intervalles de temps spécifiés correspondants (voir ci-dessous).

*ifld1*, *ifld2*, ... -- angles d'azimut ou vitesses angulaires et durées correspondantes des phases du mouvement (voir ci-dessous).

## Exécution

*asig* -- signal audio à traiter.

L'opcode *vbapzmove* est l'équivalent multi-canaux d'opcodes comme *vbap4move*, travaillant sur *inumchnls* et utilisant un tableau ZAK en sortie.



### Avertissement

Prière de noter que tous les opcodes de panoramique *vbap* nécessitent une initialisation du système *vbap* avec *vbaplsinit*.

## Exemples

Voir l'entrée sur *vbap8move* pour un exemple d'utilisation des opcodes *vbapXmove*.

## Référence

Ville Pulkki: « Virtual Sound Source Positioning Using Vector Base Amplitude Panning » *Journal of the Audio Engineering Society*, juin 1997, Vol. 45/6, p. 456.

## Voir aussi

*vbap16, vbap16move, vbap4, vbap4move, vbap8, vbap8move, vbaplsinit, vbapz,*

## Crédits

John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Mai 2000

Nouveau dans la Version 4.07 de Csound.

# vcella

vcella — Automate Cellulaire

## Description

Automate Cellulaire unidimensionnel appliqué à des vecteurs de Csound.

## Syntaxe

```
vcella ktrig, kreinit, ioutFunc, initStateFunc, \  
        iRuleFunc, ielements, irulelen [, iradius]
```

## Initialisation

*ioutFunc* - numéro de la table dans laquelle l'état de chaque cellule est stocké

*initStateFunc* - numéro de la table contenant l'état initial de chaque cellule

*iRuleFunc* - numéro de la table de consultation contenant les règles

*ielements* - nombre total de cellules

*irulelen* - nombre total de règles

*iradius* (facultatif) - rayon de l'Automate Cellulaire. Actuellement, le rayon de l'AC peut valoir 1 ou 2 (la valeur par défaut est 1)

## Exécution

*ktrig* - signal de déclenchement. Chaque fois qu'il est non nul, une nouvelle génération de cellules est évaluée.

*kreinit* - signal de déclenchement. Chaque fois qu'il est non nul, l'état de toutes les cellules est forcé à celui de *initStateFunc*.

*vcella* met en œuvre un automate cellulaire pour lequel l'état de chaque cellule est stocké dans *ioutFunc*. Ainsi *ioutFunc* est un vecteur contenant l'état courant de chaque cellule. Ce vecteur variable peut être utilisé avec d'autres opcodes basés sur des vecteurs, tels que *adsynt*, *vmap*, *vpowv* etc.

*initStateFunc* est un vecteur d'entrée contenant la valeur initiale de la rangée de cellules, tandis que *iRuleFunc* est un vecteur d'entrée contenant les règles sous la forme d'une table de consultation. Notez que *initStateFunc* et *iRuleFunc* peuvent être modifiés pendant l'exécution au moyen d'autres opcodes basés sur des vecteurs (par exemple *vcopy*) afin de forcer un changement de règle et d'état pendant l'exécution.

Une nouvelle génération de cellules est évaluée chaque fois que *ktrig* contient une valeur non nulle. De plus, l'état de toutes les cellules peut être forcé à l'état correspondant dans *initStateFunc* chaque fois que *kreinit* contient une valeur non nulle.

Le rayon de l'algorithme d'AC peut valoir 1 ou 2 (argument facultatif *iradius*).

## Exemples

Voici un exemple de l'opcode *vcella*. Il utilise le fichier *vcella.csd* [examples/vcella.csd].

L'exemple suivant utilise l'opcode *vcella*

### Exemple 1084. Exemple de l'opcode *vcella*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>

<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vcella.wav -W ;; for file output any platform
</CsOptions>

<CsInstruments>
; vcella.csd
; by Anthony Kozar

; This file demonstrates some of the new opcodes available in
; Csound 5 that come from Gabriel Maldonado's CsoundAV.

sr          = 44100
kr          = 4410
ksmps      = 10
nchnls     = 1

; Cellular automata-driven oscillator bank using vcella and adsynt
instr 1
  idur      = p3
  iCArate   = p4                                ; number of times per second the CA calculates new values

  ; f-tables for CA parameters
  iCAinit   = p5                                ; CA initial states
  iCARule   = p6                                ; CA rule values
  ; The rule is used as follows:
  ; the states (values) of each cell are summed with their neighboring cells within
  ; the specied radius (+/- 1 or 2 cells). Each sum is used as an index to read a
  ; value from the rule table which becomes the new state value for its cell.
  ; All new states are calculated first, then the new values are all applied
  ; simultaneously.

  ielements = ftlen(iCAinit)
  inumrules  = ftlen(iCARule)
  iradius    = 1

  ; create some needed tables
  iCAstate   ftgen    0, 0, ielements, -2, 0      ; will hold the current CA states
  ifreqs     ftgen    0, 0, ielements, -2, 0      ; will hold the oscillator frequency for each cell
  iamps      ftgen    0, 0, ielements, -2, 0      ; will hold the amplitude for each cell

  ; calculate cellular automata state
  ktrig      metro    iCArate                      ; trigger the CA to update iCArate times per second
              vcella   ktrig, 0, iCAstate, iCAinit, iCARule, ielements, inumrules, iradius

  ; scale CA state for use as amplitudes of the oscillator bank
              vcopy    iamps, iCAstate, ielements
              vmult     iamps, (1/3), ielements    ; divide by 3 since state values are 0-3

              vport     iamps, .01, ielements      ; need to smooth the amplitude changes for adsynt
; we could use adsynt2 instead of adsynt, but it does not seem to be working

; i-time loop for calculating frequencies
```

```

index      =      0
inew       =      1
iratio     =      1.125                      ; just major second (creating a whole tone scale)
loop1:
    tableiw inew, index, ifreqs, 0          ; 0 indicates integer indices
    inew    =      inew * iratio
    index   =      index + 1
    if (index < ielements) igoto loop1

; create sound with additive oscillator bank
ifreqbase = 64
iwavefn   = 1
iphs      = 2                              ; random oscillator phases

kenv      linseg 0.0, 0.5, 1.0, idur - 1.0, 1.0, 0.5, 0.0
aosc      adsynt kenv, ifreqbase, iwavefn, ifreqs, iamps, ielements, iphs

    out    aosc * ampdb(68)

endin

</CsInstruments>

<CsScore>
f1 0 16384 10 1

; This example uses a 4-state cellular automata
; Possible state values are 0, 1, 2, and 3

; CA initial state
; We have 16 cells in our CA, so the initial state table is size 16
f10 0 16 -2 0 1 0 0 1 0 0 2 2 0 0 1 0 0 1 0

; CA rule
; The maximum sum with radius 1 (3 cells) is 9, so we need 10 values in the rule (0-9)
f11 0 16 -2 1 0 3 2 1 0 0 2 1 0

; Here is our one and only note!
i1 0 20 4 10 11

e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Ecrit par : Gabriel Maldonado.

Nouveau dans Csound 5 (Disponible auparavant seulement dans CsoundAV)

Exemple par : Anthony Kozar



## VCO

*vco* — Implémentation de la modélisation d'un oscillateur analogique à bande de fréquence limitée.

## Description

Implémentation de la modélisation d'un oscillateur analogique à bande de fréquence limitée, basée sur l'intégration d'impulsions à bande de fréquence limitée. *vco* peut être utilisé pour simuler différentes formes d'onde analogiques.

## Syntaxe

```
ares vco xamp, xcps, iwave, kpw [, ifn] [, imaxd] [, ileak] [, inyx] \  
    [, iphs] [, iskip]
```

## Initialisation

*iwave* -- détermine la forme d'onde :

- *iwave* = 1 - dent de scie
- *iwave* = 2 - carrée/PWM
- *iwave* = 3 - triangle/dent de scie/rampe

*ifn* (facultatif, par défaut 1) -- numéro de table d'une fonction sinus stockée. Doit pointer sur une table valide qui contient une onde sinus. Csound rapportera une erreur si ce paramètre n'est pas fixé et que la table n°1 n'existe pas.

*imaxd* (facultatif, par défaut 1) -- temps de retard maximum. Une durée de  $1/4f_c$  peut être nécessaire pour les formes d'onde PWM et triangle. Le temps d'ajustement de la hauteur à cette valeur peut aller jusqu'à  $1/(\text{fréquence minimale})$ .

*ileak* (facultatif, par défaut 0) -- si *ileak* se situe entre zéro et un ( $0 < \text{ileak} < 1$ ), *ileak* est utilisé comme facteur de fuite de l'intégrateur. Sinon un facteur de fuite de 0,999 est utilisé pour les ondes en dent de scie et carrée et de 0,995 pour l'onde triangle. On peut l'utiliser pour « aplatiser » l'onde carrée ou « renforcer » l'onde en dent de scie dans les fréquences basses en fixant *ileak* à 0,99999 ou à une valeur semblable. Le résultat devrait être une onde carrée sonnant plus faux.

*inyx* (facultatif, par défaut 0,5) -- est utilisé pour déterminer le nombre d'harmoniques dans l'impulsion à bande de fréquence limitée. Tous les harmoniques jusqu'à  $sr * \text{inyx}$  seront utilisés. La valeur par défaut donne  $sr * 0,5$  ( $sr/2$ ). Pour  $sr/4$  utiliser *inyx* = 0,25. Cela peut générer un son plus « gras » dans certains cas.

*iphs* (facultatif, par défaut 0) -- c'est une valeur de phase. Il y a un artefact (comme un bogue) dans *vco* qui se produit pendant la première demi-période de l'onde carrée et qui rend la forme d'onde plus grande en amplitude que les autres. La valeur de *iphs* a un effet sur cet artefact. En particulier, si l'on fixe *iphs* à 0,5 la première demi-période de l'onde carrée ressemblera à une petite onde triangulaire. Ceci peut être préférable à la grande forme d'onde de l'artefact qui est le comportement par défaut.

*iskip* (facultatif, par défaut 0) -- s'il est non nul, l'initialisation du filtre est ignorée. (Nouveau dans les versions 4.23f13 et 5.0 de Csound)

## Exécution

*kpw* -- détermine soit la largeur de la pulsation (si *iwave* vaut 2) soit le caractère de la dent de scie / rampe (si *iwave* vaut 3). La valeur de *kpw* doit être supérieure à 0 et inférieure à 1. Une valeur de 0,5 générera une onde carrée (si *iwave* vaut 2) ou une onde triangle (si *iwave* vaut 3).

*xamp* -- détermine l'amplitude

*xcps* -- fréquence de l'onde en cycles par seconde.

## Exemples

Voici un exemple de l'opcode *vco*. Il utilise le fichier *vco.csd* [exemples/vco.csd].

### Exemple 1085. Exemple de l'opcode *vco*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vco.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1

; Instrument #1
instr 1
; Set the amplitude.
kamp = p4

; Set the frequency.
kcps = cpspch(p5)

; Select the wave form.
iwave = p6

; Set the pulse-width/saw-ramp character.
kpw init 0.5

; Use Table #1.
ifn = 1

; Generate the waveform.
asig vco kamp, kcps, iwave, kpw, ifn

; Output and amplification.
out asig
endin

</CsInstruments>
```

```
<CsScore>

; Table #1, a sine wave.
f 1 0 65536 10 1

; Define the score.
; p4 = raw amplitude (0-32767)
; p5 = frequency, in pitch-class notation.
; p6 = the waveform (1=Saw, 2=Square/PWM, 3=Tri/Saw-Ramp-Mod)
i 1 00 02 20000 05.00 1
i 1 02 02 20000 05.00 2
i 1 04 02 20000 05.00 3

i 1 06 02 20000 07.00 1
i 1 08 02 20000 07.00 2
i 1 10 02 20000 07.00 3

i 1 12 02 20000 09.00 1
i 1 14 02 20000 09.00 2
i 1 16 02 20000 09.00 3

i 1 18 02 20000 11.00 1
i 1 20 02 20000 11.00 2
i 1 22 02 20000 11.00 3
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*vco2*

## Crédits

Auteur : Hans Mikelson  
Décembre 1998

Nouveau dans la version 3.50 de Csound

Novembre 2002. Correction de la documentation pour le paramètre *kpw*. Merci à Luis Jure et à Hans Mikelson.

## vco2

vco2 — Implémentation d'un oscillateur à bande de fréquence limitée qui utilise des tables pré-calculées.

## Description

*vco2* est semblable à *vco*. Mais l'implémentation utilise des tables pré-calculées de formes d'onde à bande de fréquence limitée (voir aussi *GEN30*) plutôt que d'intégrer des impulsions. Cet opcode peut être plus rapide que *vco* (particulièrement lors de l'utilisation d'un faible taux de contrôle) et il permet également une meilleure qualité sonore. De plus, il y a plus de formes d'onde et la phase de l'oscillateur peut être modulée au taux-k. Il a pour inconvénient une utilisation plus importante de la mémoire. Pour plus de détails sur les tables de *vco2*, voir aussi *vco2init* et *vco2ft*.

## Syntaxe

```
ares vco2 kamp, kcps [, imode] [, kpw] [, kphs] [, inyx]
```

## Initialisation

*imode* (facultatif, par défaut 0) -- somme des valeurs représentant la forme d'onde et ses valeurs de contrôle.

On peut utiliser ces valeurs pour *imode* :

- 16 : active le contrôle de la phase au taux-k (s'il est positionné, *kphs* est un paramètre de taux-k nécessaire pour permettre la modulation de la phase)
- 1 : ignorer l'initialisation

On peut utiliser exactement une seule de ces valeurs de *imode* pour choisir la forme d'onde à générer :

- 14 : forme d'onde -1 définie par l'utilisateur (nécessite l'utilisation de l'opcode *vco2init*)
- 12 : triangle (pas de rampe, plus rapide)
- 10 : onde carrée (pas de PWM, plus rapide)
- 8 :  $4 * x * (1 - x)$  (c'est-à-dire l'intégration d'une dent de scie)
- 6 : pulsation (non normalisée)
- 4 : dent de scie / triangle / rampe
- 2 : carrée / PWM
- 0 : dent de scie

La valeur par défaut de *imode* est zéro, ce qui signifie une onde en dent de scie sans contrôle de la phase au taux-k.

*inyx* (facultatif, par défaut 0,5) -- largeur de bande de l'onde générée exprimée en pourcentage (0 à 1) du taux d'échantillonnage. L'intervalle attendu va de 0 à 0,5 (c'est-à-dire jusqu'à *sr/2*), les autres valeurs étant limitées à cet intervalle.

En fixant *inyx* à 0,25 (*sr*/4), ou à 0,3333 (*sr*/3), on peut produire un son plus « gras » dans certains cas, bien que la qualité sera probablement réduite.

## Exécution

*ares* -- le signal audio en sortie.

*kamp* -- amplitude. Si *imode* vaut 6 (pulsation), le niveau de sortie réel peut être bien plus élevé que cette valeur.

*kcps* -- fréquence en Hz (doit être dans l'intervalle  $-sr/2$  à  $sr/2$ ).

*kpw* (facultatif) -- largeur de pulsation de l'onde carrée (*imode* = 2) ou caractéristiques de l'onde triangle ou rampe (*imode* = 4). Il n'est requis que pour ces formes d'onde et il est ignoré dans les autres cas. L'intervalle attendu va de 0 à 1, toutes les autres valeurs y étant ramenées cycliquement.



### Avertissement

*kpw* ne doit pas être une valeur entière exacte (0 ou 1) lors de la génération d'une onde en dent de scie / triangle / rampe (*imode* = 4). Dans ce cas, l'intervalle recommandé est d'environ 0,01 à 0,99. Cette limitation n'existe pas pour une forme d'onde carrée/PWM.

*kphs* (facultatif) -- phase de l'oscillateur (en fonction de *imode*, ce sera un paramètre facultatif de taux-*i* qui vaut zéro par défaut ou un paramètre obligatoire de taux-*k*). Comme pour *kpw*, l'intervalle attendu va de 0 à 1.



### Note

Si l'on utilise un faible taux de contrôle, la largeur de pulsation (*kpw*) et la modulation de phase (*kphs*) sont converties en interne en modulation de fréquence. Cela permet un traitement plus rapide et réduit le nombre d'artefacts. Mais dans le cas de notes très longues avec des changements rapides et continus de *kpw* ou de *kphs*, la phase peut se décaler par rapport à la valeur voulue. Dans la plupart des cas, l'erreur de phase sera au maximum de 0,037 par heure (en supposant un taux d'échantillonnage de 44100 Hz).

Ceci pose problème principalement avec la largeur d'impulsion (*kpw*) par la possible apparition de divers artefacts. En attendant la résolution de ces problèmes dans de futures versions de *vco2*, les recommandations suivantes peuvent être utiles :

- N'utiliser que des valeurs de *kpw* dans l'intervalle 0,05 à 0,95. (Il y a plus d'artefacts au voisinage des valeurs entières)
- Essayer d'éviter de moduler *kpw* par des formes d'onde asymétriques telles que l'onde en dent de scie. Il est très peu probable qu'une modulation symétrique relativement lente ( $\leq 20$  Hz) (par exemple une onde sinus ou triangle), que des fonctions splines aléatoires (également lentes) ou qu'une pulsation de largeur fixe causent des problèmes de synchronisation.
- Dans certains cas, l'ajout d'un tremblement aléatoire (par exemple, des fonctions spline avec une amplitude d'environ 0,01) à *kpw* peut aussi résoudre le problème.

## Exemples

Voici un exemple de l'opcode *vco2*. Il utilise le fichier *vco2.csd* [examples/*vco2.csd*].

**Exemple 1086. Exemple de l'opcode vco2.**

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o vco2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 10
nchnls  = 1

; user defined waveform -1: trapezoid wave with default parameters (can be
; accessed at ftables starting from 10000)
itmp    ftgen 1, 0, 16384, 7, 0, 2048, 1, 4096, 1, 4096, -1, 4096, -1, 2048, 0
ift     vco2init -1, 10000, 0, 0, 0, 1
; user defined waveform -2: fixed table size (4096), number of partials
; multiplier is 1.02 (~238 tables)
itmp    ftgen 2, 0, 16384, 7, 1, 4095, 1, 1, -1, 4095, -1, 1, 0, 8192, 0
ift     vco2init -2, ift, 1.02, 4096, 4096, 2

instr 1
kcps    expon p4, p3, p5                ; instr 1: basic vco2 example
a1      vco2 12000, kcps                 ; (sawtooth wave with default
out a1                                     ; parameters)
endin

instr 2
kcps    expon p4, p3, p5                ; instr 2:
kpw     linseg 0.1, p3/2, 0.9, p3/2, 0.1 ; PWM example
a1      vco2 10000, kcps, 2, kpw
out a1
endin

instr 3
kcps    expon p4, p3, p5                ; instr 3: vco2 with user
a1      vco2 14000, kcps, 14             ; defined waveform (-1)
aenv    linseg 1, p3 - 0.1, 1, 0.1, 0   ; de-click envelope
out a1 * aenv
endin

instr 4
kcps    expon p4, p3, p5                ; instr 4: vco2ft example,
kfn     vco2ft kcps, -2, 0.25            ; with user defined waveform
a1      oscilikt 12000, kcps, kfn        ; (-2), and sr/4 bandwidth
out a1
endin

</CsInstruments>
<CsScore>

i 1 0 3 20 2000
i 2 4 2 200 400
i 3 7 3 400 20
i 4 11 2 100 200

f 0 14

```

e

```
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*vco*, *vco2ft*, *vco2ift* et *vco2init*.

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.22

## vco2ft

*vco2ft* — Retourne un numéro de table au taux-k pour une fréquence d'oscillateur donnée et une forme d'onde.

## Description

*vco2ft* retourne le numéro d'une table de fonction pour générer la forme d'onde spécifiée à une fréquence donnée. Ce numéro de table de fonction peut être utilisé par n'importe quel opcode de Csound qui génère un signal en lisant une table de fonction (comme *oscilikt*). Les tables doivent avoir été calculées par *vco2init* avant l'appel de *vco2ft* et partagées comme ftables de Csound (*ibasfn*).

## Syntaxe

```
kfn vco2ft kcps, iwave [, inyx]
```

## Initialisation

*iwave* -- la forme d'onde dont le numéro doit être choisi. Les valeurs permises sont :

- 0 : dent de scie
- 1 :  $4 * x * (1 - x)$  (intégration d'une dent de scie)
- 2 : pulsation (non normalisée)
- 3 : onde carrée
- 4 : triangle

De plus, les valeurs négatives de *iwave* sélectionnent des formes d'onde définies par l'utilisateur (voir aussi *vco2init*).

*inyx* (facultatif, par défaut 0,5) -- largeur de bande de la forme d'onde générée, exprimée en pourcentage (0 à 1) du taux d'échantillonnage. L'intervalle attendu va de 0 à 0,5 (c'est-à-dire jusqu'à  $sr/2$ ), les autres valeurs étant limitées à cet intervalle.

En fixant *inyx* à 0,25 ( $sr/4$ ), ou à 0,3333 ( $sr/3$ ), on peut produire un son plus « gras » dans certains cas, bien que la qualité sera probablement réduite.

## Exécution

*kfn* -- le numéro de la ftable, retourné au taux-k.

*kcps* -- fréquence en Hz, retournée au taux-k. On peut utiliser zéro ou des valeurs négatives. Cependant, si la valeur absolue dépasse  $sr/2$  (ou  $sr * inyx$ ), la table sélectionnée ne contiendra que du silence.

## Exemples

Voir l'exemple de l'opcode *vco2*.



## Voir aussi

*vco2ift*, *vco2init* et *vco2*.

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.22

# vco2ift

*vco2ift* — Retourne un numéro de table au temps-*i* pour une fréquence d'oscillateur donnée et une forme d'onde.

## Description

*vco2ift* est le même que *vco2ft*, mais il travaille au temps-*i*. Il est prévu pour être utilisé avec les opcodes qui attendent un numéro de table au taux-*i* (par exemple, *oscili*).

## Syntaxe

```
ifn vco2ift icps, iwave [, inyx]
```

## Initialisation

*ifn* -- le numéro de ftable.

*icps* -- fréquence en Hz. On peut utiliser zéro ou des valeurs négatives. Cependant, si la valeur absolue dépasse  $sr/2$  (ou  $sr * inyx$ ), la table sélectionnée ne contiendra que du silence.

*iwave* -- la forme d'onde dont le numéro doit être choisi. Les valeurs permises sont :

- 0 : dent de scie
- 1 :  $4 * x * (1 - x)$  (intégration d'une dent de scie)
- 2 : pulsation (non normalisée)
- 3 : onde carrée
- 4 : triangle

De plus, les valeurs négatives de *iwave* sélectionnent des formes d'onde définies par l'utilisateur (voir aussi *vco2init*).

*inyx* (facultatif, par défaut 0,5) -- largeur de bande de la forme d'onde générée, exprimée en pourcentage (0 à 1) du taux d'échantillonnage. L'intervalle attendu va de 0 à 0,5 (c'est-à-dire jusqu'à  $sr/2$ ), les autres valeurs étant limitées à cet intervalle.

En fixant *inyx* à 0,25 ( $sr/4$ ), ou à 0,3333 ( $sr/3$ ), on peut produire un son plus « gras » dans certains cas, bien que la qualité sera probablement réduite.

## Exemples

Voici un exemple de l'opcode *vco2ift*. Il utilise le fichier *vco2ift.csd* [examples/vco2ift.csd].

### Exemple 1087. Exemple de l'opcode *vco2ift*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vco2ift.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

; user defined waveform -2: fixed table size (64), number of partials
; multiplier is 1.4
itmp   ftgen 2, 0, 64, 5, 1, 2, 120, 60, 1, 1, 0.001, 1
ift    vco2init -2, 3, 1.4, 4096, 4096, 2

instr 1

icps = p4
ifn   vco2ift icps, -2, 0.5 ;with user defined waveform
print ifn
asig  oscili 1, 220, ifn ; (-2), and sr/2 bandwidth
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 2 20
i 1 3 2 2000
i 1 6 2 20000

e
</CsScore>
</CsoundSynthesizer>

```

La sortie contiendra des lignes comme celles-ci :

```

instr 1:  ifn = 22.000
instr 1:  ifn = 8.000
instr 1:  ifn = 3.000

```

Voir aussi l'exemple de l'opcode *vco2*.

## Voir aussi

*vco2ft*, *vco2init* et *vco2*.

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.22

# vco2init

vco2init — Calcul des tables à utiliser par l'opcode *vco2*.

## Description

*vco2init* calcule des tables à utiliser par l'opcode *vco2*. En option, on peut accéder aussi à ces tables comme si elles étaient des tables de fonction standard de Csound. Dans ce cas, on peut utiliser *vco2ft* pour trouver le numéro de table correct pour une fréquence d'oscillateur donnée.

Dans la plupart des cas, cet opcode est appelé depuis l'en-tête de l'orchestre. L'utilisation de *vco2init* dans des instruments est possible mais non recommandée. En effet, le remplacement de tables durant l'exécution peut causer un plantage de Csound si d'autres opcodes sont en train d'accéder à ces tables au même moment.

Notez que *vco2init* n'est pas nécessaire au fonctionnement de *vco2* (les tables sont automatiquement allouées au premier appel de *vco2*, si ce n'est pas déjà fait), cependant il peut être utile dans certains cas :

- Pré-calcul des tables pendant le chargement de l'orchestre. C'est utile lorsque l'on ne veut pas générer les tables pendant l'exécution, afin de ne pas risquer une interruption du traitement en temps réel.
- Partage des tables comme ftables Csound. Par défaut, ces tables ne sont accessibles que par *vco2*.
- Modification des paramètres par défaut des tables (par exemple leur taille) ou utilisation d'une forme d'onde définie par l'utilisateur spécifiée dans une table de fonction.

## Syntaxe

```
ifn vco2init iwave [, ibasfn] [, ipmul] [, iminsiz] [, imaxsiz] [, isrcft]
```

## Initialisation

*ifn* -- le premier numéro de table libre après les tables allouées. Si *ibasfn* n'a pas été spécifié, -1 est retourné.

*iwave* -- somme des valeurs suivantes sélectionnant quelles tables d'onde il faut calculer :

- 16 : triangle
- 8 : onde carrée
- 4 : pulsation (non normalisée)
- 2 :  $4 * x * (1 - x)$  (intégration d'une dent de scie)
- 1 : dent de scie

Alternativement, *iwave* peut être fixé à un entier négatif qui sélectionne une forme d'onde définie par l'utilisateur. Pour cela, le paramètre *isrcft* doit être aussi spécifié. *vco2* peut accéder à la forme d'onde numéro -1. Cependant, les autres formes d'onde définies par l'utilisateur ne sont utilisables qu'avec *vco2ft* ou *vco2ift*.

*ibasfn* (facultatif, par défaut -1) -- numéro de ftable à partir duquel les opcodes autres que *vco2* peuvent accéder à l'ensemble de tables. Il est nécessaire pour les formes d'onde définies par l'utilisateur, à l'exception de -1. Si cette valeur est inférieure à 1, il n'est pas possible d'accéder aux tables calculées par *vco2init* en tant que tables de fonction de Csound.

*ipmul* (facultatif, par défaut 1,05) -- coefficient multiplicatif pour le nombre d'harmoniques. Si une table a  $n$  harmoniques, la suivante en aura  $n * ipmul$  (au moins  $n + 1$ ). L'intervalle autorisé pour *ipmul* va de 1,01 à 2. Zéro et les valeurs négatives sélectionnent la valeur par défaut (1,05).

*iminsiz* (facultatif, par défaut -1) -- taille de table minimale.

*imaxsiz* (facultatif, par défaut -1) -- taille de table maximale.

La taille de table réelle est calculée en multipliant la racine carrée du nombre d'harmoniques par *iminsiz*, puis en arrondissant le résultat à la puissance de deux supérieure, tout en l'obligeant à ne pas dépasser *imaxsiz*.

Les deux paramètres, *iminsiz* et *imaxsiz*, doivent être des puissances de deux, dans l'intervalle autorisé. L'intervalle autorisé va de 16 à 262144 pour *iminsiz* jusqu'à 16777216 pour *imaxsiz*. Zéro ou des valeurs négatives sélectionnent les réglages par défaut :

- La taille minimale est 128 pour toutes les formes d'onde sauf pour la pulsation (*iwave* = 4). Sa taille minimale est de 256.
- La taille maximale par défaut vaut normalement la taille minimale multipliée par 64, mais pas plus de 16384 si possible. Elle vaut toujours au moins la taille minimale.

*isrcft* (facultatif, par défaut -1) -- numéro de la ftable source pour les formes d'onde définies par l'utilisateur (si *iwave* < 0). *isrcft* doit pointer sur une table de fonction contenant la forme d'onde à utiliser pour générer le tableau de tables. Il est recommandé d'avoir une taille de table d'au moins *imaxsiz* points. Si *iwave* n'est pas négatif (les tables d'onde internes sont utilisées), *isrcft* est ignoré.



## Avertissement

Le nombre et la taille des tables ne sont pas fixes. Les orchestres ne doivent pas dépendre de ces paramètres, car ils peuvent changer d'une version à l'autre de Csound.

Si la table sélectionnée existe déjà, elle est remplacée. Si un opcode est en train d'accéder aux tables au même moment, il est fort probable qu'un plantage se produise. C'est pourquoi il est recommandé de n'utiliser *vco2init* que dans l'en-tête de l'orchestre.

Il ne faut pas remplacer/écraser ces tables par les routines GEN ou l'opcode *fgen*. Sinon, un comportement imprévisible voire un plantage de Csound peuvent se produire si *vco2* est utilisé. Le premier numéro de ftable libre après le tableau de tables est retourné dans *ifn*.

## Exemples

Voici un exemple de l'opcode *vco2init*. Il utilise le fichier *vco2init.csd* [examples/vco2init.csd].

### Exemple 1088. Exemple de l'opcode *vco2init*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vco2init.wav -W   ;; for file output any platform
```

```
</CsOptions>
<CsInstruments>

sr=44100
ksmps=1
nchnls=2

; create waveform with discontinuities, so it has a lot of high freq content
gitable ftgen 0, 0, 2^16+1, 7, -1, 2^14, 1, 0, -1, 2^14, 1, 0, -1, 2^15, 1
; make bandlimited tables of the waveform
gi_nextfree vco2init -gitable, gitable+1, 1.05, 128, 2^16, gitable
gitable_b1 = -gitable

instr 1

kfreq expon 14000, p3, 500
kfn vco2ft kfreq, gitable_b1
asig oscilikt 5000, kfreq, kfn
printk 0.1, kfn

; remove semicolon on next line to hear original waveform, demonstrating the aliasing
;asig oscili 5000, kfreq, gitable
outs asig, asig

endin
</CsInstruments>
<CsScore>
i1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

La sortie contiendra des lignes comme celles-ci :

```
i 1 time 0.00002: 103.00000
i 1 time 0.10000: 103.00000
i 1 time 0.20000: 103.00000
i 1 time 0.30002: 103.00000
i 1 time 0.40000: 104.00000
i 1 time 0.50000: 104.00000
.....
.....
i 1 time 4.80002: 135.00000
i 1 time 4.90000: 136.00000
i 1 time 5.00000: 138.00000
```

Voir aussi l'exemple de l'opcode *vco2*.

## Voir aussi

*vco2ft*, *vco2ift* et *vco2*.

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.22

# vcomb

vcomb — Réverbération variable du signal d'entrée avec une réponse en fréquence « colorée ».

## Description

Réverbération variable du signal d'entrée avec une réponse en fréquence « colorée ».

## Syntaxe

```
ares vcomb asig, krvt, xlpt, imaxlpt [, iskip] [, insmps]
```

## Initialisation

*imaxlpt* -- durée de boucle maximale pour *klpt*

*iskip* (facultatif, 0 par défaut) -- état initial de l'espace de données de la boucle de retard (cf. *reson*). La valeur par défaut est 0.

*insmps* (facultatif, 0 par défaut) -- valeur du retard, en nombre d'échantillons.

## Exécution

*krvt* -- la durée de réverbération (définie comme le temps en secondes pris par un signal pour décroître à 1/1000 ou 60 dB de son amplitude originale).

*xlpt* -- durée de boucle variable en secondes, comme *ilpt* dans *comb*. La durée de boucle peut aller jusqu'à *imaxlpt*.

Ce filtre répète l'entrée avec une densité d'écho déterminée par la durée de boucle *xlpt*. Le taux d'atténuation est indépendant et il est déterminé par *krvt*, la durée de réverbération (définie comme le temps en secondes pris par un signal pour décroître à 1/1000 ou 60 dB de son amplitude originale). La sortie n'apparaît qu'après *ilpt* secondes.

## Exemples

Voici un exemple de l'opdoce vcomb. Il utilise le fichier *vcomb.csd* [examples/vcomb.csd].

### Exemple 1089. Exemple de l'opdoce vcomb.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc          -M0 ;;;RT audio I/O with MIDI in
</CsOptions>
<CsInstruments>

; Example by Jonathan Murphy and Charles Gran 2007
sr      = 44100
ksmps   = 10
nchnls  = 2
```

```

; new, and important. Make sure that midi note events are only
; received by instruments that actually need them.

; turn default midi routing off
massign 0, 0
; route note events on channel 1 to instr 1
massign 1, 1

; Define your midi controllers
#define C1 #21#
#define C2 #22#
#define C3 #23#

; Initialize MIDI controllers
initc7 1, $C1, 0.5 ;delay send
initc7 1, $C2, 0.5 ;delay: time to zero
initc7 1, $C3, 0.5 ;delay: rate

gaosc    init    0

; Define an opcode to "smooth" the MIDI controller signal
opcode smooth, k, k
kin      xin
kport    linseg    0, 0.0001, 0.01, 1, 0.01
kin      portk      kin, kport
xout     kin
endop

instr 1
; Generate a sine wave at the frequency of the MIDI note that triggered the instrument
ifqc cpsmidi
iamp    ampmidi    10000
aenv    linenr     iamp, .01, .1, .01 ;envelope
a1      oscil      aenv, ifqc, 1
; All sound goes to the global variable gaosc
gaosc   = gaosc + a1
endin

instr 198 ; ECHO
kcbsnd  ctrl7      1, $C1, 0, 1 ;delay send
ktime   ctrl7      1, $C2, 0.01, 6 ;time loop fades out
kloop   ctrl7      1, $C3, 0.01, 1 ;loop speed
; Receive MIDI controller values and then smooth them
kcbsnd   smooth    kcbsnd
ktime    smooth    ktime
kloop    smooth    kloop
imaxlpt  = 1 ;max loop time
; Create a variable reverberation (delay) of the gaosc signal
acomb    vcomb     gaosc, ktime, kloop, imaxlpt, 1
aout     = (acomb * kcbsnd) + gaosc * (1 - kcbsnd)
outs     aout, aout
gaosc    = 0
endin

</CsInstruments>

<CsScore>
f1 0 16384 10 1
i198 0 10000
e
</CsScore>
</CsoundSynthesizer>

```



## Voir aussi

*alpass, comb, reverb, valpass*

## Crédits

Auteur : William « Pete » Moss  
Université du Texas à Austin  
Austin, Texas USA  
Janvier 2002

# vcopy

vcopy — Copie entre deux signaux vectoriels de contrôle.

## Description

Copie entre deux signaux vectoriels de contrôle.

## Syntaxe

```
vcopy ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [, kverbose]
```

## Initialisation

*ifn1* - numéro de la table dans laquelle le signal vectoriel sera copié (destination).

*ifn2* - numéro de la table hébergeant le signal vectoriel à copier (source).

## Exécution

*kelements* - nombre de composantes du vecteur.

*kdstoffset* - décalage d'indexation pour la table de destination *ifn1* (vaut 0 par défaut).

*ksrcoffset* - décalage d'indexation pour la table source *ifn2* (vaut 0 par défaut).

*kverbose* - Indique si les avertissements sont affichés (vaut 0 par défaut).

*vcopy* copie *kelements* éléments de *ifn2* (à partir de la position *ksrcoffset*) vers *ifn1* (à partir de la position *kdstoffset*). Utile pour conserver les valeurs de l'ancien vecteur en les stockant dans une autre table.

Des valeurs négatives pour *kdstoffset* et *ksrcoffset* sont acceptables. Si *kdstoffset* est négatif, la partie du vecteur hors-limites est ignorée. Si *ksrcoffset* est négatif, les éléments hors-limites seront supposés valoir 1 (c'est-à-dire que les éléments de destination ne seront pas changés). Si des éléments pour le vecteur de destination sont au-delà de la taille de la table (point de garde inclus), ces éléments sont ignorés (les éléments ne sont pas repliés autour des tables). Si des éléments pour le vecteur source sont au-delà de la longueur de la table, ces éléments sont supposés valoir 1 (le vecteur de destination ne sera pas changé pour ces éléments).

Si l'argument facultatif *kverbose* est différent de 0, l'opcode affichera des messages d'avertissement à chaque passe-k si les longueurs de table sont dépassées.



### Avertissement

L'utilisation de la même table comme source et comme destination dans les versions antérieures à la 5.04 peut induire un comportement imprévu. A utiliser avec précaution.

Cet opcode travaille au taux-k (cela signifie qu'à chaque passe-k les vecteurs sont copiés). Il y a une version de taux-i de cet opcode appelée *vcopy\_i*.



## Note

Prière de noter que l'argument *elements* a changé dans la version 5.03 du taux-i au taux-k. Cela change le comportement de l'opcode dans le cas inhabituel où la variable de taux-i *ielements* est modifiée à l'intérieur de l'instrument, par exemple dans :

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

Tous ces opérateurs (*vaddv*, *vsubv*, *vmultv*, *vddivv*, *vpowv*, *vexp*, *vcopy* et *vmap*) sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2* etc.

## Exemples

Voici un exemple de l'opcode *vcopy*. Il utilise le fichier *vcopy.csd* [examples/vcopy.csd].

### Exemple 1090. Exemple de l'opcode *vcopy*.

```
<CsSoundSynthesizer>
<CsOptions>; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc    -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vcopy.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
kr=4410
ksmps=10
nchnls=2

instr 1 ;table playback
ar lposcil 1, 1, 0, 262144, 1
outs ar,ar
endin

instr 2
vcopy 2, 1, 20000 ;copy vector from sample to empty table
vmult 5, 20000, 262144 ;scale noise to make it audible
vcopy 1, 5, 20000 ;put noise into sample
turnoff
endin

instr 3
vcopy 1, 2, 20000 ;put original information back in
turnoff
endin

</CsInstruments>
<CsScore>
f1 0 262144 -1 "beats.wav" 0 4 0
f2 0 262144 2 0

f5 0 262144 21 3 30000
```

```
i1 0 4
i2 3 1

s
i1 0 4
i3 3 1
s

i1 0 4

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# **vcopy\_i**

`vcopy_i` — Copie un vecteur d'une table dans une autre.

## **Description**

Copie un vecteur d'une table dans une autre.

## **Syntaxe**

```
vcopy_i ifn1, ifn2, ielements [,idstoffset, isrcoffset]
```

## **Initialisation**

*ifn1* - numéro de la table dans laquelle le signal vectoriel sera copié.

*ifn2* - numéro de la table hébergeant le signal vectoriel à copier.

*ielements* - nombre de composantes du vecteur.

*idstoffset* - décalage d'indexation pour la table de destination.

*isrcoffset* - décalage d'indexation pour la table source.

## **Exécution**

`vcopy_i` copie *ielements* éléments de *ifn2* (à partir de la position *isrcoffset*) vers *ifn1* (à partir de la position *idstoffset*). Utile pour conserver les valeurs de l'ancien vecteur en les stockant dans une autre table. Cet opcode est exactement le même que `vcopy` sauf qu'il exécute toute la copie pendant le passe d'initialisation.

Des valeurs négatives pour *idstoffset* et *isrcoffset* sont acceptables. Si *idstoffset* est négatif, la partie du vecteur hors-limites est ignorée. Si *isrcoffset* est négatif, les éléments hors-limites seront supposés valoir 0 (c'est-à-dire que les éléments de destination ne seront pas changés). Si des éléments pour le vecteur de destination sont au-delà de la taille de la table (point de garde inclus), ces éléments sont ignorés (les éléments ne sont pas repliés autour des tables). Si des éléments pour le vecteur source sont au-delà de la longueur de la table, ces éléments sont supposés valoir 0 (le vecteur de destination ne sera pas changé pour ces éléments).



### **Avertissement**

L'utilisation de la même table comme source et comme destination dans les versions antérieures à la 5.04 peut induire un comportement imprévu. A utiliser avec précaution.

Tous ces opérateurs (`vaddv`, `vsubv`, `vmultv`, `vdivv`, `vpowv`, `vexp`, `vcopy` et `vmap`) sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que `vcella`, `adsynt`, `adsynt2` etc.

## **Exemples**

See `vcopy` pour un exemple.

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vdelay

vdelay — Un délai variable avec interpolation.

## Description

C'est un délai variable avec interpolation qui n'est pas très différent de l'implémentation existante (*deltapi*), il est simplement plus facile à utiliser.

## Syntaxe

```
ares vdelay asig, adel, imaxdel [, iskip]
```

## Initialisation

*imaxdel* -- Valeur maximale du délai en millisecondes. Si *adel* reçoit une valeur supérieure à *imaxdel* celle-ci est repliée autour de *imaxdel*. Cela est à éviter.

*iskip* -- L'initialisation est ignorée s'il est présent et différent de zéro.

## Exécution

Avec ce générateur unitaire il est possible de faire des effets Doppler ou de chorus et de flanger.

*asig* -- Signal en entrée.

*adel* -- Valeur courante du délai en millisecondes. Noter que les fonctions linéaires n'ont pas d'effet de modification de la hauteur. Des valeurs de *adel* changeant rapidement provoqueront des discontinuités dans la forme d'onde ce qui donne du bruit.

## Exemples

Voici un exemple de l'opcode *vdelay*. Il utilise le fichier *vdelay.csd* [examples/vdelay.csd].

### Exemple 1091. Exemple de l'opcode *vdelay*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vdelay.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 32
nchnls  = 2
0dbfs   = 1
```

```
instr 1

ims = 100      ;maximum delay time in msec
aout poscil .8, 220, 1 ;make a signal
a2 poscil3 ims/2, 1/p3, 1 ;make an LFO
a2 = a2 + ims/2 ;offset the LFO so that it is positive
asig vdelay aout, a2, ims ;use the LFO to control delay time
outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1 ;sine wave

i 1 0 5

e
</CsScore>
</CsoundSynthesizer>
```

Deux points importants ici. D'abord, la valeur du retard doit toujours être positive. Ensuite, même si la valeur du retard peut être contrôlée au taux-k, il n'est pas prudent d'agir ainsi, car des changements de durée soudains provoqueront des clics.

## Voir aussi

*vdelay3*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995



# vdelay3

vdelay3 — Un délai variable avec interpolation cubique.

## Description

*vdelay3* est expérimental. Il est semblable à *vdelay* sauf qu'il utilise l'interpolation cubique. (Nouveau dans la version 3.50.)

## Syntaxe

```
ares vdelay3 asig, adel, imaxdel [, iskip]
```

## Initialisation

*imaxdel* -- Valeur maximale du délai en millisecondes. Si *adel* reçoit une valeur supérieure à *imaxdel* celle-ci est repliée autour de *imaxdel*. Cela est à éviter.

*iskip* (facultatif) -- L'initialisation est ignorée s'il est présent et différent de zéro.

## Exécution

Avec ce générateur unitaire il est possible de faire des effets Doppler ou de chorus et de flanger.

*asig* -- Signal en entrée.

*adel* -- Valeur courante du délai en millisecondes. Noter que les fonctions linéaires n'ont pas d'effet de modification de la hauteur. Des valeurs de *adel* changeant rapidement provoqueront des discontinuités dans la forme d'onde ce qui donne du bruit.

## Exemples

Voici un exemple de l'opcode *vdelay3*. Il utilise le fichier *vdelay3.csd* [examples/vdelay3.csd].

### Exemple 1092. Exemple de l'opcode *vdelay3*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vdelay3.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
```

```
instr 1

ims = 100 ;maximum delay time in msec
aout poscil .8, 220, 1 ;make a signal
a2 poscil ims/2, 1/p3, 1 ;make an LFO
a2 = a2 + ims/2 ;offset the LFO so that it is positive
asig vdelay3 aout, a2, ims ;use the LFO to control delay time
outs asig, asig

endin

</CsInstruments>
<CsScore>
f1 0 8192 10 1

i 1 0 5

e

</CsScore>
</CsoundSynthesizer>
```

Deux points importants ici. D'abord, la valeur du retard doit toujours être positive. Ensuite, même si la valeur du retard peut être contrôlée au taux-k, il n'est pas prudent d'agir ainsi, car des changements de durée soudains provoqueront des clics.

## Voir aussi

*vdelay*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

# vdelayx

vdelayx — Un opcode de délai variable avec interpolation de grande qualité.

## Description

Un opcode de délai variable avec interpolation de grande qualité.

## Syntaxe

```
aout vdelayx ain, adl, imd, iws [, ist]
```

## Initialisation

*imd* -- durée maximale du délai (en secondes).

*iws* -- taille de la fenêtre d'interpolation (voir ci-dessous).

*ist* (facultatif) -- l'initialisation est ignorée s'il est différent de zéro.

## Exécution

*aout* -- signal audio en sortie.

*ain* -- signal audio en entrée.

*adl* -- durée du délai en secondes.

Cet opcode utilise une interpolation de grande qualité (et peu rapide), qui est bien plus précise que les interpolations linéaire et cubique couramment disponibles. Le paramètre *iws* fixe le nombre d'échantillons en entrée utilisés pour le calcul d'un échantillon en sortie (les valeurs permises sont des multiples entiers de 4 compris entre 4 et 1024) ; plus les valeurs sont élevées, meilleure est la qualité et plus lent le processus.



### Notes

- La durée du délai est mesurée en secondes (à la différence de *vdelay* et de *vdelay3*), et doit être de taux-a.
- Le délai minimum autorisé est de *iws*/2 échantillons.
- Il est permis d'utiliser les mêmes variables en entrée et en sortie dans ces opcodes.
- Dans *vdelayxw*\*, le changement de la durée du délai à des effets sur le volume de sortie :  
$$a = 1 / (1 + dt)$$
  
où *a* est le gain en sortie et *dt* est la valeur du changement du délai par seconde.
- Ces opcodes sont plus adaptés à la version de Csound en double précision.

## Exemples

Voici un exemple de l'opcode *vdelayx*. Il utilise le fichier *vdelayx.csd* [examples/vdelayx.csd].

### Exemple 1093. Exemple de l'opcode **vdelayx**.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime output leave only the line below:
; -o vdelayx.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ims = .5      ;maximum delay time in seconds
iws = 1024    ;window size
adl = .5      ;delay time
asig diskin2 "fox.wav", 1, 0, 1 ;loop fox.wav
a2 poscil .2, .2, 1 ;make an LFO
adl = a2 + ims/2 ;offset the LFO so that it is positive
aout vdelayx asig, adl, ims, iws ;use the LFO to control delay time
outs aout, aout

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1

i 1 0 10

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*vdelayxq, vdelayxs, vdelayxw, vdelayxwq, vdelayxws*

# vdelayxq

vdelayxq — Un opcode de délai variable sur 4 canaux avec interpolation de grande qualité.

## Description

Un opcode de délai variable sur 4 canaux avec interpolation de grande qualité.

## Syntaxe

```
aout1, aout2, aout3, aout4 vdelayxq ain1, ain2, ain3, ain4, adl, imd, iws [, ist]
```

## Initialisation

*imd* -- durée maximale du délai (en secondes).

*iws* -- taille de la fenêtre d'interpolation (voir ci-dessous).

*ist* (facultatif) -- l'initialisation est ignorée s'il est différent de zéro.

## Exécution

*aout1, aout2, aout3, aout4* -- signaux audio en sortie.

*ain1, ain2, ain3, ain4* -- signaux audio en entrée.

*adl* -- durée du délai en secondes.

Cet opcode utilise une interpolation de grande qualité (et peu rapide), qui est bien plus précise que les interpolations linéaire et cubique couramment disponibles. Le paramètre *iws* fixe le nombre d'échantillons en entrée utilisés pour le calcul d'un échantillon en sortie (les valeurs permises sont des multiples entiers de 4 compris entre 4 et 1024) ; plus les valeurs sont élevées, meilleure est la qualité et plus lent le processus.

Les opcodes multicanaux (par exemple *vdelayxq*) permettent de retarder 2 ou 4 variables à la fois (signaux stéréo ou quadro) ; c'est bien plus efficace que d'utiliser un opcode séparé pour chaque canal.



### Notes

- La durée du délai est mesurée en secondes (à la différence de *vdelay* et de *vdelay3*), et doit être de taux-a.
- Le délai minimum autorisé est de  $iws/2$  échantillons.
- Il est permis d'utiliser les mêmes variables en entrée et en sortie dans ces opcodes.
- Dans *vdelayxw\**, le changement de la durée du délai a des effets sur le volume de sortie :  
$$a = 1 / (1 + dt)$$
  
où *a* est le gain en sortie et *dt* est la valeur du changement du délai par seconde.
- Ces opcodes sont plus adaptés à la version de Csound en double précision.

## Exemples

Voici un exemple de l'opcode `vdelayxq`. Il utilise le fichier `vdelayxq.csd` [examples/vdelayxq.csd].

### Exemple 1094. Exemple de l'opcode `vdelayxq`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vdelayxq.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr      = 44100
ksmps   = 32
nchnls  = 4
0dbfs   = 1

instr 1

ims      = .5      ;maximum delay time in seconds
iws      = 1024     ;window size
adl      = .5

aout1    diskin2 "beats.wav", 1, 0, 1   ;loop beats.wav
aout2    diskin2 "fox.wav", 1, 0, 1    ;loop fox.wav
aout3    diskin2 "Church.wav", 1, 0, 1  ;loop Church.wav
aout4    diskin2 "flute.aiff", 1, 0, 1   ;loop flute.aiff
a2       poscil .1, .5, 1              ;make an LFO, 1 cycle per 2 seconds
adl      = a2 + ims/2                  ;offset the LFO so that it is positive
aout1, aout2, aout3, aout4 vdelayxq aout1, aout2, aout3, aout4, adl, ims, iws; Use the LFO to control d
outq aout1, aout2, aout3, aout4

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1

i 1 0 10

e
</CsScore>
</CsoundSynthesizer>
```

Deux points importants ici. D'abord, la valeur du retard doit toujours être positive. Ensuite, même si la valeur du retard peut être contrôlée au taux-k, il n'est pas prudent d'agir ainsi, car des changements de durée soudains provoqueront des clics.

## Voir aussi

`vdelayx`, `vdelayxs`, `vdelayxw`, `vdelayxwq`, `vdelayxws`

# vdelayxs

vdelayxs — Un opcode de délai variable stéréo avec interpolation de grande qualité.

## Description

Un opcode de délai variable stéréo avec interpolation de grande qualité.

## Syntaxe

```
aout1, aout2 vdelayxs ain1, ain2, adl, imd, iws [, ist]
```

## Initialisation

*imd* -- durée maximale du délai (en secondes).

*iws* -- taille de la fenêtre d'interpolation (voir ci-dessous).

*ist* (facultatif) -- l'initialisation est ignorée s'il est différent de zéro.

## Exécution

*aout1*, *aout2* -- signaux audio en sortie.

*ain1*, *ain2* -- signaux audio en entrée.

*adl* -- durée du délai en secondes.

Cet opcode utilise une interpolation de grande qualité (et peu rapide), qui est bien plus précise que les interpolations linéaire et cubique couramment disponibles. Le paramètre *iws* fixe le nombre d'échantillons en entrée utilisés pour le calcul d'un échantillon en sortie (les valeurs permises sont des multiples entiers de 4 compris entre 4 et 1024) ; plus les valeurs sont élevées, meilleure est la qualité et plus lent le processus.

Les opcodes multicanaux (par exemple *vdelayxq*) permettent de retarder 2 ou 4 variables à la fois (signaux stéréo ou quadro) ; c'est bien plus efficace que d'utiliser un opcode séparé pour chaque canal.



### Notes

- La durée du délai est mesurée en secondes (à la différence de *vdelay* et de *vdelay3*), et doit être de taux-a.
- Le délai minimum autorisé est de *iws*/2 échantillons.
- Il est permis d'utiliser les mêmes variables en entrée et en sortie dans ces opcodes.
- Dans *vdelayxw\**, le changement de la durée du délai a des effets sur le volume de sortie :  
$$a = 1 / (1 + dt)$$
  
où *a* est le gain en sortie et *dt* est la valeur du changement du délai par seconde.
- Ces opcodes sont plus adaptés à la version de Csound en double précision.

## Exemples

Voici un exemple de l'opcode `vdelayxs`. Il utilise le fichier `vdelayxs.csd` [examples/vdelayxs.csd].

### Exemple 1095. Exemple de l'opcode `vdelayxs`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vdelayxs.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ims = .5      ;maximum delay time in seconds
iws = 1024    ;window size
adl = .5      ;delay time
asig1, asig2 diskin2 "kickroll.wav", 1, 0, 1 ;loop stereo file kickroll.wav
a2 poscil .25, .1, 1 ;make an LFO, 1 cycle per 2 seconds
adl = a2 + ims/2 ;offset the LFO so that it is positive
aoutL, aoutR vdelayxs asig1, asig2, adl, ims, iws ;use the LFO to control delay time
outs aoutL, aoutR

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1

i 1 0 10

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

`vdelayx`, `vdelayxq`, `vdelayxw`, `vdelayxwq`, `vdelayxws`



# vdelayxw

vdelayxw — Opcode de délai variable avec interpolation de grande qualité.

## Description

Opcode de délai variable avec interpolation de grande qualité.

## Syntaxe

```
aout vdelayxw ain, adl, imd, iws [, ist]
```

## Initialisation

*imd* -- durée maximale du délai (en secondes).

*iws* -- taille de la fenêtre d'interpolation (voir ci-dessous).

*ist* (facultatif) -- l'initialisation est ignorée s'il est différent de zéro.

## Exécution

*aout* -- signal audio en sortie.

*ain* -- signal audio en entrée.

*adl* -- durée du délai en secondes.

Cet opcode utilise une interpolation de grande qualité (et peu rapide), qui est bien plus précise que les interpolations linéaire et cubique couramment disponibles. Le paramètre *iws* fixe le nombre d'échantillons en entrée utilisés pour le calcul d'un échantillon en sortie (les valeurs permises sont des multiples entiers de 4 compris entre 4 et 1024) ; plus les valeurs sont élevées, meilleure est la qualité et plus lent le processus.

Les opcodes *vdelayxw*\* changent la position d'écriture dans la ligne à retard (au contraire de tous les autres générateurs unitaires de délai qui déplacent la position de lecture), et sont particulièrement utiles pour implémenter l'effet Doppler dans lequel la position de l'auditeur est fixe alors que la source est en mouvement.



### Notes

- La durée du délai est mesurée en secondes (à la différence de *vdelay* et de *vdelay3*), et doit être de taux-a.
- Le délai minimum autorisé est de *iws*/2 échantillons.
- Il est permis d'utiliser les mêmes variables en entrée et en sortie dans ces opcodes.
- Dans *vdelayxw*\*, le changement de la durée du délai à des effets sur le volume de sortie :

$$a = 1 / (1 + dt)$$

où *a* est le gain en sortie et *dt* est la valeur du changement du délai par seconde.

- Ces opcodes sont plus adaptés à la version de Csound en double précision.

## Exemples

Voici un exemple de l'opcode `vdelayxw`. Il utilise le fichier `vdelayxw.csd` [examples/vdelayxw.csd].

### Exemple 1096. Exemple de l'opcode `vdelayxw`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vdelayxw.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ims = .5      ;maximum delay time in seconds
iws = 1024    ;best quality
adl = .5      ;delay time
asig diskln2 "flute.aiff", .5, 0, 1 ;loop flute.aiff at half speed
a2  poscil3 .2, .1, 1      ;make an LFO, 1 cycle per 2 seconds
adl = a2 + ims/2           ;offset the LFO so that it is positive
aout vdelayxw asig, adl, ims, iws ;use the LFO to control delay time
outs aout, aout

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1

i 1 0 10

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

`vdelayx`, `vdelayxq`, `vdelayxs`, `vdelayxwq`, `vdelayxws`

# vdelayxwq

vdelayxwq — Opcode de délai variable avec interpolation de grande qualité.

## Description

Opcode de délai variable avec interpolation de grande qualité.

## Syntaxe

```
aout1, aout2, aout3, aout4 vdelayxwq ain1, ain2, ain3, ain4, adl, \  
imd, iws [, ist]
```

## Initialisation

*imd* -- durée maximale du délai (en secondes).

*iws* -- taille de la fenêtre d'interpolation (voir ci-dessous).

*ist* (facultatif) -- l'initialisation est ignorée s'il est différent de zéro.

## Exécution

*ain1, ain2, ain3, ain4* -- signaux audio en entrée.

*aout1, aout2, aout3, aout4* -- signaux audio en sortie.

*adl* -- durée du délai en secondes.

Cet opcode utilise une interpolation de grande qualité (et peu rapide), qui est bien plus précise que les interpolations linéaire et cubique couramment disponibles. Le paramètre *iws* fixe le nombre d'échantillons en entrée utilisés pour le calcul d'un échantillon en sortie (les valeurs permises sont des multiples entiers de 4 compris entre 4 et 1024) ; plus les valeurs sont élevées, meilleure est la qualité et plus lent le processus.

Les opcodes *vdelayxw\** changent la position d'écriture dans la ligne à retard (au contraire de tous les autres générateurs unitaires de délai qui déplacent la position de lecture), et sont particulièrement utiles pour implémenter l'effet Doppler dans lequel la position de l'auditeur est fixe alors que la source est en mouvement.

Les opcodes multicanaux (par exemple *vdelayxq*) permettent de retarder 2 ou 4 variables à la fois (signaux stéréo ou quadro) ; c'est bien plus efficace que d'utiliser un opcode séparé pour chaque canal.



### Notes

- La durée du délai est mesurée en secondes (à la différence de *vdelay* et de *vdelay3*), et doit être de taux-a.
- Le délai minimum autorisé est de *iws*/2 échantillons.
- Il est permis d'utiliser les mêmes variables en entrée et en sortie dans ces opcodes.
- Dans *vdelayxw\**, le changement de la durée du délai à des effets sur le volume de sortie :

$$a = 1 / (1 + dt)$$

où  $a$  est le gain en sortie et  $dt$  est la valeur du changement du délai par seconde.

- Ces opcodes sont plus adaptés à la version de Csound en double précision.

## Exemples

Voici un exemple de l'opcode `vdelayxwq`. Il utilise le fichier `vdelayxwq.csd` [examples/vdelayxwq.csd].

### Exemple 1097. Exemple de l'opcode `vdelayxwq`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vdelayxwq.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 4
0dbfs = 1

instr 1

ims = .5      ;maximum delay time in seconds
iws = 1024    ;best quality
adl = .5      ;delay time
aout1 diskin2 "beats.wav", 1, 0, 1 ;loop beats.wav
aout2 diskin2 "fox.wav", 1, 0, 1 ;loop fox.wav
aout3 diskin2 "Church.wav", 1, 0, 1 ;loop Church.wav
aout4 diskin2 "flute.aiff", 1, 0, 1 ;loop flute.aiff
a2 poscil3 .2, .1, 1 ;make an LFO, 1 cycle per 2 seconds
adl = a2 + ims/2 ;offset the LFO so that it is positive
aout1, aout2, aout3, aout4 vdelayxwq aout1, aout2, aout3, aout4, adl, ims, iws ;use the LFO to control
outq aout1, aout2, aout3, aout4

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1

i 1 0 10

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

`vdelayx`, `vdelayxq`, `vdelayxs`, `vdelayxw`, `vdelayxws`

# vdelayxws

vdelayxws — Opcode de délai variable avec interpolation de grande qualité.

## Description

Opcode de délai variable avec interpolation de grande qualité.

## Syntaxe

```
aout1, aout2 vdelayxws ain1, ain2, adl, imd, iws [, ist]
```

## Initialisation

*imd* -- durée maximale du délai (en secondes).

*iws* -- taille de la fenêtre d'interpolation (voir ci-dessous).

*ist* -- (facultatif) -- l'initialisation est ignorée s'il est différent de zéro.

## Exécution

*ain1, ain2* -- signaux audio en entrée.

*aout1, aout2* -- signaux audio en sortie.

*adl* -- durée du délai en secondes.

Cet opcode utilise une interpolation de grande qualité (et peu rapide), qui est bien plus précise que les interpolations linéaire et cubique couramment disponibles. Le paramètre *iws* fixe le nombre d'échantillons en entrée utilisés pour le calcul d'un échantillon en sortie (les valeurs permises sont des multiples entiers de 4 compris entre 4 et 1024) ; plus les valeurs sont élevées, meilleure est la qualité et plus lent le processus.

Les opcodes *vdelayxw\** changent la position d'écriture dans la ligne à retard (au contraire de tous les autres générateurs unitaires de délai qui déplacent la position de lecture), et sont particulièrement utiles pour implémenter l'effet Doppler dans lequel la position de l'auditeur est fixe alors que la source est en mouvement.

Les opcodes multicanaux (par exemple *vdelayxq*) permettent de retarder 2 ou 4 variables à la fois (signaux stéréo ou quadro) ; c'est bien plus efficace que d'utiliser un opcode séparé pour chaque canal.



### Notes

- La durée du délai est mesurée en secondes (à la différence de *vdelay* et de *vdelay3*), et doit être de taux-a.
- Le délai minimum autorisé est de *iws*/2 échantillons.
- Il est permis d'utiliser les mêmes variables en entrée et en sortie dans ces opcodes.
- Dans *vdelayxw\**, le changement de la durée du délai a des effets sur le volume de sortie :

$$a = 1 / (1 + dt)$$

où  $a$  est le gain en sortie et  $dt$  est la valeur du changement du délai par seconde.

- Ces opcodes sont plus adaptés à la version de Csound en double précision.

## Exemples

Voici un exemple de l'opcode `vdelayxws`. Il utilise le fichier `vdelayxws.csd` [examples/vdelayxws.csd].

### Exemple 1098. Exemple de l'opcode `vdelayxws`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vdelayxws.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ims = .5      ;maximum delay time in seconds
iws = 1024    ;window size
adl = .5      ;delay time
asig1, asig2 diskin2 "kickroll.wav", 1, 0, 1 ;loop stereo file kickroll.wav
a2 poscil .2, .1, 1 ;make an LFO, 1 cycle per 10 seconds
adl = a2 + ims/2 ;offset the LFO so that it is positive
aoutL, aoutR vdelayxws asig1, asig2, adl, ims, iws ;use the LFO to control delay time
outs aoutL, aoutR

endin
</CsInstruments>
<CsScore>
f1 0 8192 10 1

i 1 0 10

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

`vdelayx`, `vdelayxq`, `vdelayxs`, `vdelayxw`, `vdelayxwq`

# vdelayk

vdelayk — Délai variable au taux-k.

## Description

Délai variable appliqué à un signal de taux-k.

## Syntaxe

```
kout vdelayk ksig, kdel, imaxdel [, iskip, imode]
```

## Initialisation

*imaxdel* - Valeur maximale du délai en secondes.

*iskip* (facultatif) - L'initialisation est ignorée s'il est présent et différent de zéro.

*imode* (facultatif) - S'il est différent de zéro, l'interpolation linéaire est supprimée. Bien que, normalement, l'interpolation améliore la qualité du signal, il faut la supprimer si l'on utilise *vdelayk* avec des signaux de contrôle discrets comme, par exemple, des signaux de déclenchement.

## Exécution

*kout* - signal retardé en sortie.

*ksig* - signal d'entrée.

*kdel* - valeur du retard en secondes. Peut être changé au taux-k.

*vdelayk* est semblable à *vdelay*, mais il fonctionne au taux-k. Il est conçu pour retarder des signaux de contrôle. A utiliser, par exemple, dans la composition algorithmique.

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vdivv

vdivv — Division entre deux signaux vectoriels de contrôle.

## Description

Division entre deux signaux vectoriels de contrôle.

## Syntaxe

```
vdivv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
```

## Initialisation

*ifn1* - numéro de la table hébergeant le premier vecteur à traiter.

*ifn2* - numéro de la table hébergeant le second vecteur à traiter.

## Exécution

*kelements* - nombre de composantes des deux vecteurs.

*kdstoffset* - décalage d'indexation pour la table de destination *ifn1* (vaut 0 par défaut).

*ksrcoffset* - décalage d'indexation pour la table source *ifn2* (vaut 0 par défaut).

*kverbose* - Indique si les avertissements sont affichés (vaut 0 par défaut).

*vdivv* divise deux signaux vectoriels de contrôle, chaque composante de *ifn1* étant divisée par la composante correspondante de *ifn2*. Chaque signal vectoriel est hébergé dans une table (*ifn1* et *ifn2*). Le nombre de composantes de chaque vecteur doit être identique.

Le résultat est un nouveau signal vectoriel de contrôle qui écrase les anciennes valeurs de *ifn1*. Si l'on veut garder l'ancien vecteur *ifn1*, il faut utiliser l'opcode *vcopy\_i* pour le copier dans une autre table. On peut utiliser *kdstoffset* et *ksrcoffset* pour spécifier des vecteurs à n'importe quelle position dans les tables.

Des valeurs négatives pour *kdstoffset* et *ksrcoffset* sont acceptables. Si *kdstoffset* est négatif, la partie du vecteur hors-limites est ignorée. Si *ksrcoffset* est négatif, les éléments hors-limites seront supposés valoir 0 (c'est-à-dire que les éléments de destination seront mis à 0). Si des éléments pour le vecteur de destination sont au-delà de la taille de la table (point de garde inclus), ces éléments sont ignorés (les éléments ne sont pas repliés autour des tables). Si des éléments pour le vecteur source sont au-delà de la longueur de la table, ces éléments sont supposés valoir 0 (les éléments de destination seront mis à 0).

Si l'argument facultatif *kverbose* est différent de 0, l'opcode affichera des messages d'avertissement à chaque passe-k si les longueurs de table sont dépassées.



### Avertissement

L'utilisation de la même table comme source et comme destination dans les versions antérieures à la 5.04 peut induire un comportement imprévu. A utiliser avec précaution.

Cet opcode travaille au taux-k (cela signifie qu'à chaque passe-k les vecteurs sont divisés). Il y a une version de taux-i de cet opcode appelée *vdivv\_i*.





## Note

Prière de noter que l'argument *elements* a changé dans la version 5.03 du taux-i au taux-k. Cela change le comportement de l'opcode dans le cas inhabituel où la variable de taux-i *ielements* est modifiée à l'intérieur de l'instrument, par exemple dans :

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

Tous ces opérateurs (*vaddv*, *vsubv*, *vmultv*, *vdivv*, *vpowv*, *vexpv*, *vcopy* et *vmap*) sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Exemples

Voici un exemple de l'opcode *vdivv*. Il utilise le fichier *vdivv.csd* [examples/vdivv.csd].

### Exemple 1099. Exemple de l'opcode *vdivv*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ifn2 = p5
ielements = p6
idstoffset = p7
isrcoffset = p8
kval init 25
vdivv ifn1, ifn2, ielements, idstoffset, isrcoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
turnoff
endif
```

```

kcount = kcount + 1
endin

</CsInstruments>
<CsScore>

f 1 0 16 -7 1 15 16

f 2 0 16 -7 1 15 2


i2 0.0 0.2 1
i2 0.2 0.2 2
i1 0.4 0.01 1 2 5 3 8
i2 0.8 0.2 1
i1 1.0 0.01 1 2 5 10 -2
i2 1.2 0.2 1
i1 1.4 0.01 1 2 8 14 0
i2 1.6 0.2 1
i1 1.8 0.01 1 2 8 0 14
i2 2.0 0.2 1
i1 2.2 0.002 1 1 8 5 2
i2 2.4 0.2 1
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vdivv\_i

`vdivv_i` — Division entre deux signaux vectoriels de contrôle à l'initialisation.

## Description

Division entre deux signaux vectoriels de contrôle à l'initialisation.

## Syntaxe

```
vdivv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
```

## Initialisation

*ifn1* - numéro de la table hébergeant le premier vecteur à traiter.

*ifn2* - numéro de la table hébergeant le second vecteur à traiter.

*ielements* - nombre de composantes des deux vecteurs.

*idstoffset* - décalage d'indexation pour la table de destination *ifn1* (vaut 0 par défaut).

*isrcoffset* - décalage d'indexation pour la table source *ifn2* (vaut 0 par défaut).

## Exécution

`vdivv_i` divise deux signaux vectoriels de contrôle, chaque composante de *ifn1* étant divisée par la composante correspondante de *ifn2*. Chaque signal vectoriel est hébergé dans une table (*ifn1* et *ifn2*). Le nombre de composantes de chaque vecteur doit être identique.

Le résultat est un nouveau signal vectoriel de contrôle qui écrase les anciennes valeurs de *ifn1*. Si l'on veut garder l'ancien vecteur *ifn1*, il faut utiliser l'opcode `vcopy_i` pour le copier dans une autre table. On peut utiliser *idstoffset* et *isrcoffset* pour spécifier des vecteurs à n'importe quelle position dans les tables.

Des valeurs négatives pour *idstoffset* et *isrcoffset* sont acceptables. Si *idstoffset* est négatif, la partie du vecteur hors-limites est ignorée. Si *isrcoffset* est négatif, les éléments hors-limites seront supposés valoir 1 (c'est-à-dire que les éléments de destination ne seront pas changés). Si des éléments pour le vecteur de destination sont au-delà de la taille de la table (point de garde inclus), ces éléments sont ignorés (les éléments ne sont pas repliés autour des tables). Si des éléments pour le vecteur source sont au-delà de la longueur de la table, ces éléments sont supposés valoir 1 (les éléments correspondants du vecteur de destination ne seront pas changés).



### Avertissement

L'utilisation de la même table comme source et comme destination dans les versions antérieures à la 5.04 peut induire un comportement imprévu. A utiliser avec précaution.

Cet opcode travaille à l'initialisation. Il y a une version de taux-k de appelée `vdivv`.

Tous ces opérateurs (`vaddv_i`, `vsubv_i`, `vmultv_i`, `vdivv_i`, `vpowv_i`, `vexpv_i`, `vcopy` et `vmap`) sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que `vcella`, `adsynt`, `adsynt2`, etc.

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vecdelay

vecdelay — Ligne à retard vectorielle au taux-k.

## Description

Génère une sorte de retard "vectoriel".

## Syntaxe

```
vecdelay ifn, ifnIn, ifnDel, ielements, imaxdel [, iskip]
```

## Initialisation

*ifn* - numéro de la table contenant le vecteur de sortie.

*ifnIn* - numéro de la table contenant le vecteur d'entrée.

*ifnDel* - numéro de la table contenant un vecteur dont les composantes contiennent des valeurs de retard en secondes.

*ielements* - nombre de composantes des deux vecteurs.

*imaxdel* - Valeur maximale du retard en secondes.

*iskip* (facultatif) - disposition initiale de l'espace des données de la boucle de retard (voir *reson*). La valeur par défaut est 0.

## Exécution

*vecdelay* est semblable à *vdelay*, mais il fonctionne au taux-k et, au lieu de retarder un signal unique, il retarde un vecteur. *ifnIn* est le vecteur d'entrée des signaux, *ifn* est le vecteur de sortie des signaux, et *ifnDel* est un vecteur contenant les valeurs de retard pour chaque composante, exprimées en secondes. Les composantes de *ifnDel* peuvent être modifiées au taux-k. Chaque retard unique peut être différent de celui des autres composantes, et il peut varier au taux-k. *imaxdel* fixe le retard maximum autorisé pour toutes les composantes de *ifnDel*.

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# veloc

veloc — Donne la vélocité d'un évènement MIDI.

## Description

Donne la vélocité d'un évènement MIDI.

## Syntaxe

```
ival veloc [ilow] [, ihigh]
```

## Initialisation

*ilow, ihigh* -- Limites basse et haute pour le mappage

## Exécution

Donne la valeur de l'octet MIDI (0 - 127) pour la vélocité de l'évènement courant.

## Exemples

Voici un exemple de l'opcode veloc. Il utilise le fichier *veloc.csd* [examples/veloc.csd].

### Exemple 1100. Exemple le l'opcode veloc.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -+rtmidi=virtual -M0   ;;realtime audio I/O with MIDI in
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisine ftgen 0, 0, 1024, 10, 1

instr 1

ivel veloc 0, 1   ;scale 0 - 1
print ivel      ;print velocity
asig poscil .5*ivel, 220, gisine ;and use it as amplitude
      outs asig, asig

endin
</CsInstruments>
<CsScore>
```

```
f 0 30 ;runs 30 seconds
```

```
e  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*aftouch, ampmidi, cpsmidi, cpsmidib, midictrl, notnum, octmidi, octmidib, pchbend, pchmidi, pchmidib*

## Crédits

Auteur : Barry L. Vercoe - Mike Berry  
MIT - Mills  
Mai 1997

# vexp

vexp — Elévation à une puissance entre un scalaire et un vecteur.

## Description

Elévation à une puissance entre un scalaire et un vecteur.

## Syntaxe

```
vexp ifn, kval, kelements [, kdstoffset] [, kverbose]
```

## Initialisation

*ifn* - numéro de la table hébergeant le signal vectoriel à traiter.

## Exécution

*kval* - opérande scalaire à traiter.

*kelements* - nombre de composantes du vecteur.

*kdstoffset* - décalage d'indexation pour la table de destination (facultatif, vaut 0 par défaut).

*kverbose* - Indique si les avertissements sont affichés (vaut 0 par défaut).

*vexp* élève *kval* à la puissance de chaque élément contenu dans un vecteur de la table *ifn*, à partir de l'indice *kdstoffset*. Cela permet de traiter une section particulière d'une table en spécifiant le décalage et le nombre d'éléments à traiter. Le décalage est compté à partir de 0, si bien que si aucun décalage n'est spécifié (ou s'il est fixé à 0), la table est modifiée depuis le début.

Noter que cet opcode est exécuté au taux-k si bien que la valeur de *kval* est traitée à chaque période de contrôle. A utiliser avec précaution si l'on ne veut pas finir avec des nombres très grands ou très petits (ou utiliser *vexp\_i*).

Ces opcodes (*vadd*, *vmult*, *vpow* et *vexp*) réalisent des opérations numériques entre un signal vectoriel de contrôle (hébergé par la table *ifn*), et un signal scalaire (*kval*). Le résultat est un nouveau vecteur qui écrase les anciennes valeurs de *ifn*. Tous ces opcodes travaillent au taux-k.

Les valeurs négatives sont valides pour *kdstoffset*. Les composantes du vecteur se trouvant en dehors de la table sont alors ignorées, et elles ne sont pas repliées autour de la table.

Si l'argument facultatif *kverbose* est différent de 0, l'opcode affichera des messages d'avertissement à chaque passe-k si les longueurs de table sont dépassées.

Dans tous ces opcodes, les vecteurs résultants sont stockés dans *ifn*, écrasant les vecteurs initiaux. Si l'on veut garder le vecteur initial, il faut utiliser *vcopy* ou *vcopy\_i* pour le copier dans une autre table. Tous ces opérateurs sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc. Ils peuvent aussi être utiles en conjonction avec les opcodes spectraux *pvsftw* et *pvsftr*.





## Note

Prière de noter que l'argument *elements* a changé dans la version 5.03 du taux-i au taux-k. Cela change le comportement de l'opcode dans le cas inhabituel où la variable de taux-i *ielements* est modifiée à l'intérieur de l'instrument, par exemple dans :

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

## Exemples

Voici un exemple de l'opcode *vexp*. Il utilise le fichier *vexp.csd* [examples/vexp.csd].

### Exemple 1101. Exemple de l'opcode *vexp*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vexp ifn1, ival, ielements, idstoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
turnoff
endif

kcount = kcount + 1
endin

</CsInstruments>
```

```
<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
e

</CsScore>

</CsoundSynthesizer>
```

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vexp\_i

vexp\_i — Elévation à une puissance entre un scalaire et un vecteur.

## Description

Elévation à une puissance entre un scalaire et un vecteur.

## Syntaxe

```
vexp_i ifn, ival, ielements[, idstoffset]
```

## Initialisation

*ifn* - numéro de la table hébergeant le signal vectoriel à traiter.

*ival* - opérande scalaire à traiter.

*ielements* - nombre de composantes du vecteur.

*ival* - opérande scalaire à traiter.

*idstoffset* - décalage d'indexation pour la table de destination (facultatif, vaut 0 par défaut).

## Exécution

*vexp\_i* élève *ival* à la puissance de chaque élément contenu dans un vecteur de la table *ifn*, à partir de l'indice *idstoffset*. Cela permet de traiter une section particulière d'une table en spécifiant le décalage et le nombre d'éléments à traiter. Le décalage est compté à partir de 0, si bien que si aucun décalage n'est spécifié (ou s'il est fixé à 0), la table est modifiée depuis le début.

Les valeurs négatives sont valides pour *idstoffset*. Les composantes du vecteur se trouvant en dehors de la table sont alors ignorées, et elles ne sont pas repliées autour de la table.

Cet opcode ne s'exécute qu'à l'initialisation. Il y a une version de taux-k de cet opcode appelée *vexp*.

Dans tous ces opcodes, les vecteurs résultants sont stockés dans *ifn*, écrasant les vecteurs initiaux. Si l'on veut garder le vecteur initial, il faut utiliser *vcopy* ou *vcopy\_i* pour le copier dans une autre table. Tous ces opérateurs sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc. Ils peuvent aussi être utiles en conjonction avec les opcodes spectraux *pvsftw* et *pvsftr*.

## Exemples

Voici un exemple de l'opcode *vexp\_i*. Il utilise le fichier *vexp\_i.csd* [examples/vexp\_i.csd].

### Exemple 1102. Exemple de l'opcode *vexp\_i*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>
```

```

; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vexp_i ifn1, ival, ielements, idstoffset
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
  turnoff
endif

kcount = kcount + 1
endin

</CsInstruments>

<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
e

</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

*vadd*, *vmult\_i*, *vpow\_i* et *vexp\_i*.

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vexpseg

vexpseg — Générateur d'enveloppe vectorielle.

## Description

Génère des segments exponentiels vectoriels.

## Syntaxe

```
vexpseg ifnout, ielements, ifn1, idur1, ifn2 [, idur2, ifn3 [...]]
```

## Initialisation

*ifnout* - numéro de la table hébergeant le signal vectoriel de sortie.

*ifn1* - vecteur de départ.

*ifn2, ifn3, etc.* - vecteur après *idurx* secondes.

*idur1* - durée en secondes du premier segment.

*idur2, idur3, etc.* - durée en secondes des segments suivants.

*ielements* - nombre de composantes des vecteurs.

## Exécution

Ces opcodes sont semblables à *linseg* et à *expseg*, mais ils opèrent avec des signaux vectoriels au lieu de signaux scalaires.

La sortie est un signal vectoriel de contrôle hébergé par *ifnout* (qui doit avoir été allouée au préalable), tandis que chaque point charnière de l'enveloppe est un vecteur de valeurs. Tous les points charnière doivent avoir le même nombre de composantes (*ielements*).

Tous ces opérateurs sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Exemples

Voici un exemple de l'opcode vexpseg. Il utilise le fichier *vexpseg.csd* [examples/vexpseg.csd].

### Exemple 1103. Exemple de l'opcode vexpseg.

```
<CsoundSynthesizer>  
<CsOptions>  
-odac -B441 -b441  
</CsOptions>  
<CsInstruments>
```

```

sr=44100
ksmps=10
nchnls=2

gilen init 32

gitable1 ftgen 0, 0, gilen, 10, 1
gitable2 ftgen 0, 0, gilen, 10, 1

gitable3 ftgen 0, 0, gilen, -7, 30, gilen, 35
gitable4 ftgen 0, 0, gilen, -7, 400, gilen, 450
gitable5 ftgen 0, 0, gilen, -7, 5000, gilen, 5500

instr 1
vcopy gitable2, gitable1, gilen
turnoff
endin

instr 2
vexpseg gitable2, 16, gitable3, 2, gitable4, 2, gitable5
endin

instr 3
kcount init 0
if kcount < 16 then
  kval table kcount, gitable2
  printk 0,kval
  kcount = kcount +1
else
  turnoff
endif
endin

</CsInstruments>
<CsScore>
i1 0 1
s
i2 0 10
i3 0 1
i3 1 1
i3 1.5 1
i3 2 1
i3 2.5 1
i3 3 1
i3 3.5 1
i3 4 1
i3 4.5 1

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Ecrit par Gabriel Maldonado.

Exemple par Andres Cabrera.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vexpv

vexpv — Exponentiation entre deux signaux vectoriels de contrôle.

## Description

Exponentiation entre deux signaux vectoriels de contrôle.

## Syntaxe

```
vexpv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
```

## Initialisation

*ifn1* - numéro de la table hébergeant le premier vecteur à traiter.

*ifn2* - numéro de la table hébergeant le second vecteur à traiter.

## Exécution

*kelements* - nombre de composantes des deux vecteurs.

*kdstoffset* - décalage d'indexation pour la table de destination *ifn1* (vaut 0 par défaut).

*ksrcoffset* - décalage d'indexation pour la table source *ifn2* (vaut 0 par défaut).

*kverbose* - Indique si les avertissements sont affichés (vaut 0 par défaut).

*vexpv* élève chaque élément de *ifn2* à la puissance donnée par l'élément correspondant de *ifn1*. Chaque signal vectoriel est hébergé dans une table (*ifn1* et *ifn2*). Le nombre de composantes de chaque vecteur doit être identique.

Le résultat est un nouveau signal vectoriel de contrôle qui écrase les anciennes valeurs de *ifn1*. Si l'on veut garder l'ancien vecteur *ifn1*, il faut utiliser l'opcode *vcopy\_i* pour le copier dans une autre table. On peut utiliser *kdstoffset* et *ksrcoffset* pour spécifier des vecteurs à n'importe quelle position dans les tables.

Des valeurs négatives pour *kdstoffset* et *ksrcoffset* sont acceptables. Si *kdstoffset* est négatif, la partie du vecteur hors-limites est ignorée. Si *ksrcoffset* est négatif, les éléments hors-limites seront supposés valoir 0 (c'est-à-dire que les éléments de destination seront fixés à 1). Si des éléments pour le vecteur de destination sont au-delà de la taille de la table (point de garde inclus), ces éléments sont ignorés (les éléments ne sont pas repliés autour des tables). Si des éléments pour le vecteur source sont au-delà de la longueur de la table, ces éléments sont supposés valoir 0 (c'est-à-dire que les éléments de destination seront fixés à 1).

Si l'argument facultatif *kverbose* est différent de 0, l'opcode affichera des messages d'avertissement à chaque passe-k si les longueurs de table sont dépassées.



### Avertissement

L'utilisation de la même table comme source et comme destination dans les versions antérieures à la 5.04 peut induire un comportement imprévu. A utiliser avec précaution.

Cet opcode travaille au taux-k (cela signifie qu'à chaque passe-k les vecteurs sont traités). Il y a une version de taux-i de cet opcode appelée *vexpv\_i*.



## Note

Prière de noter que l'argument *elements* a changé dans la version 5.03 du taux-i au taux-k. Cela change le comportement de l'opcode dans le cas inhabituel où la variable de taux-i *ielements* est modifiée à l'intérieur de l'instrument, par exemple dans :

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

Tous ces opérateurs (*vaddv*, *vsubv*, *vmultv*, *vdivv*, *vpowv*, *vexpv*, *vcopy* et *vmap*) sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Exemples

Voici un exemple de l'opcode *vexpv*. Il utilise le fichier *vexpv.csd* [examples/vexpv.csd].

### Exemple 1104. Exemple de l'opcode *vexpv*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ifn2 = p5
ielements = p6
idstoffset = p7
isrcoffset = p8
kval init 25
vexpv ifn1, ifn2, ielements, idstoffset, isrcoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
turnoff
endif
```



```

kcount = kcount + 1
endin

</CsInstruments>

<CsScore>

f 1 0 16 -7 1 16 17

f 2 0 16 -7 0 16 1

i2 0.0 0.2 1
i2 0.2 0.2 2
i1 0.4 0.01 1 2 5 3 8
i2 0.8 0.2 1
i1 1.0 0.01 1 2 5 10 -2
i2 1.2 0.2 1
i1 1.4 0.01 1 2 8 14 0
i2 1.6 0.2 1
i1 1.8 0.002 1 2 8 0 14
i2 2.0 0.2 1
i1 2.2 0.002 1 1 8 5 2
i2 2.4 0.2 1
e

</CsScore>

</CsoundSynthesizer>

```

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vexpv\_i

vexpv\_i — Exponentiation entre deux signaux vectoriels de contrôle à l'initialisation.

## Description

Exponentiation entre deux signaux vectoriels de contrôle à l'initialisation.

## Syntaxe

```
vexpv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
```

## Initialisation

*ifn1* - numéro de la table hébergeant le premier vecteur à traiter.

*ifn2* - numéro de la table hébergeant le second vecteur à traiter.

*ielements* - nombre de composantes des deux vecteurs.

*idstoffset* - décalage d'indexation pour la table de destination *ifn1* (vaut 0 par défaut).

*isrcoffset* - décalage d'indexation pour la table source *ifn2* (vaut 0 par défaut).

## Exécution

*vexpv\_i* élève chaque élément de *ifn2* à la puissance donnée par l'élément correspondant de *ifn1*. Chaque signal vectoriel est hébergé dans une table (*ifn1* et *ifn2*). Le nombre de composantes de chaque vecteur doit être identique.

Le résultat est un nouveau signal vectoriel de contrôle qui écrase les anciennes valeurs de *ifn1*. Si l'on veut garder l'ancien vecteur *ifn1*, il faut utiliser l'opcode *vcopy\_i* pour le copier dans une autre table. On peut utiliser *idstoffset* et *isrcoffset* pour spécifier des vecteurs à n'importe quelle position dans les tables.

Des valeurs négatives pour *idstoffset* et *isrcoffset* sont acceptables. Si *idstoffset* est négatif, la partie du vecteur hors-limites est ignorée. Si *isrcoffset* est négatif, les éléments hors-limites seront supposés valoir 0 (c'est-à-dire que les éléments de destination seront fixés à 1). Si des éléments pour le vecteur de destination sont au-delà de la taille de la table (point de garde inclus), ces éléments sont ignorés (les éléments ne sont pas repliés autour des tables). Si des éléments pour le vecteur source sont au-delà de la longueur de la table, ces éléments sont supposés valoir 0 (c'est-à-dire que les éléments de destination seront fixés à 1).



### Avertissement

L'utilisation de la même table comme source et comme destination dans les versions antérieures à la 5.04 peut induire un comportement imprévu. A utiliser avec précaution.

Cet opcode travaille à l'initialisation. Il y a une version de taux-k de appelée *vexpv*.

Tous ces opérateurs (*vaddv\_i*, *vsubv\_i*, *vmultv\_i*, *vdivv\_i*, *vpowv\_i*, *vexpv\_i*, *vcopy* et *vmap*) sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vibes

vibes — Modèle physique de la frappe d'un bloc de métal.

## Description

La sortie audio est un son de métal frappé comme sur un vibraphone. La méthode est un modèle physique développé d'après Perry Cook, mais recodé pour Csound.

## Syntaxe

ares **vibes** kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec

## Initialisation

*ihrd* -- la dureté de la baguette utilisé pour frapper. Compris entre 0 et 1. 0,5 est une valeur adaptée.

*ipos* -- l'endroit où le bloc est frappé, compris entre 0 et 1.

*imp* -- une table des impulsions de la frappe. Le fichier *marmstk1.wav* [examples/marmstk1.wav] contient une fonction adéquate créée à partir de mesures et l'on peut le charger dans une table *GEN01*. Il est aussi disponible à <ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modelling/>.

*ivfn* -- forme du tremolo, habituellement une table sinus, créée par une fonction

*idec* -- durée avant la fin de la note lorsqu'il y a une atténuation

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note.

*kvibf* -- Fréquence du tremolo en Hertz. L'intervalle conseillé va de 0 à 12.

*kvamp* -- Amplitude du tremolo.

## Exemples

Voici un exemple de l'opcode vibes. Il utilise les fichiers *vibes.csd* [examples/vibes.csd] et *marmstk1.wav* [examples/marmstk1.wav].

### Exemple 1105. Exemple de l'opcode vibes.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
```

```
; For Non-realtime ouput leave only the line below:
; -o vibes.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 10
nchnls = 2

; Instrument #1.
instr 1
; kamp = 20000
; kfreq = 440
; ihrd = 0.5
; ipos = p4
; imp = 1
; kvibf = 6.0
; kvamp = 0.05
; ivibfn = 2
; idec = 0.1
asig vibes 20000, 440, .5, p4 , 1, 6.0, 0.05, 2, .1
outs asig, asig
endin

</CsInstruments>
<CsScore>

; Table #1, the "marmstkl.wav" audio file.
f 1 0 256 1 "marmstkl.wav" 0 0 0
; Table #2, a sine wave for the vibrato.
f 2 0 128 10 1

; Play Instrument #1 for four seconds.
i 1 0 4 0.561
i 1 + 4 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*marimba*

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
Université de Bath, Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.47 de Csound

# vibr

vibr — Vibrato contrôlable par l'utilisateur, d'usage plus facile.

## Description

Vibrato contrôlable par l'utilisateur, d'usage plus facile.

## Syntaxe

```
kout vibr kAverageAmp, kAverageFreq, ifn
```

## Initialisation

*ifn* -- Numéro de la table de vibrato. Elle contient normalement une onde sinus ou triangle.

## Exécution

*kAverageAmp* -- Valeur d'amplitude moyenne du vibrato

*kAverageFreq* -- Valeur de fréquence moyenne du vibrato (en cps)

*vibr* est une version de *vibrato* d'usage plus facile. Il a le même moteur de génération que *vibrato*, mais les paramètres correspondant aux arguments d'entrée manquants sont codés en dur sur des valeurs par défaut.

## Exemples

Voici un exemple de l'opcode vibr. Il utilise le fichier *vibr.csd* [examples/vibr.csd].

### Exemple 1106. Exemple de l'opcode vibr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vibr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kaverageamp init 500
kaveragefreq init 4
kvib vibr kaverageamp, kaveragefreq, 1
```

```
asig poscil .8, 220+kvib, 1 ;add vibrato
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave

i 1 0 10

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*jitter, jitter2, vibrato*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.15

# vibrato

vibrato — Génère un vibrato naturel contrôlable par l'utilisateur.

## Description

Génère un vibrato naturel contrôlable par l'utilisateur.

## Syntaxe

```
kout vibrato kAverageAmp, kAverageFreq, kRandAmountAmp, \  
      kRandAmountFreq, kAmpMinRate, kAmpMaxRate, kcpsMinRate, \  
      kcpsMaxRate, ifn [, iphs
```

## Initialisation

*ifn* -- Numéro de la table de vibrato. Elle contient normalement une onde sinus ou triangle.

*iphs* -- (facultatif) Phase initiale de la table, exprimée comme une fraction d'une période (0 à 1). Avec une valeur négative, l'initialisation de la phase sera ignorée. La valeur par défaut est 0.

## Exécution

*kAverageAmp* -- Valeur de l'amplitude moyenne du vibrato

*kAverageFreq* -- Valeur de la fréquence moyenne du vibrato (en cps)

*kRandAmountAmp* -- Importance de la déviation aléatoire de l'amplitude

*kRandAmountFreq* -- Importance de la déviation aléatoire de la fréquence

*kAmpMinRate* -- Fréquence minimale des segments de déviation aléatoire de l'amplitude (en cps)

*kAmpMaxRate* -- Fréquence maximale des segments de déviation aléatoire de l'amplitude (en cps)

*kcpsMinRate* -- Fréquence minimale des segments de déviation aléatoire de la fréquence (en cps)

*kcpsMaxRate* -- Fréquence maximale des segments de déviation aléatoire de la fréquence (en cps)

*vibrato* produit un vibrato naturel contrôlable par l'utilisateur. Le concept consiste à varier aléatoirement la fréquence et l'amplitude de l'oscillateur générant le vibrato, afin de simuler les irrégularités d'un vibrato réel.

Afin d'avoir un contrôle total de ces variations aléatoires, plusieurs arguments sont présents en entrée. Les variations aléatoires sont obtenues à partir de deux suites séparées de segments, la première contrôlant les déviations d'amplitude, la seconde les déviations de fréquence. La durée moyenne de chaque segment dans chaque suite peut être raccourcie ou allongée par les arguments *kAmpMinRate*, *kAmpMaxRate*, *kcpsMinRate*, *kcpsMaxRate*, et les déviations par rapport aux valeurs d'amplitude et de fréquence moyennes peuvent être ajustées indépendamment au moyen de *kRandAmountAmp* et de *kRandAmountFreq*.

## Exemples

Voici un exemple de l'opcode vibrato. Il utilise le fichier *vibrato.csd* [examples/vibrato.csd].



## Exemple 1107. Exemple de l'opcode vibrato.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vibrato.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kaverageamp    init .5
kaveragefreq   init 5
krandamountamp line p4, p3, p5 ;increase random amplitude of vibrato
krandamountfreq init .3
kampminrate    init 3
kampmaxrate    init 5
kcpsminrate    init 3
kcpsmaxrate    init 5
kvib vibrato kaverageamp, kaveragefreq, krandamountamp, krandamountfreq, kampminrate, kampmaxrate, kcps
asig poscil .8, 220+kvib, 1 ;add vibrato
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave

i 1 0 15 .01 20

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*jitter, jitter2, vibr*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.15

# vincr

vincr — Accumule des signaux audio.

## Description

*vincr* incrémente une variable audio avec un autre signal, c-à-d qu'il accumule les valeurs dans sa sortie.

## Syntaxe

```
vincr accum, aincr
```

## Exécution

*accum* -- variable accumulateur de taux-a à incrémenter

*aincr* -- signal d'incrémentation

*vincr* (variable increment) et *clear* sont prévus pour être utilisés ensemble. *vincr* stocke la somme de deux variables audio dans la première variable (qui joue ainsi le rôle d'un accumulateur en polyphonie). L'accumulateur est habituellement une variable globale qui est utilisée pour combiner des signaux provenant de plusieurs sources (différents instruments ou instances d'instruments) pour un traitement ultérieur (par exemple via un effet global qui lit l'accumulateur) ou pour sortir le signal composé par un autre moyen que les opcodes *out* (par exemple via l'opcode *fout*). Après son utilisation, la variable accumulateur doit être remise à zéro au moyen de l'opcode *clear* (sinon elle sera saturée).

## Exemples

Voici un exemple de l'opcode *vincr*. Il utilise le fichier *vincr.csd* [examples/vincr.csd].

### Exemple 1108. Exemple de l'opcode *vincr*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vincr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gaReverb init 0

instr 1
```

```

idur = p3
kpitch = p4
a1 diskin2 "fox.wav", kpitch
a1 = a1*.5 ;reduce volume
vincr gaReverb, a1
endin

instr 99 ; global reverb
a1, ar reverbsc gaReverb, gaReverb, .8, 10000
outs gaReverb+a1, gaReverb+ar

clear gaReverb

endin

</CsInstruments>
<CsScore>

i1 0 3 1
i99 0 5
e

</CsScore>
</CsoundSynthesizer>

```

Voici un autre exemple de l'opcode vincr. Il utilise le fichier *vincr-complex.csd* [examples/vincr-complex.csd].

### Exemple 1109.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vincr-complex.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gaReverbSend init 0

instr 1

iamp = p4
ifreq = p5
aenv linseg 0.0, 0.1*p3, iamp, 0.6*p3, iamp, 0.3*p3, 0.0
aosc poscil aenv, ifreq, 1
vincr gaReverbSend, aosc
endin

instr 2 ; global reverb instrument

a1, ar reverbsc gaReverbSend, gaReverbSend, 0.85, 12000
outs gaReverbSend+a1, gaReverbSend+ar
clear gaReverbSend
endin

</CsInstruments>

```

```
<CsScore>
f1 0 4096 10 1

{ 4 CNT
{ 8 PARTIAL
; start time duration amplitude frequency
i1 [0.5 * $CNT.] [1 + ($CNT * 0.2)] [.04 + (~ * .02)] [800 + (200 * $CNT.) + ($PARTIAL. * 20.
}
}

i2 0 6
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*clear*

## Crédits

Auteur : Gabriel Maldonado  
Italie  
1999

Nouveau dans la version 3.56 de Csound

# vlimit

vlimit — Limitation et enroulement de signaux vectoriels.

## Description

Limite les éléments de signaux vectoriels de contrôle.

## Syntaxe

```
vlimit ifn, kmin, kmax, ielements
```

## Initialisation

*ifn* - numéro de la table hébergeant le vecteur à traiter.

*ielements* - nombre de composantes du vecteur.

## Exécution

*kmin* - valeur du seuil inférieur.

*kmax* - valeur du seuil supérieur.

*vlimit* fixe des limites inférieures et supérieures sur chaque élément du vecteur traité.

Ces opcodes sont semblables à *limit*, *wrap* et *mirror*, mais ils opèrent sur un signal vectoriel au lieu d'un signal scalaire.

Le résultat écrase les anciennes valeurs de *ifn1*, si celles-ci sont en dehors de l'intervalle min/max. Si l'on veut conserver le vecteur d'entrée, il faut utiliser l'opcode *vcopy* pour le copier dans une autre table.

Tous ces opcodes sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vlinseg

vlinseg — Générateur d'enveloppe vectoriel.

## Description

Génère des segments linéaires vectoriels.

## Syntaxe

```
vlinseg ifnout, ielements, ifn1, idur1, ifn2 [, idur2, ifn3 [...]]
```

## Initialisation

*ifnout* - numéro de la table hébergeant le signal vectoriel de sortie.

*ifn1* - vecteur de départ.

*ifn2, ifn3, etc.* - vecteur après *idurx* secondes.

*idur1* - durée en secondes du premier segment.

*idur2, idur3, etc.* - durée en secondes des segments suivants.

*ielements* - nombre de composantes des vecteurs.

## Exécution

Ces opcodes sont semblables à *linseg* et à *expseg*, mais ils opèrent avec des signaux vectoriels au lieu de signaux scalaires.

La sortie est un signal vectoriel de contrôle hébergé par *ifnout* (qui doit avoir été allouée au préalable), tandis que chaque point charnière de l'enveloppe est un vecteur de valeurs. Tous les points charnière doivent avoir le même nombre de composantes (*ielements*).

Tous ces opérateurs sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Exemples

Voici un exemple de l'opcode vlinseg. Il utilise le fichier *vlinseg.csd* [examples/vlinseg.csd].

### Exemple 1110. Exemple de l'opcode vlinseg.

```
<CsoundSynthesizer>  
<CsOptions>  
-odac -B441 -b441  
</CsOptions>  
<CsInstruments>
```

```

sr=44100
ksmps=10
nchnls=2

gilen init 32

gitable1 ftgen 0, 0, gilen, 10, 1
gitable2 ftgen 0, 0, gilen, 10, 1

gitable3 ftgen 0, 0, gilen, -7, 30, gilen, 35
gitable4 ftgen 0, 0, gilen, -7, 400, gilen, 450
gitable5 ftgen 0, 0, gilen, -7, 5000, gilen, 5500

instr 1
vcopy gitable2, gitable1, gilen
turnoff
endin

instr 2
vlinseg gitable2, 16, gitable3, 2, gitable4, 2, gitable5
endin

instr 3
kcount init 0
if kcount < 16 then
  kval table kcount, gitable2
  printk 0,kval
  kcount = kcount +1
else
  turnoff
endif
endin

</CsInstruments>
<CsScore>
i1 0 1
s
i2 0 10
i3 0 1
i3 1 1
i3 1.5 1
i3 2 1
i3 2.5 1
i3 3 1
i3 3.5 1
i3 4 1
i3 4.5 1

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Ecrit par Gabriel Maldonado.

Exemple par Andres Cabrera.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vlowres

vlowres — Un banc de filtres dans laquelle la fréquence de coupure peut être séparée sous le contrôle de l'utilisateur.

## Description

Un banc de filtres dans laquelle la fréquence de coupure peut être séparée sous le contrôle de l'utilisateur.

## Syntaxe

```
ares vlowres asig, kfco, kres, iord, ksep
```

## Initialisation

*iord* -- nombre total de filtres (1 à 10)

## Exécution

*asig* -- signal d'entrée

*kfco* -- fréquence de coupure (pas en Hz)

*kres* -- quantité de résonance

*ksep* -- séparation de la fréquence de coupure pour chaque filtre : le premier filtre a pour fréquence de coupure *kfreq*, le second *kfreq* + *ksep* et le troisième *kfreq* + 2\**ksep*, et ainsi de suite, en fonction du nombre de filtres.

*vlowres* (variable resonant lowpass filter) permet d'avoir une courbe de réponse variable dans les filtres à résonance. On peut l'imaginer comme un banc de filtres passe-bas à résonance, chacun avec la même résonance, connectés en série. La fréquence de coupure de chaque filtre peut varier avec les paramètres *kcfo* et *ksep*.

## Exemples

Voici un exemple de l'opcode vlowres. Il utilise le fichier *vlowres.csd* [examples/vlowres.csd].

### Exemple 1111. Exemple de l'opcode vlowres.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vlowres.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```



```

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kamp init p4
asig vco2 kamp, 110 ;saw wave
kfco line 30, p3, 300 ;vary the cutoff frequency from 30 to 300 Hz.
kres = 20
ksep = p5 ;different resonance values
iord = p6 ;and different number of filters
aout vlowres asig, kfco, kres, iord, ksep
aclp clip aout, 1, 1 ;avoid distortion
outs aclp, aclp

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine

s
i 1 0 10 .1 5 2 ;compensate volume and
i 1 + 10 .1 25 2 ;number of filters = 2
s
i 1 0 10 .01 5 6 ;compensate volume and
i 1 + 10 .04 15 6 ;number of filters = 6

e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.49 de Csound.

# vmap

vmap — Permute les éléments d'un vecteur selon les indices contenus dans un autre vecteur.

## Description

Transfert les éléments d'un vecteur vers un autre selon les indices contenus dans ce dernier.

## Syntaxe

```
vmap ifn1, ifn2, ielements [,idstoffset, isrcoffset]
```

## Initialisation

*ifn1* - numéro de la table dans laquelle le signal vectoriel sera copié et qui contient le vecteur d'indices.

*ifn2* - numéro de la table hébergeant le signal vectoriel à copier.

*ielements* - nombre d'éléments à traiter.

*idstoffset* - décalage d'indexation pour la table de destination *ifn1*.

*isrcoffset* - décalage d'indexation pour la table source *ifn2*.

## Exécution

*vmap* permute les éléments de *ifn2* selon les valeurs de la table *ifn1*. Les éléments de *ifn1* sont traités comme indices de la table *ifn2*, si bien que les valeurs des éléments de *ifn1* ne doivent pas dépasser la longueur de la table *ifn2*, sinon Csound rapportera une erreur. Les éléments de *ifn1* sont traités comme des entiers, chaque partie décimale étant tronquée. Il n'y a aucune interpolation lors de cette opération.

En pratique, les éléments de *ifn1* sont utilisés comme indices pour *ifn2*, et sont ensuite remplacés par les éléments correspondants de *ifn2*. *ifn1* doit être différente de *ifn2*, sinon les résultats sont imprévisibles. Csound générera une erreur d'initialisation si elles ne sont pas différentes.

Tous ces opérateurs (*vaddv*, *vsubv*, *vmultv*, *vdivv*, *vpowv*, *vexpv*, *vcopy* et *vmap*) sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Exemples

Voici un exemple de l'opcode *vmap*. Il utilise le fichier *vmap.csd* [examples/vmap.csd].

### Exemple 1112. Exemple de l'opcode *vmap*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>

<CsOptions>
; Select audio/midi flags here according to platform
```

```
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vmap.wav -W ;;; for file output any platform
</CsOptions>

<CsInstruments>
ksmps = 256
nchnls = 2
gisize = 64

gitable ftgen 0, 0, gisize, 10, 1 ;Table to be processed
gimap1 ftgen 0, 0, gisize, -7, gisize-1, gisize-1, 0 ; Mapping function to reverse table
gimap2 ftgen 0, 0, gisize, -5, 1, gisize-1, gisize-1 ; Mapping function for PWM
gimap3 ftgen 0, 0, gisize, -7, 1, (gisize/2)-1, gisize-1, 1, 1, (gisize/2)-1, gisize-1 ; Double frequency

instr 1 ;Hear an oscillator using gitable
asig oscil 10000, 440, gitable
outs asig,asig
endin

instr 2 ;Reverse the table (no sound change, except for a single click
vmap gimap1, gitable, gisize
vcopy_i gitable, gimap1, gisize
turnoff
endin

instr 3 ;Non-interpolated PWM (or phase waveshaping)
vmap gimap2, gitable, gisize
vcopy_i gitable, gimap2, gisize
turnoff
endin

instr 4 ;Double frequency
vmap gimap3, gitable, gisize
vcopy_i gitable, gimap3, gisize
turnoff
endin

</CsInstruments>

<CsScore>
i 1 0 8

i 2 2 1
i 3 4 1
i 4 6 1

e
</CsScore>

</CsoundSynthesizer>
```

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vmirror

vmirror — Limitation et enroulement de signaux vectoriels.

## Description

"Réfléchit" les éléments de signaux vectoriels de contrôle selon des seuils.

## Syntaxe

```
vmirror ifn, kmin, kmax, ielements
```

## Initialisation

*ifn* - numéro de la table hébergeant le vecteur à traiter.

*ielements* - nombre de composantes du vecteur.

## Exécution

*kmin* - valeur du seuil inférieur.

*kmax* - valeur du seuil supérieur.

*vmirror* "réfléchit" chaque élément du vecteur correspondant s'il dépasse les limites inférieure ou supérieure.

Ces opcodes sont semblables à *limit*, *wrap* et *mirror*, mais ils opèrent sur un signal vectoriel au lieu d'un signal scalaire.

Le résultat écrase les anciennes valeurs de *ifn1*, si celles-ci sont en dehors de l'intervalle min/max. Si l'on veut conserver le vecteur d'entrée, il faut utiliser l'opcode *vcopy* pour le copier dans une autre table.

Tous ces opcodes sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vmult

vmult — Multiplication d'un vecteur dans une table par une valeur scalaire.

## Description

Multiplication d'un vecteur dans une table par une valeur scalaire.

## Syntaxe

```
vmult ifn, kval, kelements [, kdstoffset] [, kverbose]
```

## Initialisation

*ifn* - numéro de la table hébergeant le signal vectoriel à traiter.

## Exécution

*kval* - valeur scalaire à multiplier.

*kelements* - nombre de composantes du vecteur.

*kdstoffset* - décalage d'indexation pour la table de destination (facultatif, vaut 0 par défaut).

*kverbose* - Indique si les avertissements sont affichés (vaut 0 par défaut).

*vmult* multiplie chaque élément du vecteur contenu dans la table *ifn* par *kval*, à partir de l'index de table *kdstoffset*. Cela permet de traiter une section particulière d'une table en spécifiant le décalage et le nombre d'éléments à traiter. Le décalage est compté à partir de 0, si bien que si aucun décalage n'est spécifié (ou s'il est fixé à 0), la table est modifiée depuis le début.

Noter que cet opcode est exécuté au taux-k si bien que la valeur de *kval* est multipliée à chaque période de contrôle. A utiliser avec précaution si l'on ne veut pas finir avec des nombres très grands (ou utiliser *vmult\_i*).

Ces opcodes (*vadd*, *vmult*, *vpow* et *vexp*) réalisent des opérations numériques entre un signal vectoriel de contrôle (hébergé par la table *ifn*), et un signal scalaire (*kval*). Le résultat est un nouveau vecteur qui écrase les anciennes valeurs de *ifn*. Tous ces opcodes travaillent au taux-k.

Les valeurs négatives sont valides pour *kdstoffset*. Les composantes du vecteur se trouvant en dehors de la table sont alors ignorées, et elles ne sont pas repliées autour de la table.

Si l'argument facultatif *kverbose* est différent de 0, l'opcode affichera des messages d'avertissement à chaque passe-k si les longueurs de table sont dépassées.

Dans tous ces opcodes, les vecteurs résultants sont stockés dans *ifn*, écrasant les vecteurs initiaux. Si l'on veut garder le vecteur initial, il faut utiliser *vcopy* ou *vcopy\_i* pour le copier dans une autre table. Tous ces opérateurs sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc. Ils peuvent aussi être utiles en conjonction avec les opcodes spectraux *pvsftw* et *pvsftr*.



## Note

Prière de noter que l'argument *elements* a changé dans la version 5.03 du taux-i au taux-k. Cela change le comportement de l'opcode dans le cas inhabituel où la variable de taux-i *ielements* est modifiée à l'intérieur de l'instrument, par exemple dans :

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

## Exemples

Voici un exemple de l'opcode *vmult*. Il utilise le fichier *vmult-2.csd* [examples/vmult-2.csd].

### Exemple 1113. Exemple de l'opcode *vmult*.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vmult ifn1, ival, ielements, idstoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
turnoff
endif

kcount = kcount + 1
endin

</CsInstruments>

<CsScore>
```

```
f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
e
```

```
</CsScore>
```

```
</CsoundSynthesizer>
```

Voici un autre exemple de l'opcode vmult. Il utilise le fichier *vmult.csd* [examples/vmult.csd].

### Exemple 1114. Exemple de l'opcode vmult.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
kr=4410
ksmps=10
nchnls=2

instr 1 ;table playback
ar lposcil 1, 1, 0, 262144, 1
outs ar,ar
endin

instr 2
vcopy 2, 1, 40000 ;copy vector from sample to empty table
vmult 5, 10000, 262144 ;scale noise to make it audible
vcopy 1, 5, 40000 ;put noise into sample
turnoff
endin

instr 3
vcopy 1, 2, 40000 ;put original information back in
turnoff
endin

</CsInstruments>
<CsScore>
f1 0 262144 -1 "beats.wav" 0 4 0
f2 0 262144 2 0

f5 0 262144 21 3 30000

i1 0 4
i2 3 1

s
i1 0 4
```

```
i3 3 1  
s  
  
i1 0 4  
  
</CsScore>  
</CsoundSynthesizer>
```

## Voir aussi

*vadd\_i*, *vadd\_i*, *vmult*, *vpow* et *vexp*.

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Exemple par Andrés Cabrera.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)



# vmult\_i

vmult\_i — Multiplication d'un vecteur dans une table par une valeur scalaire.

## Description

Multiplication d'un vecteur dans une table par une valeur scalaire.

## Syntaxe

```
vmult_i ifn, ival, ielements [, idstoffset]
```

## Initialisation

*ifn* - numéro de la table hébergeant le signal vectoriel à traiter.

*ival* - valeur scalaire à multiplier.

*ielements* - nombre de composantes du vecteur.

*idstoffset* - décalage d'indexation pour la table de destination.

## Exécution

*vmult\_i* multiplie chaque élément du vecteur contenu dans la table *ifn* par *ival*, à partir de l'index de table *idstoffset*. Cela permet de traiter une section particulière d'une table en spécifiant le décalage et le nombre d'éléments à traiter. Le décalage est compté à partir de 0, si bien que si aucun décalage n'est spécifié (ou s'il est fixé à 0), la table est modifiée depuis le début.

Cet opcode n'est exécuté qu'à l'initialisation. Il y a une version de taux-k de cet opcode appelée *vmult*.

Les valeurs négatives sont valides pour *idstoffset*. Les composantes du vecteur se trouvant en dehors de la table sont alors ignorées, et elles ne sont pas repliées autour de la table.

Dans tous ces opcodes, les vecteurs résultants sont stockés dans *ifn*, écrasant les vecteurs initiaux. Si l'on veut garder le vecteur initial, il faut utiliser *vcopy* ou *vcopy\_i* pour le copier dans une autre table. Tous ces opérateurs sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc. Ils peuvent aussi être utiles en conjonction avec les opcodes spectraux *pvsftw* et *pvsftr*.

## Exemples

Voici un exemple de l'opcode *vmult\_i*. Il utilise le fichier *vmult\_i.csd* [examples/vmult\_i.csd].

### Exemple 1115. Exemple de l'opcode *vmult\_i*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsoundOptions>
```

```

; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vmult_i ifn1, ival, ielements, idstoffset
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
  turnoff
endif

kcount = kcount + 1
endin

</CsInstruments>

<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
e

</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

*vadd*, *vadd*, *vmult*, *vpow\_i*, *vpow*, *vexp\_i* et *vexp*.

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Exemple par Andrés Cabrera.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vmultv

vmultv — Multiplication entre deux signaux vectoriels de contrôle.

## Description

Multiplication entre deux signaux vectoriels de contrôle.

## Syntaxe

```
vmultv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
```

## Initialisation

*ifn1* - numéro de la table hébergeant le premier vecteur à traiter.

*ifn2* - numéro de la table hébergeant le second vecteur à traiter.

## Exécution

*kelements* - nombre de composantes des deux vecteurs.

*kdstoffset* - décalage d'indexation pour la table de destination *ifn1* (vaut 0 par défaut).

*ksrcoffset* - décalage d'indexation pour la table source *ifn2* (vaut 0 par défaut).

*kverbose* - Indique si les avertissements sont affichés (vaut 0 par défaut).

*vmultv* multiplie deux signaux vectoriels de contrôle, chaque composante du premier vecteur n'étant traitée qu'avec la composante correspondante de l'autre vecteur. Chaque signal vectoriel est hébergé dans une table (*ifn1* et *ifn2*). Le nombre de composantes de chaque vecteur doit être identique.

Le résultat est un nouveau signal vectoriel de contrôle qui écrase les anciennes valeurs de *ifn1*. Si l'on veut garder l'ancien vecteur *ifn1*, il faut utiliser l'opcode *vcopy\_i* pour le copier dans une autre table. On peut utiliser *kdstoffset* et *ksrcoffset* pour spécifier des vecteurs à n'importe quelle position dans les tables.

Des valeurs négatives pour *kdstoffset* et *ksrcoffset* sont acceptables. Si *kdstoffset* est négatif, la partie du vecteur hors-limites est ignorée. Si *ksrcoffset* est négatif, les éléments hors-limites seront supposés valoir 1 (c'est-à-dire que les éléments de destination ne seront pas changés). Si des éléments pour le vecteur de destination sont au-delà de la taille de la table (point de garde inclus), ces éléments sont ignorés (les éléments ne sont pas repliés autour des tables). Si des éléments pour le vecteur source sont au-delà de la longueur de la table, ces éléments sont supposés valoir 1 (le vecteur de destination ne sera pas changé pour ces éléments).

Si l'argument facultatif *kverbose* est différent de 0, l'opcode affichera des messages d'avertissement à chaque passe-k si les longueurs de table sont dépassées.



### Avertissement

L'utilisation de la même table comme source et comme destination dans les versions antérieures à la 5.04 peut induire un comportement imprévu. A utiliser avec précaution.

Cet opcode travaille au taux-k (cela signifie qu'à chaque passe-k les vecteurs sont multipliés). Il y a une version de taux-i de cet opcode appelée *vmultv\_i*.



## Note

Prière de noter que l'argument *elements* a changé dans la version 5.03 du taux-i au taux-k. Cela change le comportement de l'opcode dans le cas inhabituel où la variable de taux-i *ielements* est modifiée à l'intérieur de l'instrument, par exemple dans :

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

Tous ces opérateurs (*vaddv*, *vsubv*, *vmultv*, *vddivv*, *vpowv*, *vexpv*, *vcopy* et *vmap*) sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Exemples

Voici un exemple de l'opcode *vmultv*. Il utilise le fichier *vmultv.csd* [exemples/vmultv.csd].

### Exemple 1116. Exemple de l'opcode *vmultv*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ifn2 = p5
ielements = p6
idstoffset = p7
isrcoffset = p8
kval init 25
vmultv ifn1, ifn2, ielements, idstoffset, isrcoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval
```

```

if (kcount == isize) then
    turnoff
endif

kcount = kcount + 1
endin

</CsInstruments>

<CsScore>

f 1 0 16 -7 1 16 17

f 2 0 16 -7 1 16 2

i2 0.0 0.2 1
i2 0.2 0.2 2
i1 0.4 0.01 1 2 5 3 8
i2 0.8 0.2 1
i1 1.0 0.01 1 2 5 10 -2
i2 1.2 0.2 1
i1 1.4 0.01 1 2 8 14 0
i2 1.6 0.2 1
i1 1.8 0.01 1 2 8 0 14
i2 2.0 0.2 1
i1 2.2 0.002 1 1 8 5 2
i2 2.4 0.2 1
e

</CsScore>

</CsoundSynthesizer>

```

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vmultv\_i

vmultv\_i — Multiplication entre deux signaux vectoriels de contrôle à l'initialisation.

## Description

Multiplication entre deux signaux vectoriels de contrôle à l'initialisation.

## Syntaxe

```
vmultv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
```

## Initialisation

*ifn1* - numéro de la table hébergeant le premier vecteur à traiter.

*ifn2* - numéro de la table hébergeant le second vecteur à traiter.

*ielements* - nombre de composantes des deux vecteurs.

*idstoffset* - décalage d'indexation pour la table de destination *ifn1* (vaut 0 par défaut).

*isrcoffset* - décalage d'indexation pour la table source *ifn2* (vaut 0 par défaut).

## Exécution

*vmultv\_i* multiplie deux signaux vectoriels de contrôle, chaque composante du premier vecteur n'étant traitée qu'avec la composante correspondante de l'autre vecteur. Chaque signal vectoriel est hébergé dans une table (*ifn1* et *ifn2*). Le nombre de composantes de chaque vecteur doit être identique.

Le résultat est un nouveau signal vectoriel de contrôle qui écrase les anciennes valeurs de *ifn1*. Si l'on veut garder l'ancien vecteur *ifn1*, il faut utiliser l'opcode *vcopy\_i* pour le copier dans une autre table. On peut utiliser *idstoffset* et *isrcoffset* pour spécifier des vecteurs à n'importe quelle position dans les tables.

Des valeurs négatives pour *idstoffset* et *isrcoffset* sont acceptables. Si *idstoffset* est négatif, la partie du vecteur hors-limites est ignorée. Si *isrcoffset* est négatif, les éléments hors-limites seront supposés valoir 1 (c'est-à-dire que les éléments de destination ne seront pas changés). Si des éléments pour le vecteur de destination sont au-delà de la taille de la table (point de garde inclus), ces éléments sont ignorés (les éléments ne sont pas repliés autour des tables). Si des éléments pour le vecteur source sont au-delà de la longueur de la table, ces éléments sont supposés valoir 1 (le vecteur de destination ne sera pas changé pour ces éléments).



### Avertissement

L'utilisation de la même table comme source et comme destination dans les versions antérieures à la 5.04 peut induire un comportement imprévu. A utiliser avec précaution.

Cet opcode travaille au taux-i. Il y a une version de taux-k de cet opcode appelée *vmultv*.

Tous ces opérateurs (*vaddv\_i*, *vsubv\_i*, *vmultv\_i*, *vdivv\_i*, *vpowv\_i*, *vexpv\_i*, *vcopy* et *vmap*) sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)



# voice

voice — Simulation d'une voix humaine.

## Description

Simulation d'une voix humaine.

## Syntaxe

ares **voice** kamp, kfreq, kphoneme, kform, kvibf, kvamp, ifn, ivfn

## Initialisation

*ifn*, *ivfn* -- numéros des deux tables contenant la forme d'onde de la porteuse et la forme d'onde du vibrato. Les fichiers *impuls20.aiff* [examples/impuls20.aiff], *ahh.aiff* [examples/ahh.aiff], *eee.aiff* [examples/eee.aiff] ou *ooo.aiff* [examples/ooo.aiff] conviennent pour la première, et la deuxième peut contenir une sinusoïde. Ces fichiers sont disponibles à <ftp://ftp.cs.bath.ac.uk/pub/dream/documentation/sounds/modeling/>.

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Frequency de la note. Elle peut varier pendant l'exécution.

*kphoneme* -- un entier compris entre 0 et 16, pour choisir les formants des sons :

- « eee », « ihh », « ehk », « aaa »,
- « ahh », « aww », « ohh », « uhh »,
- « uuu », « ooo », « rrr », « lll »,
- « mmm », « nnn », « nng », « ngg ».

Actuellement les phonèmes

- « fff », « sss », « thh », « shh »,
- « xxx », « hee », « hoo », « hah »,
- « bbb », « ddd », « jjj », « ggg »,
- « vvv », « zzz », « thz », « zhh »

ne sont pas disponibles (!)

*kform* -- gain pour le phonème. Des valeurs entre 0,0 et 1,2 sont recommandées.

*kvibf* -- fréquence du vibrato en Hertz. On suggère des valeurs entre 0 et 12

*kvamp* -- amplitude du vibrato

## Exemples

Voici un exemple de l'opcode `voice`. Il utilise les fichiers `voice.csd` [examples/voice.csd] et `impuls20.aiff` [examples/impuls20.aiff].

### Exemple 1117. Exemple de l'opcode `voice`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o voice.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kamp = p4
kphon = p5
asig voice kamp, 200, kphon, 0.488, 0, 1, 1, 2
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 256 1 "impuls20.aiff" 0 0 0 ;audio file for the carrier waveform
f 2 0 256 10 1 ;sine wave for the vibrato waveform

;      ampl phoneme
i 1 0 2 0.8 1
i 1 + . 0.6 2
i 1 + . 1.8 3
i 1 + . 15.0 4
i 1 + . 0.05 5
i 1 + . 0.06 6
i 1 + . 0.03 7
i 1 + . 0.0002 8
i 1 + . 0.1 9
i 1 + . 0.5 10
i 1 + . 100 11
i 1 + . 0.03 12
i 1 + . 0.04 13
i 1 + . 0.04 14
i 1 + . 0.04 15
i 1 + . 0.05 16

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitich (d'après Perry Cook)

Université de Bath, Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.47 de Csound

# vosim

vosim — Simulation vocale simple basée sur des pulsations glottales avec des caractéristiques de formant.

## Description

Cet opcode produit une simulation vocale simple basée sur des pulsations glottales avec des caractéristiques de formant. La sortie est une suite d'évènements sonores dans laquelle chaque élément est composé d'une explosion de pulsations sinusoïdales élevées au carré suivies par un silence. La méthode de synthèse VOSIM (VOcal SIMulation) fut développée par Kaegi et Tempelaars dans les années 1970.

## Syntaxe

ar **vosim** kamp, kFund, kForm, kDecay, kPulseCount, kPulseFactor, ifn [, iskip]

## Intialisation

*ifn* - une table sonore contenant normalement une demie-période d'une onde sinusoïdale, élevée au carré (voir les notes ci-dessous).

*iskip* - (facultatif) L'initialisation est ignorée, pour les notes liées.

## Exécution

*ar* - signal en sortie. Noter que la sortie est habituellement unipolaire - seulement positive.

*kamp* - amplitude de la sortie, l'amplitude de crête de la première pulsation dans chaque explosion.

*kFund* - hauteur fondamentale, en Hz. Chaque évènement dure  $1/kFund$  secondes.

*kForm* - fréquence du formant central. La longueur de chaque pulsation dans l'explosion vaut  $1/kForm$  secondes.

*kDecay* - facteur d'amortissement d'une pulsation à l'autre. Il est soustrait de l'amplitude à chaque nouvelle pulsation.

*kPulseCount* - nombre de pulsations dans la partie explosive de chaque évènement.

*kPulseFactor* - la largeur de pulsation est multipliée par cette valeur à chaque nouvelle pulsation. Cela provoque un glissement de formant. Si le factor est  $< 1.0$ , le formant monte, s'il est  $> 1.0$  chaque nouvelle pulsation est plus longue et ainsi le format descend. La hauteur finale du formant vaut  $kForm * \text{pow}(kPulseFactor, kPulseCount)$

La sortie de *vosim* est une suite d'évènements sonores, dans laquelle chaque évènement est composé d'une explosion de pulsations sinusoïdales élevées au carré suivies par un silence. La durée totale des évènements détermine la fréquence fondamentale. La longueur de chaque impulsion individuelle dans l'explosion de sinus au carré produit une bande de fréquence formantique. La largeur du formant est déterminée par le taux de silence par rapport aux pulsations (voir ci-dessous). Le résultat final est aussi modelé par le facteur d'atténuation entre pulsations.

Le fait qu'aucune fonction GEN ne crée une onde sinusoïdale élevée au carré telle quelle pose un petit problème dans l'utilisation de cet opcode. On peut créer la table appropriée depuis la partition en utilisant quelque chose comme ce qui suit.

```
; use GEN09 to create half a sine in table 17
```

```
f 17 time size 9 0.5 1 0
; run instr 101 on table 17 for a single init-pass
i 101 0 0 17
```

On peut aussi le faire avec un instrument qui remplit une ftable dans l'orchestre :

```
; square each point in table #p4. This should be run as init-only, just once in the performance.
instr 101

  index tablen p4

  index = index - 1 ; start from last point
loop:

  ival table index, p4

  ival = ival * ival

  tableiw ival, index, p4

  index = index - 1

  if index < 0 igoto endloop
    igoto loop
endloop:
endin
```



## Limites de Paramètre

Le nombre de pulsations multiplié par la largeur de pulsation doit être inclus dans la longueur de l'évènement ( $1/kFund$ ). Si ce n'est pas le cas, l'algorithme fonctionne quand même, mais les pulsations qui se trouveraient en dehors de l'évènement ne sont pas démarrées. Cela peut introduire un silence à la fin de l'évènement même s'il n'est pas désiré. En conséquence,  $kForm$  doit être supérieur à  $kFund$ , sinon il n'y aura que du silence en sortie.

*Vosim* a été créé pour émuler des sons vocaux en modélisant des impulsions glottales. On peut créer des sons riches en combinant plusieurs instances de *vosim* avec différents paramètres. Le fait que le signal ne soit pas à bande limitée est un inconvénient. Mais comme les auteurs le souligne, l'atténuation des composants aigus est de -60 dB à six fois la fréquence fondamentale. On peut également modifier le signal en changeant le signal source dans la table de lecture. La technique a un intérêt historique et peut produire des sons riches à moindre frais (chaque échantillon ne nécessite qu'une lecture dans la table suivie d'une seule multiplication pour l'atténuation).

Comme indiqué, la largeur de bande du formant dépend du rapport entre l'explosion de pulsation et le silence dans un évènement. Mais ce n'est pas un paramètre indépendant : la fondamentale fixe la longueur de l'évènement tandis que le centre du formant définit la longueur de la pulsation. Il est ainsi impossible de garantir un rapport explosion/silence spécifique, car la longueur de l'explosion doit être un multiple entier de la longueur de la pulsation. La chute des pulsations peut être utilisée pour lisser la transition de  $N$  à  $N \pm 1$  pulsations, mais il y aura toujours des paliers dans le profil spectral de la sortie. L'exemple de code ci-dessous montre une telle approche.

Tous les paramètres en entrée sont de taux-k. Les paramètres en entrée ne sont utilisés que pour déterminer chaque nouvel évènement (ou grain). L'amplitude de l'évènement est fixée pour chaque évènement à l'initialisation. Pour les valeurs usuelles des paramètres, lorsque  $ksmps < 500$ , les paramètres de taux-k sont mis à jour plus souvent que les évènements ne sont créés. Dans tous les cas, il n'y aura pas de bruit à large bande injecté dans le système à cause d'entrées de taux-k mises à jour moins souvent qu'elles ne sont lues, mais quelques artefacts peuvent être créés.

L'opcode devrait se comporter raisonnablement pour toutes les entrées. Quelques détails :

- $kFund < 0$  : il est forcé à une valeur positive - pas de points dans des événements "inversés".
- $kFund == 0$  : cela conduit à un événement de longueur "infinie", c'est-à-dire une explosion de pulsation suivie par un très long silence indéfini.
- $kForm == 0$  : cela conduit à une pulsation de longueur infinie, ainsi aucune pulsation n'est générée (c'est-à-dire silence).
- $kForm < 0$  : la table est lue à l'envers. Si la table est symétrique,  $kform$  et  $-kform$  donneront des sorties identiques bit à bit.
- $kPulseFactor == 0$  : la seconde pulsation en avant est zéro. Voir (c).
- $kPulseFactor < 0$  : les pulsations lisent la table alternativement à l'endroit et à l'envers.

Avec une table de pulsation asymétrique, un  $kForm$  ou un  $kPulseFactor$  négatifs peuvent être utiles.

## Exemples

Voici un exemple de l'opcode vosim. Il utilise le fichier *vosim.csd* [examples/vosim.csd].

### Exemple 1118. Exemple de l'opcode vosim.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in
;-odac           -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
-o vosim.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr      = 44100
ksmps   = 100
nchnls  = 1

;#####
; By Rasmus Ekman 2008

; Square each point in table #p4. This should only be run once in the performance.
instr 10

  index tableng p4
  index = index - 1 ; start from last point
loop:
  ival table index, p4
  ival = ival * ival
  tableiw ival, index, p4
  index = index - 1
  if index < 0 igoto endloop
  igoto loop
endloop:
endin

;#####

; vosim instrument. Sweeps kfund and kform between start-end values.
; Attempts to smooth spectral transitions as pulse count changes
```

```

; p4:      amp
; p5, p6: fund beg-end
; p7, p8: form beg-end
; p9:      amp decay (ignored)
; p10:     pulse count (ignored - calc internally)
; p11:     pulse length mod
; p12:     skip (for tied events)
; p13:     don't fade out (if followed by tied note)
instr 1
  kamp    init p4
  ; freq start, end
  kfund   line p5, p3, p6
  ; formant start, end
  kform   line p7, p3, p8

  ; Get many pulses as we can fit
  kPulseCount = (kform / kfund) ;init p10

  ; Attempt to smooth steps between formant bandwidth,
  ; matching decay to the current pulse count and remaining silent part
  kPulseREM = kPulseCount - int(kPulseCount) ; Pulse remainder (empty samples after last pulse)
  kDroprate = kPulseREM / (kPulseCount-1) ; Decay per pulse (after the 1st)
  kDecay = kamp - kDroprate
  ; Guard against kamp going negative (since kDecay is subtracted from each pulse)
  if (kDecay * (kPulseCount-1)) > kamp then
    kDecay = kamp / kPulseCount
  endif

  ; Try this to get more bumpy spectral changes when pulse count changes
  ;kDecay = p9

  kPulseFactor init p11

; ar vosim kamp, kFund, kForm, kDecay, kPulseCount, kPulseFactor, ifn [, iskip]
ar1 vosim kamp, kfund, kform, kDecay, kPulseCount, kPulseFactor, 17, p12

  ; scale amplitude for 16-bit files, with quick fade out
  amp init 20000
  if (p13 != 0) goto nofade
  amp linseg 20000, p3-.02, 20000, .02, 0
nofade:
  out ar1 * amp
endin

</CsInstruments>
<CsScore>

f1      0 32768 9 1 1 0 ; sine wave
f17     0 32768 9 0.5 1 0 ; half sine wave
i10 0 0 17 ; init run only, square table 17

; Vosim score

; Picking some formants from the table in Csound manual

;      p4=amp fund      form      decay pulses pulsemod [skip] nofade
; tenor a -> e
i1 0 .5 .5 280 240 650 400 .03 5 1 0 0
i1 . . .3 . . 1080 1700 .03 5 . . .
i1 . . .2 . . 2650 2600 .03 5 . . .
i1 . . .15 . . 2900 3200 .03 5 . . .

; tenor a -> o
i1 0.6 .2 .5 300 210 650 400 .03 5 1 0 1
i1 . . .3 . . 1080 800 .03 5 . . .
i1 . . .2 . . 2650 2600 .03 5 . . .

```

```

i1 . . .15 . . 2900 2800 .03 5 . . .
; tenor o -> aah
i1 .8 .3 .5 210 180 400 650 .03 5 1 1 1
i1 . . .3 . . 800 1080 .03 5 . . .
i1 . . .2 . . 2600 2650 .03 5 . . .
i1 . . .15 . . 2800 2900 .03 5 . . .
; tenor aa -> i
i1 1.1 .2 .5 180 250 650 290 .03 5 1 1 1
i1 . . .3 . . 1080 1870 .03 5 . . .
i1 . . .2 . . 2650 2800 .03 5 . . .
i1 . . .15 . . 2900 3250 .03 5 . . .
; tenor i -> u
i1 1.3 .3 .5 250 270 290 350 .03 5 1 1 0
i1 . . .3 . . 1870 600 .03 5 . . .
i1 . . .2 . . 2800 2700 .03 5 . . .
i1 . . .15 . . 3250 2900 .03 5 . . .

e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*fof, fof2*

## Crédits

Auteur : Rasmus Ekman  
Mars 2008



# vphaseseg

vphaseseg — SHV (Synthèse Hyper Vectorielle) à une dimension.

## Description

*vphaseseg* permet une SHV (Synthèse Hyper Vectorielle) à une dimension.

## Syntaxe

```
vphaseseg kphase, ioutab, ielems, itab1, idist1, itab2 \
[,idist2, itab3, ... ,idistN-1, itabN]
```

## Initialisation

*ioutab* - numéro de la table de sortie.

*ielem* - nombre d'éléments à traiter.

*itab1,...,itabN* - numéros des tables de points pivots.

*idist1,...,idistN-1* - distances entre les points pivots en valeurs de pourcentage.

## Exécution

*kphase* - pointeur de phase.

*vphaseseg* retourne les coordonnées de points de section d'un chemin dans un espace à N dimensions. Les coordonnées des points de section sont stockées dans une table en sortie. Le nombre de dimensions de l'espace à N dimensions est déterminé par l'argument *ielem* qui est égal à N et qui peut recevoir n'importe quel nombre. Pour définir le chemin, l'utilisateur doit fournir un ensemble de points de l'espace à N dimensions, appelés points pivots. Les coordonnées de chaque point pivot doivent se trouver dans une table différente. Le nombre de coordonnées à insérer dans chaque table de point pivot doit évidemment être égal à l'argument *ielem*. Il peut y avoir n'importe quel nombre de tables de points pivots remplies par l'utilisateur.

La Synthèse Hyper Vectorielle utilise deux sortes d'espaces. Le premier espace est l'espace à N dimensions dans lequel le chemin est défini, cet espace étant appelé l'espace des paramètres variants dans le temps (ou ESPACE A). Le chemin appartenant à cet espace est parcouru en déplaçant un point dans le second espace qui a normalement un nombre de dimensions inférieur à celui du premier espace. Actuellement, le point en mouvement est la projection du point correspondant de l'espace à N dimensions (on pourrait aussi le considérer comme une section du chemin). Le second espace est appelé espace de déplacement du pointeur de l'utilisateur (ou ESPACE B) et, dans le cas de l'opcode *vphaseseg*, il n'a qu'UNE DIMENSION. L'espace B est parcouru au moyen de l'argument *kphase* (qui est une sorte de pointeur de chemin), compris entre 0 et 1. La sortie correspondant à la valeur courante du pointeur est stockée dans la table *ioutab*, dont les données peuvent être utilisées ultérieurement pour contrôler des paramètres de synthèse.

Dans *vphaseseg*, chaque point pivot est séparé du suivant par une distance exprimée en pourcentage, la longueur totale du chemin étant égale à la somme de toutes ces distances. Ainsi les distances entre les points pivots peuvent être différentes, à l'inverse des SHV dans lesquelles l'espace B a plus d'une dimension. Dans ce dernier cas, la distance entre les points pivots DOIT être LA MEME pour tous les intervalles.

## Voir aussi

*hvs1, hvs2, hvs3*

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 5.06

# vport

vport — Lignes à retard vectorielles au taux de contrôle.

## Description

Génère une sorte de portamento "vectoriel".

## Syntaxe

```
vport ifn, khtime, ielements [, ifnInit]
```

## Initialisation

*ifn* - numéro de la table contenant le vecteur de sortie.

*ielements* - nombre de composantes des deux vecteurs.

*ifnInit* (facultatif) - numéro de la table contenant un vecteur dont les composantes sont les valeurs initiales du portamento.

## Exécution

*vport* est semblable à *port*, mais il opère sur des signaux vectoriels au lieu de signaux scalaires. Chaque composante du vecteur est traitée comme un signal de contrôle indépendant. Les vecteurs d'entrée et de sortie sont placés dans la même table et le vecteur de sortie écrase le vecteur d'entrée. Si l'on veut conserver le vecteur d'entrée, il faut utiliser l'opcode *vcopy* pour le copier dans une autre table.

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vpow

vpow — Elévation de chaque composante d'un vecteur à une puissance scalaire.

## Description

Elévation de chaque composante d'un vecteur à une puissance scalaire.

## Syntaxe

```
vpow ifn, kval, kelements [, kdstoffset] [, kverbose]
```

## Initialisation

*ifn* - numéro de la table hébergeant le signal vectoriel à traiter.

## Exécution

*kval* - valeur scalaire, puissance à laquelle seront élevés les éléments de *ifn*.

*kelements* - nombre de composantes du vecteur.

*kdstoffset* - décalage d'indexation pour la table de destination (facultatif, vaut 0 par défaut).

*kverbose* - Indique si les avertissements sont affichés (vaut 0 par défaut).

*vpow* élève chaque élément du vecteur contenu dans la table *ifn* à la puissance *kval*, à partir de l'index de table *kdstoffset*. Cela permet de traiter une section particulière d'une table en spécifiant le décalage et le nombre d'éléments à traiter. Le décalage est compté à partir de 0, si bien que si aucun décalage n'est spécifié (ou s'il est fixé à 0), la table est modifiée depuis le début.

Noter que cet opcode est exécuté au taux-k si bien que la valeur de *kval* est traitée à chaque période de contrôle. A utiliser avec précaution si l'on ne veut pas finir avec des nombres très grands ou très petits (ou utiliser *vpow\_i*).

Ces opcodes (*vadd*, *vmult*, *vpow* et *vexp*) réalisent des opérations numériques entre un signal vectoriel de contrôle (hébergé par la table *ifn*), et un signal scalaire (*kval*). Le résultat est un nouveau vecteur qui écrase les anciennes valeurs de *ifn*. Tous ces opcodes travaillent au taux-k.

Les valeurs négatives sont valides pour *kdstoffset*. Les composantes du vecteur se trouvant en dehors de la table sont alors ignorées, et elles ne sont pas repliées autour de la table.

Si l'argument facultatif *kverbose* est différent de 0, l'opcode affichera des messages d'avertissement à chaque passe-k si les longueurs de table sont dépassées.

Dans tous ces opcodes, les vecteurs résultants sont stockés dans *ifn*, écrasant les vecteurs initiaux. Si l'on veut garder le vecteur initial, il faut utiliser *vcopy* ou *vcopy\_i* pour le copier dans une autre table. Tous ces opérateurs sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc. Ils peuvent aussi être utiles en conjonction avec les opcodes spectraux *pvsftw* et *pvsftr*.



## Note

Prière de noter que l'argument *elements* a changé dans la version 5.03 du taux-i au taux-k. Cela change le comportement de l'opcode dans le cas inhabituel où la variable de taux-i *ielements* est modifiée à l'intérieur de l'instrument, par exemple dans :

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

## Exemples

Voici un exemple de l'opcode *vpow*. Il utilise le fichier *vpow.csd* [examples/vpow.csd].

### Exemple 1119. Exemples de l'opcode *vpow*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vpow ifn1, ival, ielements, idstoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
turnoff
endif

kcount = kcount + 1
endin
```

```

</CsInstruments>

<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
e

</CsScore>

</CsoundSynthesizer>

```

Voici un autre exemple de l'opcode `vpow`. Il utilise le fichier `vpow-2.csd` [examples/vpow-2.csd].

### Exemple 1120.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vpow-2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ain    diskin2 "fox.wav", 1    ;soundfile
fsrc    pvsanal ain, 1024, 256, 1024, 1
ifn    ftgen    0, 0, 1024/2, 2, 0    ;create empty function table for the 513 bins
kflag    pvsftw fsrc,ifn    ;export only amplitudes to table
kval    line .001, p3, 1    ;start with big distortion, cahnge over note duration to clean sound
kbin    line p4, p3, p5    ;vary the bins
vpow    ifn, kval, kbin, 0    ;note that this operation is applied each k-cycle!
;vpow    ifn, kval, kbin, 10    ;if you set kdstoffset to 10 it will affect bins 10+(kbin line p4, p3, p5,
    pvsftr    fsrc,ifn    ;read modified data back to fsrc
aout    pvsynth fsrc    ;and resynth
    outs aout*p6, aout*p6    ;adjust volume to compensate

endin
</CsInstruments>
<CsScore>

i 1 0 4 100 100 .02 ;first 100 bins are affected
i 1 + 4 10 10 .1 ;first 10 bins
i 1 + 4 1 400 .05 ;sweep from bin 1 to 400
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*vadd\_i*, *vmult*, *vpow* et *vexp*.

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vpow\_i

vpow\_i — Elévation de chaque composante d'un vecteur à une puissance scalaire à l'initialisation.

## Description

Elévation de chaque composante d'un vecteur à une puissance scalaire à l'initialisation.

## Syntaxe

```
vpow_i ifn, ival, ielements [, idstoffset]
```

## Initialisation

*ifn* - numéro de la table hébergeant le signal vectoriel à traiter.

*ielements* - nombre de composantes du vecteur.

*ival* - valeur scalaire, puissance à laquelle seront élevés les éléments de *ifn*.

*idstoffset* - décalage d'indexation pour la table de destination.

## Exécution

*vpow\_i* élève chaque élément du vecteur contenu dans la table *ifn* à la puissance *ival*, à partir de l'index de table *idstoffset*. Cela permet de traiter une section particulière d'une table en spécifiant le décalage et le nombre d'éléments à traiter. Le décalage est compté à partir de 0, si bien que si aucun décalage n'est spécifié (ou s'il est fixé à 0), la table est modifiée depuis le début.

Cet opcode n'est exécuté qu'à l'initialisation. Il y a une version de taux-k de cet opcode appelée *vpow*.

Les valeurs négatives sont valides pour *idstoffset*. Les composantes du vecteur se trouvant en dehors de la table sont alors ignorées, et elles ne sont pas repliées autour de la table.

Dans tous ces opcodes, les vecteurs résultants sont stockés dans *ifn*, écrasant les vecteurs initiaux. Si l'on veut garder le vecteur initial, il faut utiliser *vcopy* ou *vcopy\_i* pour le copier dans une autre table. Tous ces opérateurs sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc. Ils peuvent aussi être utiles en conjonction avec les opcodes spectraux *pvsftw* et *pvsftr*.

## Exemples

Voici un exemple de l'opcode *vpow\_i*. Il utilise le fichier *vpow\_i.csd* [examples/vpow\_i.csd].

### Exemple 1121. Exemples de l'opcode *vpow\_i*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsoundOptions>
```



```

; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ival = p5
ielements = p6
idstoffset = p7
kval init 25
vpow_i ifn1, ival, ielements, idstoffset
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval

if (kcount == isize) then
  turnoff
endif

kcount = kcount + 1
endin

</CsInstruments>

<CsScore>

f 1 0 16 -7 1 16 17

i2 0.0 0.2 1
i1 0.4 0.01 1 2 3 4
i2 0.8 0.2 1
i1 1.0 0.01 1 0.5 5 -3
i2 1.2 0.2 1
i1 1.4 0.01 1 1.5 10 12
i2 1.6 0.2 1
e

</CsScore>

</CsoundSynthesizer>

```

Voici un autre exemple de l'opcode `vpow_i`. Il utilise le fichier `vpow_i-2.csd` [examples/vpow\_i-2.csd].

## Exemple 1122.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:

```

```
; -o vpow_i-2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gifil ftgen 1, 0, 0, 1, "fox.wav", 0, 0, 1

instr 1

ival      = p4      ;different distortion settings
ielements = p5
idstoffset = p6      ;index offset
vpow_i 1, ival, ielements, idstoffset
asig lposcil 1, 1, 0, 0, 1
      outs asig, asig

endin
</CsInstruments>
<CsScore>

i1 0 2.7 .5 70000 0 ;no offset
i1 3 2.7 .01 50000 70000 ;add another period of distortion, starting at sample 70000

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*vadd\_i*, *vmult\_i*, *vpow* et *vexp\_i*.

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vpowv

vpowv — Elévation de puissance entre deux signaux vectoriels de contrôle.

## Description

Elévation de puissance entre deux signaux vectoriels de contrôle.

## Syntaxe

```
vpowv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
```

## Initialisation

*ifn1* - numéro de la table hébergeant le premier vecteur à traiter.

*ifn2* - numéro de la table hébergeant le second vecteur à traiter.

## Exécution

*kelements* - nombre de composantes des deux vecteurs.

*kdstoffset* - décalage d'indexation pour la table de destination *ifn1* (vaut 0 par défaut).

*ksrcoffset* - décalage d'indexation pour la table source *ifn2* (vaut 0 par défaut).

*kverbose* - Indique si les avertissements sont affichés (vaut 0 par défaut).

*vpowv* élève chaque élément de *ifn1* à la puissance égale à la valeur de l'élément correspondant de *ifn2*. Chaque signal vectoriel est hébergé dans une table (*ifn1* et *ifn2*). Le nombre de composantes de chaque vecteur doit être identique.

Le résultat est un nouveau signal vectoriel de contrôle qui écrase les anciennes valeurs de *ifn1*. Si l'on veut garder l'ancien vecteur *ifn1*, il faut utiliser l'opcode *vcopy\_i* pour le copier dans une autre table. On peut utiliser *kdstoffset* et *ksrcoffset* pour spécifier des vecteurs à n'importe quelle position dans les tables.

Des valeurs négatives pour *kdstoffset* et *ksrcoffset* sont acceptables. Si *kdstoffset* est négatif, la partie du vecteur hors-limites est ignorée. Si *ksrcoffset* est négatif, les éléments hors-limites seront supposés valoir 1 (c'est-à-dire que les éléments de destination ne seront pas changés). Si des éléments pour le vecteur de destination sont au-delà de la taille de la table (point de garde inclus), ces éléments sont ignorés (les éléments ne sont pas repliés autour des tables). Si des éléments pour le vecteur source sont au-delà de la longueur de la table, ces éléments sont supposés valoir 1 (le vecteur de destination ne sera pas changé pour ces éléments).

Si l'argument facultatif *kverbose* est différent de 0, l'opcode affichera des messages d'avertissement à chaque passe-k si les longueurs de table sont dépassées.



### Avertissement

L'utilisation de la même table comme source et comme destination dans les versions antérieures à la 5.04 peut induire un comportement imprévu. A utiliser avec précaution.

Cet opcode travaille au taux-k (cela signifie qu'à chaque passe-k les vecteurs sont traités). Il y a une version de taux-i de cet opcode appelée *vpowv\_i*.



## Note

Prière de noter que l'argument *elements* a changé dans la version 5.03 du taux-i au taux-k. Cela change le comportement de l'opcode dans le cas inhabituel où la variable de taux-i *ielements* est modifiée à l'intérieur de l'instrument, par exemple dans :

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

Tous ces opérateurs (*vaddv*, *vsubv*, *vmultv*, *vddivv*, *vpowv*, *vexpv*, *vcopy* et *vmap*) sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Exemples

Voici un exemple de l'opcode *vpowv*. Il utilise le fichier *vpowv.csd* [examples/vpowv.csd].

### Exemple 1123. Exemple de l'opcode *vpowv*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ifn2 = p5
ielements = p6
idstoffset = p7
isrcoffset = p8
kval init 25
vpowv ifn1, ifn2, ielements, idstoffset, isrcoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval
```

```

if (kcount == isize) then
    turnoff
endif

kcount = kcount + 1
endin

</CsInstruments>

<CsScore>

f 1 0 16 -7 1 16 17

f 2 0 16 -7 1 16 2

i2 0.0 0.2 1
i2 0.2 0.2 2
i1 0.4 0.01 1 2 5 3 8
i2 0.8 0.2 1
i1 1.0 0.01 1 2 5 10 -2
i2 1.2 0.2 1
i1 1.4 0.01 1 2 8 14 0
i2 1.6 0.2 1
i1 1.8 0.01 1 2 8 0 14
i2 2.0 0.2 1
e

</CsScore>

</CsoundSynthesizer>

```

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vpowv\_i

vpowv\_i — Elévation de puissance entre deux signaux vectoriels de contrôle à l'initialisation.

## Description

Elévation de puissance entre deux signaux vectoriels de contrôle à l'initialisation.

## Syntaxe

```
vpowv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
```

## Initialisation

*ifn1* - numéro de la table hébergeant le premier vecteur à traiter.

*ifn2* - numéro de la table hébergeant le second vecteur à traiter.

*ielements* - nombre de composantes des deux vecteurs.

*idstoffset* - décalage d'indexation pour la table de destination *ifn1*.

*isrcoffset* - décalage d'indexation pour la table source *ifn2*.

## Exécution

*vpowv\_i* élève chaque élément de *ifn1* à la puissance égale à la valeur de l'élément correspondant de *ifn2*. Chaque signal vectoriel est hébergé dans une table (*ifn1* et *ifn2*). Le nombre de composantes de chaque vecteur doit être identique.

Le résultat est un nouveau signal vectoriel de contrôle qui écrase les anciennes valeurs de *ifn1*. Si l'on veut garder l'ancien vecteur *ifn1*, il faut utiliser l'opcode *vcopy\_i* pour le copier dans une autre table. On peut utiliser *idstoffset* et *isrcoffset* pour spécifier des vecteurs à n'importe quelle position dans les tables.

Des valeurs négatives pour *idstoffset* et *isrcoffset* sont acceptables. Si *idstoffset* est négatif, la partie du vecteur hors-limites est ignorée. Si *isrcoffset* est négatif, les éléments hors-limites seront supposés valoir 1 (c'est-à-dire que les éléments de destination ne seront pas changés). Si des éléments pour le vecteur de destination sont au-delà de la taille de la table (point de garde inclus), ces éléments sont ignorés (les éléments ne sont pas repliés autour des tables). Si des éléments pour le vecteur source sont au-delà de la longueur de la table, ces éléments sont supposés valoir 1 (le vecteur de destination ne sera pas changé pour ces éléments).



### Avertissement

L'utilisation de la même table comme source et comme destination dans les versions antérieures à la 5.04 peut induire un comportement imprévu. A utiliser avec précaution.

Cet opcode travaille à l'initialisation. Il y a une version de taux-k de appelée *vpowv*.

Tous ces opérateurs (*vaddv\_i*, *vsubv\_i*, *vmultv\_i*, *vdivv\_i*, *vpowv\_i*, *vexpv\_i*, *vcopy* et *vmap*) sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2* etc.

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

## vpvoc

**vpvoc** — Implémente une reconstruction de signal au moyen d'un vocoder de phase basé sur la TFR et une enveloppe supplémentaire.

## Description

Implémente une reconstruction de signal au moyen d'un vocoder de phase basé sur la TFR et une enveloppe supplémentaire.

## Syntaxe

```
ares vpvoc kimpnt, kfmod, ifile [, ispecwp] [, ifn]
```

## Initialisation

*ifile* -- le numéro pvoc (n dans pvoc.n) ou le nom entre guillemets du fichier d'analyse obtenu au moyen de *pvanal*. (Voir *pvoc*.)

*ispecwp* (facultatif, 0 par défaut) -- s'il est différent de zéro, l'opcode tente de préserver l'enveloppe spectrale tandis que le contenu fréquentiel est varié par *kfmod*. Vaut zéro par défaut.

*ifn* (facultatif, 0 par défaut) -- table de fonction facultative contenant l'information de contrôle pour *vpvoc*. Si *ifn* = 0, le contrôle est dérivé en interne d'une unité *tableseg* ou *tablexseg* précédente. Vaut 0 par défaut. (Nouveau dans la version 3.59 de Csound.)

## Exécution

*kimpnt* -- l'écoulement du temps en secondes dans le fichier d'analyse. *kimpnt* doit toujours être positif, mais il peut avancer ou reculer, rester stationnaire ou être discontinu, comme pointeur dans le fichier d'analyse.

*kfmod* -- un facteur de transposition au taux-k : une valeur de 1 signifie pas de transposition, 1.5 transpose vers le haut d'une quinte parfaite et 0.5 transpose vers le bas d'une octave.

Cette implémentation de *pvoc* a été écrite à l'origine par Dan Ellis. Elle est basée en partie sur le système de Mark Dolson, mais le concept de pré-analyse est nouveau. L'extraction spectrale et le mappage d'amplitude (nouveau dans la version 3.56 de Csound) ont été ajoutés par Richard Karpen en se basant sur les fonctions dans SoundHack par Tom Erbe.

*vpvoc* est identique à *pvoc* mais il utilise la table de fonction d'un *tableseg* ou d'un *tablexseg* précédent (passée en interne à *vpvoc*) comme enveloppe pour les amplitudes des canaux de données analysées. Une table spécifiée par *ifn* peut être utilisée de manière optionnelle.

Il en résulte une enveloppe spectrale. La taille de la fonction utilisée dans *tableseg* doit être *tailletrame*/2, où *tailletrame* est le nombre de bins dans le fichier d'analyse du vocoder de phase utilisé par *vpvoc*. Chaque position dans la table est utilisée pour échelonner un seul bin d'analyse. En utilisant différentes fonctions pour *ifn1*, *ifn2*, etc.. dans le *tableseg*, l'enveloppe spectrale devient dynamique. Voir aussi *tableseg* et *tablexseg*.

## Exemples

L'exemple suivant avec *vpvoc*, montre l'utilisation de fonctions telles que



```
f 1 0 256 5 .001 128 1 128 .001
f 2 0 256 5 1 128 .001 128 1
f 3 0 256 7 1 256 1
```

pour pondérer les amplitudes des bins d'analyse séparés.

```
ktime    line        0, p3, 3 ; pointeur de temps, en secondes, dans le fichier
          tablexseg   1, p3*0.5, 2, p3*0.5, 3
apv      vpvoc       ktime, 1, "pvoc.file"
```

Le résultat sera une « enveloppe spectrale » variant dans le temps, appliquée aux données d'analyse du vocoder de phase. Comme les fréquences appariées avec les amplitudes qui sont pondérées par ces fonctions sont amplifiées ou atténuées, cela a pour effet d'appliquer des filtres très précis au signal. Dans cet exemple, la première table aura l'effet d'un filtre passe-bande, se transformant graduellement en réjecteur de bande sur la première moitié de la note, puis allant vers aucune modification des amplitudes dans la seconde moitié.

Voici un exemple complet de l'opcode `vpvoc`. Il utilise le fichier `vpvoc.csd` [exemples/vpvoc.csd].

## Exemple 1124.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o vpvoc.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
; analyze "fox.wav" with PVANAL first
iend = p4
ktime line 0, p3, iend
tablexseg p5, p3, p6 ;morph from table 1
asig vpvoc ktime, 1, "fox.pvx" ;to table 2
outs asig*3, asig*3

endin
</CsInstruments>
<CsScore>
f 1 0 512 9 .5 1 0
f 2 0 512 5 1 60 0.01 390 0.01 62 1

i 1 0 5 2.7 1 2
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pvoc*

## Crédits

Auteurs : Dan Ellis et Richard Karpen  
Seattle, WA USA  
1997

Nouveau dans la version 3.44

# vrandh

**vrandh** — Génère un vecteur de nombre aléatoires stockés dans une table, en maintenant les valeurs pendant une certaine durée.

## Description

Génère un vecteur de nombre aléatoires stockés dans une table, en maintenant les valeurs pendant une certaine durée. Génère une sorte de "bruit vectoriel à bande limitée".

## Syntaxe

```
vrandh ifn, krange, kcps, ielements [, idstoffset] [, iseed] \
      [, isize] [, ioffset]
```

## Initialisation

*ifn* - numéro de la table dans laquelle le signal vectoriel sera généré.

*ielements* - nombre de composantes du vecteur.

*idstoffset* - (facultatif, 0 par défaut) - décalage d'indexation pour la table de destination.

*iseed* (facultatif, 0.5 par défaut) - valeur de la graine pour la formule récursive des nombres pseudo-aléatoires. Une valeur entre 0 et +1 produira comme sortie initiale  $kamp * iseed$ . Avec une valeur négative, la réinitialisation de la graine sera ignorée. Avec une valeur supérieure à 1, la graine viendra de l'horloge système, ceci étant la meilleure option pour générer une séquence aléatoire différente à chaque exécution.

*isize* (facultatif, 0 par défaut) - s'il vaut zéro, un nombre sur 16 bit est généré. S'il est différent de zéro, un nombre aléatoire sur 31 bit est généré. Vaut 0 par défaut.

*ioffset* - (facultatif, 0 par défaut) - une valeur de base ajoutée au résultat aléatoire.

## Exécution

*krange* - intervalle des éléments aléatoires (entre  $-krange$  et  $krange$ ).

*kcps* - taux de génération des éléments en Hz.

Cet opcode est semblable à *randh*, mais il opère sur des vecteurs au lieu de valeurs scalaires.

Bien que l'argument *isize* soit nul par défaut, ce qui induit l'utilisation d'un générateur de nombres aléatoires sur 16 bit, il est recommandé d'utiliser l'algorithme sur 31 bit plus récent car il produit une séquence aléatoire avec une période plus longue (plus de nombre aléatoires avant que la séquence ne se répète).

Le vecteur de sortie est contenu dans *ifn* (qui doit avoir été allouée au préalable).

Tous ces opcodes sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des vecteurs comme *adsynt*, etc.

## Exemples

Voici un exemple de l'opcode *vrandh*. Il utilise le fichier *vrandh.csd* [exemples/vrandh.csd].

**Exemple 1125. Exemple de l'opcode vrandh.**

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc    -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vranh.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
;Example by Andres Cabrera

sr=44100
ksmps=128
nchnls=2

gitab ftgen 0, 0, 16, -7, 0, 128, 0

instr 1
  krange init p4
  kcps init p5
  ioffset init p6

  kav1 init 0
  kav2 init 0
  kcount init 0

  ;      table  krange kcps ielements idstoffset iseed isize ioffset
  vrandh  gitab, krange, kcps,      3,      3,      2,  0,  ioffset

  kfreq1 table 3, gitab
  kfreq2 table 4, gitab
  kfreq3 table 5, gitab

  ;Change the frequency of three oscillators according to the random values
  aosc1 oscili 4000, kfreq1, 1
  aosc2 oscili 2000, kfreq2, 1
  aosc3 oscili 4000, kfreq3, 1

  outs aosc1+aosc2, aosc3+aosc2
endin

</CsInstruments>
<CsScore>
f 1 0 1024 10 1
;      krange kcps ioffset
i 1 0 5 100 1 300
i 1 5 5 300 1 400
i 1 10 5 100 2 1000
i 1 15 5 400 4 1000
i 1 20 5 1000 8 2000
i 1 25 5 250 16 300
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*vrandi, randh*

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vrandi

vrandi — Génère une sorte de "bruit vectoriel à bande limitée".

## Description

Génère une sorte de "bruit vectoriel à bande limitée".

## Syntaxe

```
vrandi ifn, krange, kcps, ielements [, idstoffset] [, iseed] \
      [, isize] [, ioffset]
```

## Initialisation

*ifn* - numéro de la table dans laquelle le signal vectoriel sera généré.

*ielements* - nombre d'éléments à traiter.

*idstoffset* - (facultatif, 0 par défaut) - décalage d'indexation pour la table de destination.

*iseed* (facultatif, 0.5 par défaut) - valeur de la graine pour la formule récursive des nombres pseudo-aléatoires. Une valeur entre 0 et +1 produira comme sortie initiale  $kamp * iseed$ . Avec une valeur négative, la réinitialisation de la graine sera ignorée. Avec une valeur supérieure à 1, la graine viendra de l'horloge système, ceci étant la meilleure option pour générer une séquence aléatoire différente à chaque exécution.

*isize* (facultatif, 0 par défaut) - s'il vaut zéro, un nombre sur 16 bit est généré. S'il est différent de zéro, un nombre aléatoire sur 31 bit est généré. Vaut 0 par défaut.

*ioffset* - (facultatif, 0 par défaut) - une valeur de base ajoutée au résultat aléatoire.

## Exécution

*krange* - intervalle des éléments aléatoires (entre *-krange* et *krange*).

*kcps* - taux de génération des éléments en Hz.

Cet opcode est semblable à *randi*, mais il opère sur des vecteurs au lieu de valeurs scalaires.

Bien que l'argument *isize* soit nul par défaut, ce qui induit l'utilisation d'un générateur de nombres aléatoires sur 16 bit, il est recommandé d'utiliser l'algorithme sur 31 bit plus récent car il produit une séquence aléatoire avec une période plus longue (plus de nombre aléatoires avant que la séquence ne se répète).

Le vecteur de sortie est contenu dans *ifn* (qui doit avoir été allouée au préalable).

Tous ces opcodes sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des vecteurs comme *adsynt*, etc.

## Exemples

Voici un exemple de l'opcode *vrandi*. Il utilise le fichier *vrandi.csd* [examples/vrandi.csd].

## Exemple 1126. Exemple de l'opcode vrandi.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc    -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o vrandi.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

;Example by Andres Cabrera

gitab ftgen 0, 0, 16, -7, 0, 128, 0

instr 1
  krange init p4
  kcps init p5
  ioffset init p6
  ;      table  krange kcps  ielements  idstoffset  iseed  isize ioffset
  vrandi gitab, krange, kcps,      3,          3,          2,  1,  ioffset

  kfreq1 table 3, gitab
  kfreq2 table 4, gitab
  kfreq3 table 5, gitab

  ;Change the frequency of three oscillators according to the random values
  aosc1 oscili 4000, kfreq1, 1
  aosc2 oscili 2000, kfreq2, 1
  aosc3 oscili 4000, kfreq3, 1

  outs aosc1+aosc2, aosc3+aosc2
endin

</CsInstruments>
<CsScore>

f 1 0 2048 10 1

;      krange kcps  ioffset
i 1 0 5 100 1 300
i 1 5 5 5 1 400
i 1 10 5 100 2 1000
i 1 15 5 400 4 1000
i 1 20 5 1000 8 2000
i 1 20 5 300 32 350

e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*vrandh*, *randi*

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)



# vsubv

vsubv — Soustraction entre deux signaux vectoriels de contrôle.

## Description

Soustraction entre deux signaux vectoriels de contrôle.

## Syntaxe

```
vsubv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
```

## Initialisation

*ifn1* - numéro de la table hébergeant le premier vecteur à traiter.

*ifn2* - numéro de la table hébergeant le second vecteur à traiter.

## Exécution

*kelements* - nombre de composantes des deux vecteurs.

*kdstoffset* - décalage d'indexation pour la table de destination *ifn1* (vaut 0 par défaut).

*ksrcoffset* - décalage d'indexation pour la table source *ifn2* (vaut 0 par défaut).

*kverbose* - Indique si les avertissements sont affichés (vaut 0 par défaut).

*vsubv* soustrait deux signaux vectoriels de contrôle, chaque composante de *ifn2* étant soustraite de l'élément correspondant de *ifn1*. Chaque signal vectoriel est hébergé dans une table (*ifn1* et *ifn2*). Le nombre de composantes de chaque vecteur doit être identique.

Le résultat est un nouveau signal vectoriel de contrôle qui écrase les anciennes valeurs de *ifn1*. Si l'on veut garder l'ancien vecteur *ifn1*, il faut utiliser l'opcode *vcopy\_i* pour le copier dans une autre table. On peut utiliser *kdstoffset* et *ksrcoffset* pour spécifier des vecteurs à n'importe quelle position dans les tables.

Des valeurs négatives pour *kdstoffset* et *ksrcoffset* sont acceptables. Si *kdstoffset* est négatif, la partie du vecteur hors-limites est ignorée. Si *ksrcoffset* est négatif, les éléments hors-limites seront supposés valoir 0 (c'est-à-dire que les éléments de destination ne seront pas changés). Si des éléments pour le vecteur de destination sont au-delà de la taille de la table (point de garde inclus), ces éléments sont ignorés (les éléments ne sont pas repliés autour des tables). Si des éléments pour le vecteur source sont au-delà de la longueur de la table, ces éléments sont supposés valoir 0 (le vecteur de destination ne sera pas changé pour ces éléments).

Si l'argument facultatif *kverbose* est différent de 0, l'opcode affichera des messages d'avertissement à chaque passe-k si les longueurs de table sont dépassées.



### Avertissement

L'utilisation de la même table comme source et comme destination dans les versions antérieures à la 5.04 peut induire un comportement imprévu. A utiliser avec précaution.

Cet opcode travaille au taux-k (cela signifie qu'à chaque passe-k les vecteurs sont soustraits). Il y a une version de taux-i de cet opcode appelée *vsubv\_i*.



## Note

Prière de noter que l'argument *elements* a changé dans la version 5.03 du taux-i au taux-k. Cela change le comportement de l'opcode dans le cas inhabituel où la variable de taux-i *ielements* est modifiée à l'intérieur de l'instrument, par exemple dans :

```
instr 1
ielements = 10
vadd 1, 1, ielements
ielements = 20
vadd 2, 1, ielements
turnoff
endin
```

Tous ces opérateurs (*vaddv*, *vsubv*, *vmultv*, *vdivv*, *vpowv*, *vexpv*, *vcopy* et *vmap*) sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Exemples

Voici un exemple de l'opcode *vsubv*. Il utilise le fichier *vsubv.csd* [examples/vsubv.csd].

### Exemple 1127. Exemple de l'opcode *vsubv*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o cigoto.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr=44100
ksmps=128
nchnls=2

instr 1
ifn1 = p4
ifn2 = p5
ielements = p6
idstoffset = p7
isrcoffset = p8
kval init 25
vsubv ifn1, ifn2, ielements, idstoffset, isrcoffset, 1
endin

instr 2 ;Printtable
itable = p4
isize = ftlen(itable)
kcount init 0
kval table kcount, itable
printk2 kval
```

```

if (kcount == isize) then
    turnoff
endif

kcount = kcount + 1
endin

</CsInstruments>
<CsScore>

f 1 0 16 -7 1 15 16

f 2 0 16 -7 1 15 2

i2 0.0 0.2 1
i2 0.2 0.2 2
i1 0.4 0.01 1 2 5 3 8
i2 0.8 0.2 1
i1 1.0 0.01 1 2 5 10 -2
i2 1.2 0.2 1
i1 1.4 0.01 1 2 8 14 0
i2 1.6 0.2 1
i1 1.8 0.01 1 2 8 0 14
i2 2.0 0.2 1
i1 2.2 0.002 1 1 8 5 2
i2 2.4 0.2 1
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vsubv\_i

`vsubv_i` — Soustraction entre deux signaux vectoriels de contrôle à l'initialisation.

## Description

Soustraction entre deux signaux vectoriels de contrôle à l'initialisation.

## Syntaxe

```
vsubv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
```

## Initialisation

*ifn1* - numéro de la table hébergeant le premier vecteur à traiter.

*ifn2* - numéro de la table hébergeant le second vecteur à traiter.

*ielements* - nombre de composantes des deux vecteurs.

*idstoffset* - décalage d'indexation pour la table de destination *ifn1* (vaut 0 par défaut).

*isrcoffset* - décalage d'indexation pour la table source *ifn2* (vaut 0 par défaut).

## Exécution

`vsubv_i` soustrait deux signaux vectoriels de contrôle, chaque composante de *ifn2* étant soustraite de l'élément correspondant de *ifn1*. Chaque signal vectoriel est hébergé dans une table (*ifn1* et *ifn2*). Le nombre de composantes de chaque vecteur doit être identique.

Le résultat est un nouveau signal vectoriel de contrôle qui écrase les anciennes valeurs de *ifn1*. Si l'on veut garder l'ancien vecteur *ifn1*, il faut utiliser l'opcode `vcopy_i` pour le copier dans une autre table. On peut utiliser *idstoffset* et *isrcoffset* pour spécifier des vecteurs à n'importe quelle position dans les tables.

Des valeurs négatives pour *idstoffset* et *isrcoffset* sont acceptables. Si *idstoffset* est négatif, la partie du vecteur hors-limites est ignorée. Si *isrcoffset* est négatif, les éléments hors-limites seront supposés valoir 0 (c'est-à-dire que les éléments de destination ne seront pas changés). Si des éléments pour le vecteur de destination sont au-delà de la taille de la table (point de garde inclus), ces éléments sont ignorés (les éléments ne sont pas repliés autour des tables). Si des éléments pour le vecteur source sont au-delà de la longueur de la table, ces éléments sont supposés valoir 0 (le vecteur de destination ne sera pas changé pour ces éléments).



### Avertissement

L'utilisation de la même table comme source et comme destination dans les versions antérieures à la 5.04 peut induire un comportement imprévu. A utiliser avec précaution.

Cet opcode travaille au taux-i. Il y a une version de taux-k de cet opcode appelée `vsubv`.

Tous ces opérateurs (`vaddv_i`, `vsubv_i`, `vmultv_i`, `vdivv_i`, `vpowv_i`, `vexpv_i`, `vcopy` et `vmap`) sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que `vcella`, `adsynt`, `adsynt2`, etc.

## Crédits

Ecrit par Gabriel Maldonado. Arguments facultatifs ajoutés par Andrés Cabrera et Istvan Varga.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vtable1k

vtable1k — Lit un vecteur (plusieurs scalaires simultanément) depuis une table.

## Description

Cet opcode lit des vecteurs depuis des tables au taux-k.

## Syntaxe

```
vtable1k kfn, kout1 [, kout2, kout3, .... , koutN ]
```

## Exécution

*kfn* - numéro de la table.

*kout1...koutN* - composantes du vecteur de sortie.

*vtable1k* est une version réduite de *vtablek*. Il ne permet d'accéder qu'au premier vecteur (c'est équivalent à *vtablek* avec *kndx* = 0, mais un peu plus rapide). Il est utile pour convertir facilement et rapidement un ensemble de valeurs stockées dans une table en un ensemble de variables de taux-k à utiliser dans des opcodes normaux, au lieu d'utiliser des opcodes *table* individuels pour chaque valeur.



### Note

*vtable1k* est un opcode inhabituel car il produit sa sortie dans des arguments placés à droite de l'opcode.

## Exemples

Voici un exemple de l'opcode *vtable1k*. Il utilise le fichier *vtable1k.csd* [examples/vtable1k.csd].

### Exemple 1128. Exemple de l'opcode vtable1k.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc    -d      ;;RT audio I/O
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 100
nchnls = 2

giElem init 13
giOutTab ftgen 1,0,128, 2, 0
giFreqTab ftgen 2,0,128,-7, 1,giElem, giElem+1
giSine ftgen 3,0,256,10, 1

FLpanel "This Panel contains a Slider Bank",500,400
FLslidBnk "mod1@mod2@mod3@amp@freq1@freq2@freq3@freqPo", giElem, giOutTab, 360, 600, 100, 10
```

```

FLpanel_end

FLrun

instr 1

kout1 init 0
kout2 init 0
kout3 init 0
kout4 init 0
kout5 init 0
kout6 init 0
kout7 init 0
kout8 init 0

vtablelk giOutTab, kout1 , kout2, kout3, kout4, kout5 , kout6, kout7, kout8
kmodindex1= 2 * db(kout1 * 80 )
kmodindex2= 2 * db(kout2 * 80 )
kmodindex3= 2 * db(kout3 * 80 )
kamp = 50 * db(kout4 * 70 )
kfreq1 = 1.1 * octave(kout5 * 10)
kfreq2 = 1.1 * octave(kout6 * 10)
kfreq3 = 1.1 * octave(kout7 * 10)
kfreq4 = 30 * octave(kout8 * 8)

amod1 oscili kmodindex1, kfreq1, giSine
amod2 oscili kmodindex2, kfreq2, giSine
amod3 oscili kmodindex3, kfreq3, giSine
aout oscili kamp, kfreq4+amod1+amod2+amod3, giSine

outs aout, aout
endin

</CsInstruments>
<CsScore>

i1 0 3600
f0 3600

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*vtablek*

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5.06

# vtablei

vtablei — Lit des vecteurs (à partir de tables, ou tableaux de vecteurs).

## Description

Cet opcode lit des vecteurs à partir de tables.

## Syntaxe

```
vtablei  indx, ifn, interp, ixmode, iout1 [, iout2, iout3, .... , ioutN ]
```

## Initialisation

*indx* - Index dans la ftable, soit un nombre positif inférieur à la longueur de la table (*ixmode* = 0) soit un nombre compris entre 0 et 1 (*ixmode* != 0).

*ifn* - numéro de la table.

*iout1...ioutN* - composantes du vecteur de sortie.

*ixmode* - mode d'indexation. Vaut 0 par défaut.

== 0 l'index est traité comme une position brute dans la table,

== 1 l'index est normalisé (entre 0 et 1).

*interp* - bascule entre sortie interpolée ou non-interpolée. 0 -> pas d'interpolation, différent de zéro -> interpolation activée.

## Exécution

Cet opcode est utile dans tous les cas où il faut accéder à des ensembles de valeurs associés à des indices uniques (par exemple, des échantillons multi-canaux, des trames de bin de TFCT, des formants spectraux, des partitions basées sur des p-champs, etc). Le nombre de composantes de chaque vecteur (longueur du vecteur) est déterminé par le nombre d'arguments facultatifs à droite (*iout1*, *iout2*, *iout3*, ..., *ioutN*).

La famille d'opcodes *vtable* (vector table) permet à l'utilisateur de basculer entre sortie interpolée ou non-interpolée au moyen de l'argument *interp*.

Noter qu'aucun mode de repliement ou de limitation d'indexation n'est implémenté. Si l'index tente d'accéder à une zone non allouée par la table, il est probable que Csound plante. Cependant on peut facilement éviter cet écueil en utilisant des opcodes de repliement ou de limitation appliqués à l'index avant l'utilisation de *vtablei*, afin de corriger d'éventuelles valeurs hors-limites.



### Note

Noter que les arguments de sortie de *vtablei* sont placés à droite du nom de l'opcode, contrairement à l'habitude (ce style est aussi utilisé dans d'autres opcodes utilisant des listes indéfinies d'arguments de sortie comme *fin* ou *trigseq*).



## Exemples

Voici un exemple de l'opcode `vtablei`. Il utilise le fichier `vtablei.csd` [examples/vtablei.csd]

### Exemple 1129. Exemple de l'opcode `vtablei`.

```
<CsoundSynthesizer>
<CsOptions>
-odac -B441 -b441
</CsOptions>
<CsInstruments>

sr      =      44100
kr      =      100
ksmps   =      441
nchnls  =      2

gindx   init 0

        instr    1
kindex   init 0
ktrig    metro 0.5
if ktrig = 0 goto noevent
event "i", 2, 0, 0.5, kindex
kindex = kindex + 1
noevent:

        endin

        instr 2
iout1    init 0
iout2    init 0
iout3    init 0
iout4    init 0
indx = p4
vtablei  indx, 1, 1, 0, iout1,iout2, iout3, iout4
print    iout1, iout2, iout3, iout4
turnoff
        endin

</CsInstruments>
<CsScore>
f 1 0 32 10 1
i 1 0 20

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

`vtablea`, `vtablek`, `vtabi`, `vtablewi`, `vtabwi`,

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vtablek

**vtablek** — Lit des vecteurs (à partir de tables, ou tableaux de vecteurs).

## Description

Cet opcode lit des vecteurs à partir de tables au taux-k.

## Syntaxe

```
vtablek kndx, kfn, kinterp, ixmode, kout1 [, kout2, kout3, .... , koutN ]
```

## Initialisation

*ixmode* - mode d'indexation. Vaut 0 par défaut.

== 0 l'index est traité comme une position brute dans la table,

== 1 l'index est normalisé (entre 0 et 1).

## Exécution

*kndx* - Index dans la ftable, soit un nombre positif inférieur à la longueur de la table (*ixmode* = 0) soit un nombre compris entre 0 et 1 (*ixmode* != 0).

*kfn* - numéro de la table.

*kinterp* - bascule entre sortie interpolée ou non-interpolée. 0 -> pas d'interpolation, différent de zéro -> interpolation activée.

*kout1...koutN* - composantes du vecteur de sortie.

Cet opcode est utile dans tous les cas où il faut accéder à des ensembles de valeurs associés à des indices uniques (par exemple, des échantillons multi-canaux, des trames de bin de TFCT, des formants spectraux, des partitions basées sur des p-champs, etc). Le nombre de composantes de chaque vecteur (longueur du vecteur) est déterminé par le nombre d'arguments facultatifs à droite (*kout1*, *kout2*, *kout3*, ..., *koutN*).

**vtablek** permet à l'utilisateur de basculer entre sortie interpolée ou non-interpolée au taux-k au moyen de l'argument *kinterp*.

**vtablek** permet aussi de changer le numéro de table au taux-k (mais ceci n'est possible que si les trames de vecteur de chaque table utilisée ont le même nombre d'éléments, sinon il peut y avoir des résultats imprévisibles), ainsi que de choisir le style d'indexation (brute ou normalisée, voir aussi l'argument *ixmode* de l'opcode *table*).

Noter qu'aucun mode de repliement ou de limitation d'indexation n'est implémenté. Si l'index tente d'accéder à une zone non allouée par la table, il est probable que Csound plante. Cependant on peut facilement éviter cet écueil en utilisant des opcodes de repliement ou de limitation appliqués à l'index avant l'utilisation de **vtablek**, afin de corriger d'éventuelles valeurs hors-limites.



### Note

Noter que les arguments de sortie de **vtablek** sont placés à droite du nom de l'opcode, contrairement à l'habitude (ce style est aussi utilisé dans d'autres opcodes utilisant des listes indéfinies d'arguments de sortie comme *fin* ou *trigseq*).

## Exemples

Voici un exemple de l'opcode `vtablek`. Il utilise le fichier `vtablek.csd` [examples/vtablek.csd].

### Exemple 1130. Exemple de l'opcode `vtablek`.

```
<CsoundSynthesizer>
<CsOptions>
-odac -B441 -b441
</CsOptions>
<CsInstruments>

sr      =      44100
kr      =      100
ksmps   =      441
nchnls  =      2

gkindx  init  -1

        instr  1
kindex  init  0
ktrig   metro 0.5
if ktrig = 0 goto noevent
gkindx  = gkindx + 1
noevent:

        endin

        instr  2
kout1    init  0
kout2    init  0
kout3    init  0
kout4    init  0
vtablek  gkindx, 1, 1, 0, kout1,kout2, kout3, kout4
printk2  kout1
printk2  kout2
printk2  kout3
printk2  kout4
        endin

</CsInstruments>
<CsScore>
f 1 0 32 10 1
i 1 0 20
i 2 0 20
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

`vtablea`, `vtablei`, `vtabk`, `vtablewk`, `vtabwk`,

## Crédits

Ecrit par Gabriel Maldonado.

Exemple écrit par Andrés Cabrera.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vtablea

vtablea — Lit des vecteurs (à partir de tables, ou tableaux de vecteurs).

## Description

Cet opcode lit des vecteurs à partir de tables au taux-a.

## Syntaxe

```
vtablea andx, kfn, kinterp, ixmode, aout1 [, aout2, aout3, .... , aoutN ]
```

## Initialisation

*ixmode* - mode d'indexation. Vaut 0 par défaut.

== 0 l'index est traité comme une position brute dans la table,

== 1 l'index est normalisé (entre 0 et 1).

## Exécution

*andx* - Index dans la ftable, soit un nombre positif inférieur à la longueur de la table (*ixmode* = 0) soit un nombre compris entre 0 et 1 (*ixmode* != 0).

*kfn* - numéro de la table.

*kinterp* - bascule entre sortie interpolée ou non-interpolée. 0 -> pas d'interpolation, différent de zéro -> interpolation activée.

*aout1...aoutN* - composantes du vecteur de sortie.

Cet opcode est utile dans tous les cas où il faut accéder à des ensembles de valeurs associés à des indices uniques (par exemple, des échantillons multi-canaux, des trames de bin de TFCT, des formants spectraux, des partitions basées sur des p-champs, etc). Le nombre de composantes de chaque vecteur (longueur du vecteur) est déterminé par le nombre d'arguments facultatifs à droite (*aout1*, *aout2*, *aout3*, ..., *aoutN*).

*vtablea* permet à l'utilisateur de basculer entre sortie interpolée ou non-interpolée au taux-k au moyen de l'argument *kinterp*.

*vtablea* permet aussi de changer le numéro de table au taux-k (mais ceci n'est possible que si les trames de vecteur de chaque table utilisée ont le même nombre d'éléments, sinon il peut y avoir des résultats imprévisibles), ainsi que de choisir le style d'indexation (brute ou normalisée, voir aussi l'argument *ixmode* de l'opcode *table*).

Noter qu'aucun mode de repliement ou de limitation d'indexation n'est implémenté. Si l'index tente d'accéder à une zone non allouée par la table, il est probable que Csound plante. Cependant on peut facilement éviter cet écueil en utilisant des opcodes de repliement ou de limitation appliqués à l'index avant l'utilisation de *vtablea*, afin de corriger d'éventuelles valeurs hors-limites.



### Note

Noter que les arguments de sortie de *vtablea* sont placés à droite du nom de l'opcode, contrairement à l'habitude (ce style est aussi utilisé dans d'autres opcodes utilisant des listes indéfinies d'arguments de sortie comme *fin* ou *trigseq*).

## Voir aussi

*vtablek, vtablei, vtaba, vtablewa, vtabwa,*

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# **vtablewi**

vtablewi — Ecrit des vecteurs (dans des tables, ou tableaux de vecteurs).

## **Description**

Cet opcode écrit des vecteurs dans des tables à l'initialisation.

## **Syntaxe**

```
vtablewi  indx, ifn, ixmode, inarg1 [ , inarg2, inarg3 , .... , inargN ]
```

## **Initialisation**

*indx* - Index dans la ftable, soit un nombre positif inférieur à la longueur de la table (*ixmode* = 0) soit un nombre compris entre 0 et 1 (*ixmode* != 0).

*ifn* - numéro de la table.

*ixmode* - mode d'indexation. La valeur par défaut est 0.

== 0, l'index est traité comme une position brute dans la table,

== 1, l'index est normalisé (entre 0 et 1).

*inarg1...inargN* - Composantes du vecteur d'entrée.

## **Exécution**

Cet opcode est utile dans tous les cas où il faut écrire des ensembles de valeurs associés à des indices uniques (par exemple, des échantillons multi-canaux, des trames de bin de TFCT, des formants spectraux, des partitions basées sur des p-champs, etc). Le nombre de composantes de chaque vecteur (longueur du vecteur) est déterminé par le nombre d'arguments facultatifs à droite (*inarg1, inarg2, inarg3, ..., inargN*).

Noter qu'aucun mode de repliement ou de limitation d'indexation n'est implémenté. Si l'index tente d'accéder à une zone non allouée par la table, il est probable que Csound plante. Cependant on peut facilement éviter cet écueil en utilisant des opcodes de repliement ou de limitation appliqués à l'index avant l'utilisation de *vtablewi*, afin de corriger d'éventuelles valeurs hors-limites.

## **Crédits**

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vtablewk

vtablewk — Écrit des vecteurs (dans des tables, ou tableaux de vecteurs).

## Description

Cet opcode écrit des vecteurs dans des tables au taux-k.

## Syntaxe

```
vtablewk kndx, kfn, ixmode, kinarg1 [ , kinarg2, kinarg3 , .... , kinargN ]
```

## Initialisation

*ixmode* - mode d'indexation. La valeur par défaut est 0.

== 0, l'index est traité comme une position brute dans la table,

== 1, l'index est normalisé (entre 0 et 1).

## Exécution

*kndx* - Index dans la ftable, soit un nombre positif inférieur à la longueur de la table (*ixmode* = 0) soit un nombre compris entre 0 et 1 (*ixmode* != 0).

*kfn* - numéro de la table.

*kinarg1...kinargN* - Composantes du vecteur d'entrée.

Cet opcode est utile dans tous les cas où il faut écrire des ensembles de valeurs associés à des indices uniques (par exemple, des échantillons multi-canaux, des trames de bin de TFCT, des formants spectraux, des partitions basées sur des p-champs, etc). Le nombre de composantes de chaque vecteur (longueur du vecteur) est déterminé par le nombre d'arguments facultatifs à droite (*kinarg1*, *kinarg2*, *kinarg3*, ..., *kinargN*).

**vtablewk** permet aussi de changer le numéro de table au taux-k (mais ceci n'est possible que si les trames de vecteur de chaque table utilisée ont le même nombre d'éléments, sinon il peut y avoir des résultats imprévisibles), ainsi que de choisir le style d'indexation (brute ou normalisée, voir aussi l'argument *ixmode* de l'opcode *table*).

Noter qu'aucun mode de repliement ou de limitation d'indexation n'est implémenté. Si l'index tente d'accéder à une zone non allouée par la table, il est probable que Csound plante. Cependant on peut facilement éviter cet écueil en utilisant des opcodes de repliement ou de limitation appliqués à l'index avant l'utilisation de *vtablewk*, afin de corriger d'éventuelles valeurs hors-limites.

## Exemples

Voici un exemple de l'opcode vtablewk. Il utilise le fichier *vtablewk.csd* [examples/vtablewk.csd].

**Exemple 1131. Exemple de l'opcode vtablewk.**

```

<CsoundSynthesizer>
<CsOptions>
;-ovtablewa.wav -W -b441 -B441
-odac -b441 -B441
</CsOptions>
<CsInstruments>

sr=44100
kr=441
ksmps=100
nchnls=2

instr 1
ilen = ftlen(1)

knew1 oscil 10000, 440, 3
knew2 oscil 15000, 440, 3, 0.5
kindex phasor 0.3
asig oscil 1, sr/ilen, 1
vtablewk kindex*ilen, 1, 0, knew1, knew2
out asig,asig
endin

</CsInstruments>
<CsScore>
f1 0 262144 -1 "beats.wav" 0 4 0
f2 0 262144 2 0
f3 0 1024 10 1

i1 0 10
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Ecrit par Gabriel Maldonado.

Exemple écrit par Andrés Cabrera.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)



# vtablewa

vtablewa — Ecrit des vecteurs (dans des tables, ou tableaux de vecteurs).

## Description

Cet opcode écrit des vecteurs dans des tables au taux-a.

## Syntaxe

```
vtablewa andx, kfn, ixmode, ainarg1 [, ainarg2, ainarg3 , .... , ainargN ]
```

## Initialisation

*ixmode* - mode d'indexation. La valeur par défaut est 0.

== 0, l'index est traité comme une position brute dans la table,

== 1, l'index est normalisé (entre 0 et 1).

## Exécution

*andx* - Index dans la ftable, soit un nombre positif inférieur à la longueur de la table (*ixmode* = 0) soit un nombre compris entre 0 et 1 (*ixmode* != 0).

*kfn* - numéro de la table.

*ainarg1...ainargN* - Composantes du vecteur d'entrée.

Cet opcode est utile dans tous les cas où il faut écrire des ensembles de valeurs associés à des indices uniques (par exemple, des échantillons multi-canaux, des trames de bin de TFCT, des formants spectraux, des partitions basées sur des p-champs, etc). Le nombre de composantes de chaque vecteur (longueur du vecteur) est déterminé par le nombre d'arguments facultatifs à droite (*ainarg1*, *ainarg2*, *ainarg3*, ..., *ainargN*).

*vtablewa* permet aussi de changer le numéro de table au taux-k (mais ceci n'est possible que si les trames de vecteur de chaque table utilisée ont le même nombre d'éléments, sinon il peut y avoir des résultats imprévisibles), ainsi que de choisir le style d'indexation (brute ou normalisée, voir aussi l'argument *ixmode* de l'opcode *table*).

Noter qu'aucun mode de repliement ou de limitation d'indexation n'est implémenté. Si l'index tente d'accéder à une zone non allouée par la table, il est probable que Csound plante. Cependant on peut facilement éviter cet écueil en utilisant des opcodes de repliement ou de limitation appliqués à l'index avant l'utilisation de *vtablewa*, afin de corriger d'éventuelles valeurs hors-limites.

## Exemples

Voici un exemple de l'opcode *vtablewa*. Il utilise le fichier *vtablewa.csd* [examples/vtablewa.csd].

**Exemple 1132. Exemple de l'opcode vtablewa.**

```
<CsoundSynthesizer>
<CsOptions>
-odac -b441 -B441
</CsOptions>
<CsInstruments>

sr=44100
kr=4410
ksmps=10
nchnls=2
0dbfs = 1

instr 1
vcopy 2, 1, 262144
ar random 0, 1
vtablewa ar,2,1,ar
out ar,ar
endin

</CsInstruments>
<CsScore>
f1 0 262144 -1 "beats.wav" 0 4 0
f2 0 262144 2 0

i1 0 4

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Ecrit par Gabriel Maldonado.

Exemple écrit par Andrés Cabrera.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vtabi

vtabi — Lit des vecteurs (à partir de tables, ou tableaux de vecteurs).

## Description

Cet opcode lit des vecteurs à partir de tables.

## Syntaxe

```
vtabi  indx, ifn, iout1 [, iout2, iout3, .... , ioutN ]
```

## Initialisation

*indx* - Index dans la ftable, un nombre positif inférieur à la longueur de la table.

*ifn* - numéro de la table.

*iout1...ioutN* - Composantes du vecteur de sortie.

## Exécution

Cet opcode est utile dans tous les cas où il faut accéder à des ensembles de valeurs associés à des indices uniques (par exemple, des échantillons multi-canaux, des trames de bin de TFCT, des formants spectraux, des partitions basées sur des p-champs, etc). Le nombre de composantes de chaque vecteur (longueur du vecteur) est déterminé par le nombre d'arguments facultatifs à droite (*iout1, iout2, iout3, ..., ioutN*).

Noter qu'aucun mode de repliement ou de limitation d'indexation n'est implémenté. Si l'index tente d'accéder à une zone non allouée par la table, il est probable que Csound plante. Cependant on peut facilement éviter cet écueil en utilisant des opcodes de repliement ou de limitation appliqués à l'index avant l'utilisation de *vtabi*, afin de corriger d'éventuelles valeurs hors-limites.

La famille **vtab** est semblable à **vtable**, mais elle est bien plus rapide car il n'y a pas d'interpolation, le numéro de table ne peut pas être changé après l'initialisation et seul l'indexation brute est supportée.



### Note

Noter que les arguments de sortie de *vtabi* sont placés à droite du nom de l'opcode, contrairement à l'habitude (ce style est aussi utilisé dans d'autres opcodes utilisant des listes indéfinies d'arguments de sortie comme *fin* ou *trigseq*).

## Exemples

Pour un exemple d'utilisation de l'opcode *vtabi*, voir *vtablei*.

## Voir aussi

*vtabk, vtaba, vtablei, vtablewi, vtabwi,*

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vtabk

**vtabk** — Lit des vecteurs (à partir de tables, ou tableaux de vecteurs).

## Description

Cet opcode lit des vecteurs à partir de tables au taux-k.

## Syntaxe

```
vtabk kndx, ifn, kout1 [, kout2, kout3, .... , koutN ]
```

## Initialisation

*ifn* - numéro de la table.

## Exécution

*kndx* - Index dans la ftable, un nombre positif inférieur à la longueur de la table.

*kout1...koutN* - Composantes du vecteur de sortie.

Cet opcode est utile dans tous les cas où il faut accéder à des ensembles de valeurs associés à des indices uniques (par exemple, des échantillons multi-canaux, des trames de bin de TFCT, des formants spectraux, des partitions basées sur des p-champs, etc). Le nombre de composantes de chaque vecteur (longueur du vecteur) est déterminé par le nombre d'arguments facultatifs à droite (*kout1*, *kout2*, *kout3*, ..., *koutN*).

Noter qu'aucun mode de repliement ou de limitation d'indexation n'est implémenté. Si l'index tente d'accéder à une zone non allouée par la table, il est probable que Csound plante. Cependant on peut facilement éviter cet écueil en utilisant des opcodes de repliement ou de limitation appliqués à l'index avant l'utilisation de **vtabk**, afin de corriger d'éventuelles valeurs hors-limites.

La famille **vtab** est semblable à **vtable**, mais elle est bien plus rapide car il n'y a pas d'interpolation, le numéro de table ne peut pas être changé après l'initialisation et seul l'indexation brute est supportée.



### Note

Noter que les arguments de sortie de **vtabk** sont placés à droite du nom de l'opcode, contrairement à l'habitude (ce style est aussi utilisé dans d'autres opcodes utilisant des listes indéfinies d'arguments de sortie comme *fin* ou *trigseq*).

## Exemples

Pour un exemple d'utilisation de l'opcode **vtabk**, voir *vtablek*.

## Voir aussi

*vtabi*, *vtaba*, *vtablek*, *vtablewk*, *vtabwk*,

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vtaba

*vtaba* — Lit des vecteurs (à partir de tables, ou tableaux de vecteurs).

## Description

Cet opcode lit des vecteurs à partir de tables au taux-a.

## Syntaxe

```
vtaba andx, ifn, aout1 [, aout2, aout3, .... , aoutN ]
```

## Initialisation

*ifn* - numéro de la table.

## Exécution

*andx* - Index dans la ftable, un nombre positif inférieur à la longueur de la table.

*aout1...aoutN* - Composantes du vecteur de sortie.

Cet opcode est utile dans tous les cas où il faut accéder à des ensembles de valeurs associés à des indices uniques (par exemple, des échantillons multi-canaux, des trames de bin de TFCT, des formants spectraux, des partitions basées sur des p-champs, etc). Le nombre de composantes de chaque vecteur (longueur du vecteur) est déterminé par le nombre d'arguments facultatifs à droite (*aout1*, *aout2*, *aout3*, ..., *aoutN*).

Noter qu'aucun mode de repliement ou de limitation d'indexation n'est implémenté. Si l'index tente d'accéder à une zone non allouée par la table, il est probable que Csound plante. Cependant on peut facilement éviter cet écueil en utilisant des opcodes de repliement ou de limitation appliqués à l'index avant l'utilisation de *vtaba*, afin de corriger d'éventuelles valeurs hors-limites.

La famille **vtab** est semblable à **vtable**, mais elle est bien plus rapide car il n'y a pas d'interpolation, le numéro de table ne peut pas être changé après l'initialisation et seul l'indexation brute est supportée.



### Note

Noter que les arguments de sortie de *vtaba* sont placés à droite du nom de l'opcode, contrairement à l'habitude (ce style est aussi utilisé dans d'autres opcodes utilisant des listes indéfinies d'arguments de sortie comme *fin* ou *trigseq*).

## Exemples

L'utilisation de *vtaba* est semblable à celle de *vtablek*.

## Voir aussi

*vtabk*, *vtabi*, *vtablea*, *vtablewa*, *vtabwa*,

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)



# **vtabwi**

vtabwi — Ecrit des vecteurs (dans des tables, ou tableaux de vecteurs).

## **Description**

Cet opcode écrit des vecteurs dans des tables à l'initialisation.

## **Syntaxe**

```
vtabwi  indx, ifn, inarg1 [ , inarg2, inarg3 , .... , inargN ]
```

## **Initialisation**

*indx* - Index dans la ftable, un nombre positif inférieur à la longueur de la table.

*ifn* - numéro de la table.

*inarg1...inargN* - Composantes du vecteur d'entrée.

## **Exécution**

Cet opcode est utile dans tous les cas où il faut écrire des ensembles de valeurs associés à des indices uniques (par exemple, des échantillons multi-canaux, des trames de bin de TFCT, des formants spectraux, des partitions basées sur des p-champs, etc). Le nombre de composantes de chaque vecteur (longueur du vecteur) est déterminé par le nombre d'arguments facultatifs à droite (*inarg1*, *inarg2*, *inarg3*, ..., *inargN*).

Noter qu'aucun mode de repliement ou de limitation d'indexation n'est implémenté. Si l'index tente d'accéder à une zone non allouée par la table, il est probable que Csound plante. Cependant on peut facilement éviter cet écueil en utilisant des opcodes de repliement ou de limitation appliqués à l'index avant l'utilisation de *vtabwi*, afin de corriger d'éventuelles valeurs hors-limites.

## **Crédits**

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# **vtabwk**

vtabwk — Écrit des vecteurs (dans des tables, ou tableaux de vecteurs).

## **Description**

Cet opcode écrit des vecteurs dans des tables au taux-k.

## **Syntaxe**

```
vtabwk kndx, ifn, kinarg1 [ , kinarg2, kinarg3 , .... , kinargN ]
```

## **Initialisation**

*ifn* - numéro de la table.

## **Exécution**

*kndx* - Index dans la ftable, un nombre positif inférieur à la longueur de la table.

*kinarg1...kinargN* - Composantes du vecteur d'entrée.

Cet opcode est utile dans tous les cas où il faut écrire des ensembles de valeurs associés à des indices uniques (par exemple, des échantillons multi-canaux, des trames de bin de TFCT, des formants spectraux, des partitions basées sur des p-champs, etc). Le nombre de composantes de chaque vecteur (longueur du vecteur) est déterminé par le nombre d'arguments facultatifs à droite (*kinarg1*, *kinarg2*, *kinarg3*, ..., *kinargN*).

Noter qu'aucun mode de repliement ou de limitation d'indexation n'est implémenté. Si l'index tente d'accéder à une zone non allouée par la table, il est probable que Csound plante. Cependant on peut facilement éviter cet écueil en utilisant des opcodes de repliement ou de limitation appliqués à l'index avant l'utilisation de *vtabwk*, afin de corriger d'éventuelles valeurs hors-limites.

## **Crédits**

Écrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vtabwa

vtabwa — Écrit des vecteurs (dans des tables, ou tableaux de vecteurs).

## Description

Cet opcode écrit des vecteurs dans des tables au taux-a.

## Syntaxe

```
vtabwa andx, ifn, ainarg1 [ , ainarg2, ainarg3 , .... , ainargN ]
```

## Initialisation

*ifn* - numéro de la table.

## Exécution

*andx* - Index dans la ftable, un nombre positif inférieur à la longueur de la table.

*ainarg1...ainargN* - Composantes du vecteur d'entrée.

Cet opcode est utile dans tous les cas où il faut écrire des ensembles de valeurs associés à des indices uniques (par exemple, des échantillons multi-canaux, des trames de bin de TFCT, des formants spectraux, des partitions basées sur des p-champs, etc). Le nombre de composantes de chaque vecteur (longueur du vecteur) est déterminé par le nombre d'arguments facultatifs à droite (*ainarg1*, *ainarg2*, *ainarg3*, ..., *ainargN*).

Noter qu'aucun mode de repliement ou de limitation d'indexation n'est implémenté. Si l'index tente d'accéder à une zone non allouée par la table, il est probable que Csound plante. Cependant on peut facilement éviter cet écueil en utilisant des opcodes de repliement ou de limitation appliqués à l'index avant l'utilisation de *vtabwa*, afin de corriger d'éventuelles valeurs hors-limites.

## Crédits

Écrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# vwrap

vwrap — Limitation et enroulement de signaux vectoriels.

## Description

Enroule les éléments de signaux vectoriels de contrôle.

## Syntaxe

```
vwrap ifn, kmin, kmax, ielements
```

## Initialisation

*ifn* - numéro de la table hébergeant le vecteur à traiter.

*ielements* - nombre de composantes du vecteur.

## Exécution

*kmin* - valeur du seuil inférieur.

*kmax* - valeur du seuil supérieur.

*vwrap* enroule chaque élément du vecteur correspondant s'il dépasse les seuils inférieur ou supérieur.

Ces opcodes sont semblables à *limit*, *wrap* et *mirror*, mais ils opèrent sur un signal vectoriel au lieu d'un signal scalaire.

Le résultat écrase les anciennes valeurs de *ifn1*, si celles-ci sont en dehors de l'intervalle min/max. Si l'on veut conserver le vecteur d'entrée, il faut utiliser l'opcode *vcopy* pour le copier dans une autre table.

Tous ces opcodes sont conçus pour être utilisés avec d'autres opcodes qui opèrent sur des signaux vectoriels tels que *vcella*, *adsynt*, *adsynt2*, etc.

## Crédits

Ecrit par Gabriel Maldonado.

Nouveau dans Csound 5 (Auparavant seulement disponible dans CsoundAV)

# waveset

waveset — Un variateur de durée simple par répétition de périodes.

## Description

Un variateur de durée simple par répétition de périodes.

## Syntaxe

```
ares waveset ain, krep [, ilen]
```

## Initialisation

*ilen* (facultatif, 0 par défaut) -- la longueur (en échantillons) du signal audio. Si *ilen* vaut 0, la moitié de la longueur de la note donnée (p3) est prise.

## Exécution

*ain* -- le signal audio en entrée.

*krep* -- le nombre de fois que la période est répétée.

L'entrée est lue et chaque période complète (deux passages par zéro) est répétée *krep* fois.

Il y a un tampon interne car la sortie est évidemment plus lente que l'entrée. Il faut faire attention si le tampon est trop court, car il peut y avoir des effets étranges.

## Exemples

Voici un exemple de l'opcode waveset. Il utilise le fichier *waveset.csd* [examples/waveset.csd].

### Exemple 1133. Exemple de l'opcode waveset.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;;realtime audio out
;-iadc      ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o waveset.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs  =1

instr 1
```

```
krep init p4
asig soundin "flute.aiff"
aout waveset asig, krep
    outs aout, aout

endin
</CsInstruments>
<CsScore>

i 1 0 3 1 ;no repetitions
i 1 + 10 3 ;stretching 3 times
i 1 + 14 6 ;6 times

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch  
Février 2001

Nouveau dans la version 4.11

# websocket

websocket — Lit et écrit des signaux et des tableaux en utilisant une connexion WebSocket.

## Description

Opcode du greffon websocketIO.

*websocket* lit et écrit  $N$  signaux et tableaux en utilisant une connexion WebSocket.

## Syntaxe

```
xout1[, xout2, xout3, ..., xoutN] websocket iport, xin
```

## Initialisation

*iport* -- Le port web local où lire/écrire les données.

## Exécution

## Exécution

*xout1, ..., xoutN* -- Les variables en sortie contenant les données reçues du WebSocket. Côté Web, le WebSocket doit envoyer les données en utilisant un nom de protocole qui correspond au nom de la variable de sortie, par exemple "ksignal" pour une variable de taux-k. Si un tableau est attendu il doit d'abord être initialisé avant d'être utilisé comme sortie de l'opcode. Sinon l'opcode ne connaît pas la taille des données attendues du WebSocket. Lorsque l'on envoie des données à un WebSocket depuis la page Web, elles doivent être envoyées sous la forme d'un tableau de nombres 32 ou 64 bit en fonction de la version de Csound utilisée.

*xin1* -- La variable d'entrée qui contient les données à envoyer au WebSocket. Côté Web, le WebSocket reçoit les données en utilisant un nom de protocole qui correspond au nom de la variable d'entrée, par exemple "ksignal" pour une variable de taux-k. Lorsque l'on reçoit des données d'un WebSocket sur la page Web, il faut les lire comme tableau de nombres 32 ou 64 bit en fonction de la version de Csound utilisée.



### Note

Le nombre total d'arguments d'entrée et de sortie est limité à 20.

Les variables de taux-a doivent être envoyées et reçues sous la forme de tableaux contenant *ksmps* échantillons. Les tableaux de taux-a sont de même envoyés et reçus comme *ksmps* éléments dans le tableau. Les variables de taux-k sont envoyées et reçues comme un tableau à un seul élément. Les tableaux de taux-k sont envoyés et reçus comme des tableaux ayant le même nombre d'éléments.

## Exemples

Voici un exemple simple de l'opcode websocket. Il utilise les fichiers *websocket.csd* [examples/websocket.csd] et *websocket.html* [examples/websocket.html].

### Exemple 1134. Exemple de l'opcode websocket.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>
nchnls = 2
0dbfs = 1
ksmps = 256
sr = 44100

schedule 1, 0, -1

instr 1
  klfo lfo 1, 1
  iport init 8888
  kinput websocket iport, klfo
  printk2 kinput
endin

</CsInstruments>
<CsScore>
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*OSClisten OSCsend*

## Crédits

Auteur : Edward Costello;  
NUIM, 2015

Nouveau dans la version 6.06



# weibull

weibull — Générateur de nombres aléatoires de distribution de Weibull (valeurs positives seulement).

## Description

Générateur de nombres aléatoires de distribution de Weibull (valeurs positives seulement). C'est un générateur de bruit de classe x.

## Syntaxe

```
ares weibull ksigma, ktau
```

```
ires weibull ksigma, ktau
```

```
kres weibull ksigma, ktau
```

## Exécution

*ksigma* -- contrôle l'échelle de l'étalement de la distribution.

*ktau* -- s'il est supérieur à un, les nombres proches de *ksigma* sont favorisés. S'il est inférieur à un, les petites valeurs sont favorisées. S'il est égal à 1, la distribution est exponentielle. Ne produit que des nombres positifs.

Pour des explications plus détaillées sur ces distributions, consulter :

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

## Exemples

Voici un exemple de l'opcode weibull. Il utilise le fichier *weibull.csd* [examples/weibull.csd].

### Exemple 1135. Exemple de l'opcode weibull.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o weibull.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
```

```

Odbfs = 1

instr 1  ; every run time same values

ktri weibull 100, 1
printk .2, ktri  ; look
aout oscili 0.8, 440+ktri, 1  ; & listen
outs aout, aout
endin

instr 2  ; every run time different values

seed 0
ktri weibull 100, 1
printk .2, ktri  ; look
aout oscili 0.8, 440+ktri, 1  ; & listen
outs aout, aout
endin

instr 3  ; every run time different values

seed 0
ktri weibull 100, 10  ; closer to ksigma..
printk .2, ktri  ; look
aout oscili 0.8, 440+ktri, 1  ; & listen
outs aout, aout
endin
</CsInstruments>
<CsScore>
; sine wave
f 1 0 16384 10 1

i 1 0 2
i 2 3 2
i 3 6 2
e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

i 1 time 0.00073: 168.59070
i 1 time 0.20027: 98.72078
i 1 time 0.40054: 48.57351
i 1 time 0.60009: 171.46941
i 1 time 0.80036: 50.20434
i 1 time 1.00063: 50.84775
i 1 time 1.20018: 18.16301
i 1 time 1.40045: 44.41001
i 1 time 1.60000: 0.98506
i 1 time 1.80027: 36.19192

```

WARNING: Seeding from current time 2444541554

```

i 2 time 3.00045: 20.81653
i 2 time 3.20000: 116.17060
i 2 time 3.40027: 9.23891
i 2 time 3.59982: 95.67111
i 2 time 3.80009: 296.52851
i 2 time 4.00036: 39.28636
i 2 time 4.19991: 13.54326
i 2 time 4.40018: 54.92388
i 2 time 4.59973: 268.05584
i 2 time 4.80000: 95.27069
i 2 time 5.00027: 91.62076

```

WARNING: Seeding from current time 2447542341

i	3	time	6.00091:	94.40902
i	3	time	6.20045:	111.10193
i	3	time	6.40073:	99.38797
i	3	time	6.60027:	98.54267
i	3	time	6.80054:	106.53899
i	3	time	7.00082:	106.30752
i	3	time	7.20036:	88.75486
i	3	time	7.40063:	106.45703
i	3	time	7.60091:	84.59854
i	3	time	7.80045:	106.76515

## Voir aussi

*seed, betarand, bexprnd, cauchy, exprand, gauss, linrand, pcauchy, poisson, trirand, unirand*

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

# wgbow

wgbow — Simule un son de corde frottée.

## Description

La sortie audio simule un son de corde frottée, réalisé au moyen d'un modèle physique développé par Perry Cook, mais recodé pour Csound.

## Syntaxe

```
ares wgbow kamp, kfreq, kpres, krat, kvibf, kvamp \  
[, ifn] [, iminfreq]
```

## Initialisation

*ifn* -- table facultative contenant la forme du vibrato, par défaut une table de sinus.

*iminfreq* (facultatif) -- fréquence la plus grave à laquelle l'instrument sera joué. Si elle est omise, elle prend la valeur initiale de *kfreq*. Si *iminfreq* est négative, l'initialisation est ignorée.

## Exécution

Une note est jouée sur un instrument de type corde, avec les arguments ci-dessous.

*kamp* -- amplitude de la note.

*kfreq* -- fréquence de la note jouée.

*kpres* -- un paramètre contrôlant la pression de l'archet sur la corde. Les valeurs doivent se situer autour de 3. L'intervalle utile va approximativement de 1 à 5.

*krat* -- la position de l'archet le long de la corde. Le jeu habituel se fait environ à 0.127236. L'intervalle recommandé va de 0.025 à 0.23.

*kvibf* -- fréquence du vibrato en Hz. L'intervalle recommandé va de 0 à 12.

*kvamp* -- l'amplitude du vibrato.

## Exemples

Voici un exemple de l'opcode wgbow. Il utilise le fichier *wgbow.csd* [examples/wgbow.csd].

### Exemple 1136. Exemple de l'opcode wgbow.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac      ;;;realtime audio out
```

```

;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o wgbow.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kpres = p4      ;pressure value
krat = p5      ;position along string
kvibf = 6.12723

kvib linseg 0, 0.5, 0, 1, 1, p3-0.5, 1 ; amplitude envelope for the vibrato.
kvamp = kvib * 0.01
asig wgbow .7, 55, kpres, krat, kvibf, kvamp, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 2048 10 1 ;sine wave

i 1 0 3 3 0.127236
i 1 + 3 5 0.127236
i 1 + 3 5 0.23

e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 3.47 de Csound

*ifn* est devenu facultatif dans la version 6.06

# wgbowedbar

wgbowedbar — Modèle physique d'une barre frottée.

## Description

Modèle physique d'une barre frottée, appartenant à la famille des instruments à guide d'onde de Perry Cook.

## Syntaxe

```
ares wgbowedbar kamp, kfreq, kpos, kbowpres, kgain [, iconst] [, itvel] \  
    [, ibowpos] [, ilow]
```

## Initialisation

*iconst* (facultatif, 0 par défaut) -- une constante d'intégration. Vaut zéro par défaut.

*itvel* (facultatif, 0 par défaut) -- 0 ou 1. Quand *itvel* = 0, la vitesse de l'archet suit une trajectoire de type ADSR. Quand *itvel* = 1, la valeur de la vélocité de l'archet décroît exponentiellement.

*ibowpos* (facultatif, 0 par défaut) -- la position sur l'archet, qui affecte la trajectoire de vélocité de l'archet.

*ilow* (facultatif, 0 par défaut) -- fréquence la plus basse désirée.

## Exécution

*kamp* -- amplitude du signal.

*kfreq* -- fréquence du signal.

*kpos* -- position de l'archet sur la barre, comprise entre 0 et 1.

*kbowpres* -- pression de l'archet (comme dans *wgbowed*)

*kgain* -- gain du filtre. On recommande une valeur d'environ 0.809.

## Exemples

Voici un exemple de l'opcode *wgbowedbar*. Il utilise le fichier *wgbowedbar.csd* [examples/wgbowedbar.csd].

### Exemple 1137. Exemple de l'opcode *wgbowedbar*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac      ;;realtime audio out  
;-iadc     ;;uncomment -iadc if RT audio input is needed too  
; For Non-realtime ouput leave only the line below:
```

```
; -o wgbowedbar.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kp = p6
asig wgbowedbar p4, cpspch(p5), 1, kp, 0.995
outs asig, asig

endin
</CsInstruments>
<CsScore>
s
i1 0 .5 .5 7.00 .1 ;short sound
i1 + . .3 8.00 .1
i1 + . .5 9.00 .1
s
i1 0 .5 .5 7.00 1 ;longer sound
i1 + . .3 8.00 1
i1 + . .5 9.00 1

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
Université de Bath, Codemist Ltd.  
Bath, UK

Nouveau dans la version 4.07 de Csound

# wgbrass

wgbrass — Simule un son de cuivre.

## Description

La sortie audio simule un son de cuivre, réalisé au moyen d'un modèle physique développé par Perry Cook, mais recodé pour Csound.

## Syntaxe

```
ares wgbrass kamp, kfreq, ktens, iatt, kvibf, kvamp \  
    [, ifn] [, iminfreq]
```

## Initialisation

*iatt* -- temps requis pour atteindre la pression nominale.

*ifn* -- table facultative contenant la forme du vibrato, par défaut une table de sinus.

*iminfreq* -- (facultatif) -- fréquence la plus grave à laquelle l'instrument sera joué. Si elle est omise, elle prend la valeur initiale de *kfreq*. Si *iminfreq* est négative, l'initialisation est ignorée.

## Exécution

Une note est jouée sur un instrument de type cuivre, avec les arguments ci-dessous.

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée.

*ktens* -- Tension des lèvres de l'instrumentiste. La valeur recommandée vaut environ 0.4.

*kvibf* -- Fréquence du vibrato en Hz. L'intervalle recommandé va de 0 à 12.

*kvamp* -- amplitude du vibrato



### NOTE

Ceci est assez pauvre et non contrôlable. Il faut une révision, et probablement plus de paramètres.

## Exemples

Voici un exemple de l'opcode wgbrass. Il utilise le fichier *wgbrass.csd* [examples/wgbrass.csd].

### Exemple 1138. Exemple de l'opcode wgbrass.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.



```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o wgbrass.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 10
nchnls = 1
0dbfs = 1

; Instrument #1.
instr 1
  kamp = 0.7
  kfreq = p4
  ktens = p5
  iatt = p6
  kvibf = p7
  ifn = 1

  ; Create an amplitude envelope for the vibrato.
  kvamp line 0, p3, 0.5

  a1 wgbrass kamp, kfreq, ktens, iatt, kvibf, kvamp, ifn
  out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 1024 10 1

;      freq  tens  att  vibf
i 1 0 4  440    0.4  0.1  6.137
i 1 4 4  440    0.4  0.01 0.137
i 1 8 4  880    0.4  0.1  6.137
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitich (d'après Perry Cook)  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 3.47 de Csound

*ifn* est devenu facultatif dans la version 6.06

# wgclar

wgclar — Simule un son de clarinette.

## Description

La sortie audio simule un son de clarinette, réalisé au moyen d'un modèle physique développé par Perry Cook, mais recodé pour Csound.

## Syntaxe

```
ares wgclar kamp, kfreq, kstiff, \  
iatt, idetk, kngain, kvibf, kvamp [, ifn] [, iminfreq]
```

## Initialisation

*iatt* -- temps en secondes nécessaire pour atteindre la pression de souffle nominale. 0.1 semble correspondre à un jeu raisonnable. Une durée plus longue donne un son initial de vent défini.

*idetk* -- temps en secondes pour arrêter le souffle. 0.1 correspond à une extinction douce.

*ifn* -- table facultative contenant la forme du vibrato, par défaut une table de sinus.

*iminfreq* (facultatif) -- fréquence la plus grave à laquelle l'instrument sera joué. Si elle est omise, elle prend la valeur initiale de *kfreq*. Si *iminfreq* est négative, l'initialisation est ignorée.

## Exécution

Une note est jouée sur un instrument de type clarinette, avec les arguments ci-dessous.

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée.

*kstiff* -- Paramètre de raideur de l'anche. Les valeurs doivent être négatives, aux environs de -0.3. L'intervalle utile est compris approximativement entre -0.44 et -0.18.

*kngain* -- Amplitude de la composante de bruit, approximativement comprise entre 0 et 0.5.

*kvibf* -- Fréquence du vibrato en Hz. L'intervalle recommandé va de 0 à 12.

*kvamp* -- Amplitude du vibrato

## Exemples

Voici un exemple de l'opcode wgclar. Il utilise le fichier *wgclar.csd* [examples/wgclar.csd].

### Exemple 1139. Exemple de l'opcode wgclar.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o wgclar.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kfreq = 330
kstiff = -0.3
iatt = 0.1
idetk = 0.1
kngain init p4 ;vary breath
kvibf = 5.735
kvamp = 0.1

asig wgclar .9, kfreq, kstiff, iatt, idetk, kngain, kvibf, kvamp, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave

i 1 0 2 0.2
i 1 + 2 0.5 ;more breath
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
 Université de Bath, Codemist Ltd.  
 Bath, UK

Nouveau dans la version 3.47 de Csound

*ifn* est devenu facultatif dans la version 6.06

# wgflute

wgflute — Simule un son de flûte.

## Description

La sortie audio simule un son de flûte, réalisé au moyen d'un modèle physique développé par Perry Cook, mais recodé pour Csound.

## Syntaxe

```
ares wgflute kamp, kfreq, kjet, iatt,  
      idetk, kngain, kvibf, kvamp [, ifn] [, iminfreq] [, ijetrf] [, iendrf]
```

## Initialisation

*iatt* -- temps en secondes nécessaire pour atteindre la pression de souffle nominale. 0.1 semble correspondre à un jeu raisonnable.

*idetk* -- temps en secondes pour arrêter le souffle. 0.1 correspond à une extinction douce.

*ifn* -- table facultative contenant la forme du vibrato, par défaut une table de sinus.

*iminfreq* (facultatif) -- fréquence la plus grave à laquelle l'instrument sera joué. Si elle est omise, elle prend la valeur initiale de *kfreq*. Si *iminfreq* est négative, l'initialisation est ignorée.

*ijetrf* (facultatif, 0.5 par défaut) -- quantité de réflexion dans le jet d'air qui excite la flûte. La valeur par défaut est 0.5.

*iendrf* (facultatif, 0.5 par défaut) -- coefficient de réflexion du jet d'air. La valeur par défaut est 0.5. *ijetrf* et *iendrf* sont utilisés dans le calcul de la pression différentielle.

## Exécution

*kamp* -- Amplitude de la note.

*kfreq* -- Fréquence de la note jouée. Elle peut être variée pendant l'exécution, mais je n'ai pas essayé.

*kjet* -- un paramètre contrôlant le jet d'air. Ses valeurs doivent être positives, aux environs de 0.3. L'intervalle utile est compris approximativement entre 0.08 et 0.56.

*kngain* -- amplitude de la composante de bruit, approximativement comprise entre 0 et 0.5.

*kvibf* -- fréquence du vibrato en Hz. L'intervalle recommandé va de 0 à 12

*kvamp* -- amplitude du vibrato

## Exemples

Voici un exemple de l'opcode wgflute. Il utilise le fichier *wgflute.csd* [examples/wgflute.csd].

## Exemple 1140. Exemple de l'opcode wgflute.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o wgflute.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kfreq = 440
kjet init p4 ;vary air jet
iatt = 0.1
idetk = 0.1
kngain = 0.15
kvibf = 5.925
kvamp = 0.05

asig wgflute .8, kfreq, kjet, iatt, idetk, kngain, kvibf, kvamp, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave

i 1 0 2 0.02 ;more air jet
i 1 + 2 0.32
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
Université de Bath, Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.47 de Csound

*ifn* est devenu facultatif dans la version 6.06

# wgpluck

wgpluck — Une simulation haute fidélité de corde pincée.

## Description

Une simulation haute fidélité de corde pincée, utilisant des lignes à retard avec interpolation.

## Syntaxe

```
ares wgpluck icps, iamp, kpick, iplk, idamp, ifilt, axcite
```

## Initialisation

*icps* -- fréquence de la corde pincée

*iamp* -- amplitude de la corde pincée

*iplk* -- point d'excitation le long de la corde, dans l'intervalle compris entre 0 et 1. 0 = pas d'excitation.

*idamp* -- amortissement de la note. Il contrôle l'extinction globale de la corde. Plus la valeur de *idamp* est importante, plus la décroissance est rapide. Avec une valeur négative, il y aura un accroissement progressif de la sortie.

*ifilt* -- contrôle l'atténuation du filtre sur le chevalet. Les valeurs élevées provoquent une décroissance plus rapide des harmoniques supérieurs.

## Exécution

*kpick* -- Fraction de la longueur de la corde où sera lue la sortie.

*axcite* -- un signal d'excitation de la corde.

Une corde de fréquence *icps* est pincée avec l'amplitude *iamp* au point *iplk*. L'extinction de la corde virtuelle est contrôlée par *idamp* et *ifilt* qui simule le chevalet. L'oscillation est lue au point *kpick*, et excitée par le signal *axcite*.

## Exemples

L'exemple suivant produit une note moyennement longue avec une décroissance rapide des partiels supérieurs. Il utilise le fichier *wgpluck.csd* [examples/wgpluck.csd].

### Exemple 1141. Un exemple de l'opcode wgpluck.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in    No messages
```

```

-odac          -iadc      -d      ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wgpluck.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
instr 1
  icps = 220
  iamp = 20000
  kpick = 0.5
  iplk = 0
  idamp = 10
  ifilt = 1000

  excite oscil 1, 1, 1
  apluck wgpluck icps, iamp, kpick, iplk, idamp, ifilt, excite

  out apluck
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>

```

L'exemple suivant produit une note plus courte et plus brillante. Il utilise le fichier file *wgpluck\_brighter.csd* [examples/wgpluck\_brighter.csd].

### Exemple 1142. Un exemple de l'opcode *wgpluck* avec une note plus courte et plus brillante.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac          -iadc      -d      ;;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o wgpluck_brighter.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

```

```
; Instrument #1.
instr 1
  icps = 220
  iamp = 20000
  kpick = 0.5
  iplk = 0
  idamp = 30
  ifilt = 10

  axcite oscil 1, 1, 1
  apluck wgppluck icps, iamp, kpick, iplk, idamp, ifilt, axcite

  out apluck
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Michael A. Casey  
M.I.T.  
Cambridge, Mass.  
1997

Nouveau dans la Version 3.47



# wgpluck2

wgpluck2 — Modèle physique de corde pincée.

## Description

*wgpluck2* est une implémentation du modèle physique de corde pincée. On peut contrôler le point d'excitation, le point de lecture et le filtre. Basé sur l'algorithme de Karplus-Strong.

## Syntaxe

ares **wgpluck2** *iplk*, *kamp*, *icps*, *kpick*, *krefl*

## Initialisation

*iplk* -- Le point d'excitation est *iplk*, qui représente une fraction de la longueur de la corde (0 à 1). Un point d'excitation de zéro signifie l'absence d'excitation initiale.

*icps* -- La corde produit une hauteur *deicps*.

## Exécution

*kamp* -- Amplitude de la note.

*kpick* -- Fraction de la longueur de la corde où sera lue la sortie.

*krefl* -- le coefficient de réflexion, indiquant l'amortissement et le taux d'extinction. Il doit être strictement compris entre 0 et 1 (il n'acceptera pas 0 ni 1).

## Exemples

Voici un exemple de l'opcode *wgpluck2*. Il utilise le fichier *wgpluck2.csd* [exemples/wgpluck2.csd].

### Exemple 1143. Exemple de l'opcode *wgpluck2*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o wgpluck2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
```

```
; Instrument #1.
instr 1
  iplk = 0.75
  kamp = 30000
  icps = 220
  kpick = 0.75
  krefl = 0.5

  apluck wgpluck2 iplk, kamp, icps, kpick, krefl

  out apluck
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for two seconds.
i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*repluck*

## Crédits

Auteur : John ffitch (d'après Perry Cook)  
Université de Bath, Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.47 de Csound

# wguide1

`wguide1` — Un modèle simple de guide d'onde constitué d'une ligne à retard et d'un filtre passe-bas du premier ordre.

## Description

Un modèle simple de guide d'onde constitué d'une ligne à retard et d'un filtre passe-bas du premier ordre.

## Syntaxe

```
ares wguide1 asig, xfreq, kcutoff, kfeedback
```

## Exécution

*asig* -- l'entrée du bruit d'excitation.

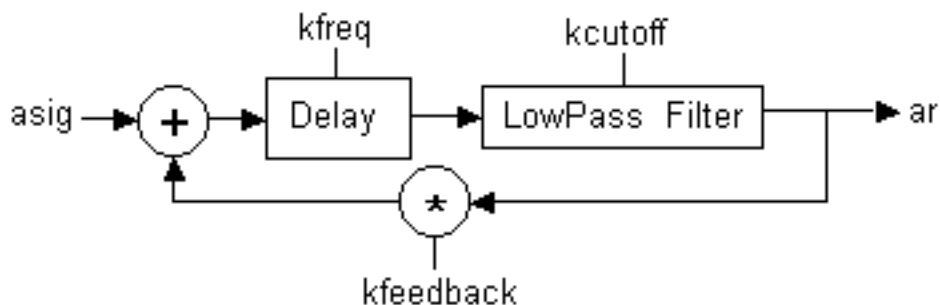
*xfreq* -- la fréquence (c-à-d l'inverse de la durée du retard). A été changé au taux-x dans la version 3.59 de Csound.

*kcutoff* -- la fréquence de coupure du filtre en Hz.

*kfeedback* -- le facteur de rétroaction.

*wguide1* est le modèle de guide d'onde le plus élémentaire, consistant en une ligne à retard et un filtre passe-bas du premier ordre.

L'implémentation des algorithmes de guide d'onde comme opcodes au lieu d'instruments d'un orchestre de Csound permet de fixer une valeur de *kr* différente de celle de *sr*, ce qui donne de meilleures performances particulièrement en .



`wguide1`.

## Exemples

Voici un exemple de l'opcode `wguide1`. Il utilise le fichier `wguide1.csd` [examples/wguide1.csd].

## Exemple 1144. Exemple de l'opcode `wguide1`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o wguide1.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1 - a simple noise waveform.
instr 1
; Generate some noise.
asig noise 20000, 0.5

out asig
endin

; Instrument #2 - a waveguide example.
instr 2
; Generate some noise.
asig noise 20000, 0.5

; Run it through a wave-guide model.
kfreq init 200
kcutoff init 3000
kfeedback init 0.8
awg1 wguide1 asig, kfreq, kcutoff, kfeedback

out awg1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 2 seconds.
i 1 0 2
; Play Instrument #2 for 2 seconds.
i 2 2 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

`wguide2`

## Crédits

Auteur : Gabriel Maldonado

Italie

Octobre 1998

Exemple écrit par Kevin Conder.

Nouveau dans la version 3.49 de Csound.

# wguide2

**wguide2** — Un modèle de plaque frappée constitué de deux lignes à retard en parallèle et de deux filtres passe-bas du premier ordre.

## Description

Un modèle de plaque frappée constitué de deux lignes à retard en parallèle et de deux filtres passe-bas du premier ordre.

## Syntaxe

```
ares wguide2 asig, xfreq1, xfreq2, kcutoff1, kcutoff2, \  
      kfeedback1, kfeedback2
```

## Exécution

*asig* -- l'entrée du bruit d'excitation.

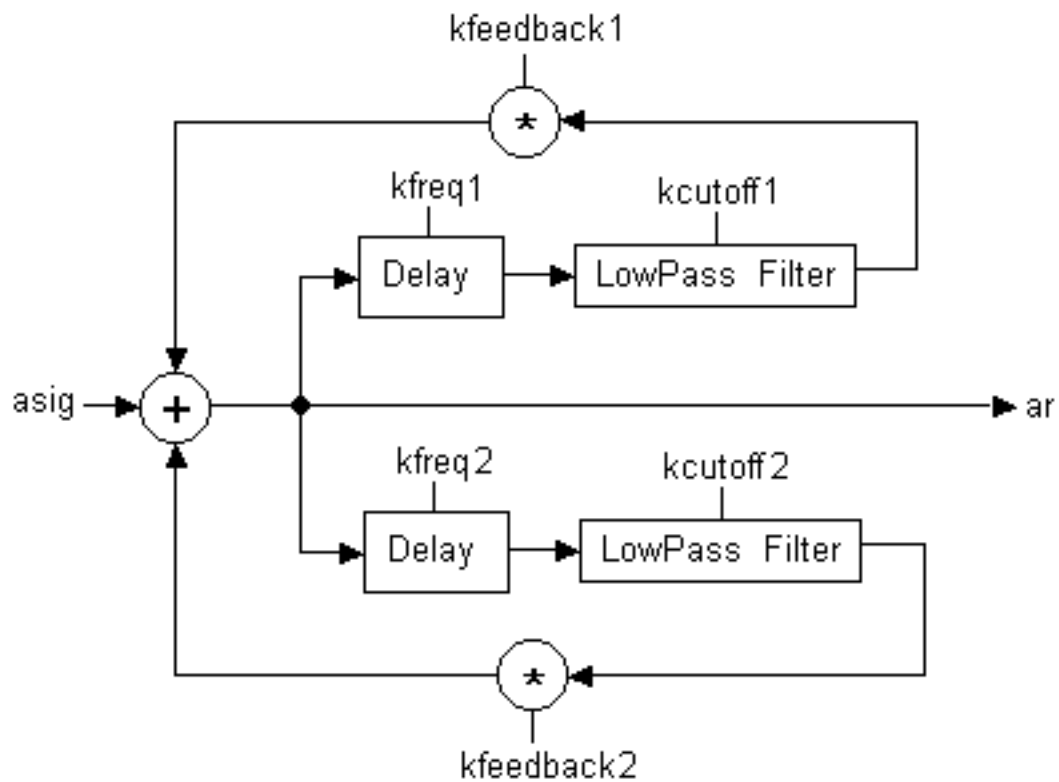
*xfreq1*, *xfreq2* -- la fréquence (c-à-d l'inverse de la durée du retard). A été changé au taux-x dans la version 3.59 de Csound.

*kcutoff1*, *kcutoff2* -- la fréquence de coupure du filtre en Hz.

*kfeedback1*, *kfeedback2* -- le facteur de rétroaction.

*wguide2* est un modèle de plaque frappée consistant en deux lignes à retard en parallèle et deux filtres passe-bas du premier ordre. Les deux lignes de rétroaction sont mélangées et réenvoyées au délai à chaque cycle.

L'implémentation des algorithmes de guide d'onde comme opcodes au lieu d'instruments d'un orchestre de Csound permet de fixer une valeur de *kr* différente de celle de *sr*, ce qui donne de meilleures performances particulièrement en .



wguide2.



### Note

De manière empirique, la somme des deux valeurs de rétroaction ne devrait pas dépasser 0.5 pour éviter de rendre *wguide2* instable.

## Exemples

Voici un exemple de l'opcode *wguide2*. Il utilise le fichier *wguide2.csd* [examples/wguide2.csd].

### Exemple 1145. Exemple de l'opcode *wguide2*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o wguide2.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
```

```

nchnls = 2
odbfs = 1

instr 1

aout diskin2 "beats.wav", 1, 0, 1 ;in signal
afreq1 line 100, p3, 2000
afreq2 line 1200, p3, p4 ;vary second frequency in the score
kcutoff1 = 3000
kcutoff2 = 1500
kfeedback1 = 0.25 ;the sum of the two feedback
kfeedback2 = 0.25 ;values should not exceed 0.5
asig wguide2 aout, afreq1, afreq2, kcutoff1, kcutoff2, kfeedback1, kfeedback2
asig dcblock2 asig ;get rid of DC
outs asig, asig

endin
</CsInstruments>
<CsScore>
i 1 0 8 1200 ;frequency of afreq2 remains the same
i 1 9 8 100 ;frequency of afreq2 gets lower
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*wguide1*

## Crédits

Auteur : Gabriel Maldonado  
 Italie  
 Octobre 1998

Nouveau dans la version 3.49 de Csound.



# while

while — Une construction syntactique de boucle.

## Description

Une construction syntactique de boucle.

## Syntaxe

```
while condition do
... od
```

## Exécution

Les instructions entre *do* et *od* forment le corps d'une boucle qui est exécutée tant que la condition reste vraie.

## Exemples

Voici un exemple de la construction while. Il utilise le fichier *while.csd* [examples/while.csd].

### Exemple 1146. Exemple de la construction while.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ifthen.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
lab99:
if p4<0 goto lab100
p4 = p4-1
print p4
goto lab99
lab100:
endin

instr 2
while p4>=0 do
p4 = p4-1
print p4
od
```

```

    endin
  </CsInstruments>
  <CsScore>
    i 1 1 1 4
    i 2 2 1 4
    e

  </CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

B 0.000 .. 1.000 T 1.000 TT 1.000 M: 0.0
new alloc for instr 1:
instr 1: p4 = 3.000
instr 1: p4 = 2.000
instr 1: p4 = 1.000
instr 1: p4 = 0.000
instr 1: p4 = -1.000
B 1.000 .. 2.000 T 2.000 TT 2.000 M: 0.0
new alloc for instr 2:
instr 2: p4 = 3.000
instr 2: p4 = 2.000
instr 2: p4 = 1.000
instr 2: p4 = 0.000
instr 2: p4 = -1.000
B 2.000 .. 3.000 T 3.000 TT 3.000 M: 0.0

```

## Voir aussi

*loop\_ge*, *loop\_gt*, *loop\_le*, *loop\_lt* et *until*.

## Crédits

John ffitch.

Nouveau dans la version 6.04 de Csound

# wiiconnect

wiiconnect — Lit des données provenant de l'un des contrôleurs Wiimote de Nintendo.

## Description

Opcode du greffon wiimote.

Ouvre et interroge au taux de contrôle de un à quatre contrôleurs externes Wiimote de Nintendo.

## Syntaxe

```
ires wiiconnect [itimeout, imaxnum]
```

## Initialisation

*itimeout* -- nombre entier de secondes pendant lesquelles le système doit attendre que toutes les Wiimotes soient connectées. S'il n'est pas spécifié, il vaut 10 secondes par défaut.

*imaxnum* -- nombre maximum de Wiimotes à repérer. S'il n'est pas spécifié, il vaut 4 par défaut.

Initialement, chaque Wiimote montre son allocation numérique en allumant une des quatre LEDs.

*ires* -- valeur de retour qui vaut 1 en cas de succès ou 0 en cas d'erreur.

## Exécution



### Note

Prière de noter que ces opcodes ne sont actuellement supportés que sous Linux.

A chaque cycle de contrôle, chaque Wiimote est interrogée sur son état et sur sa position. Ces valeurs sont lues par l'opcode *wiidata*. Le résultat retourné vaut 1 la plupart du temps, mais sera nul si une Wiimote se déconnecte.

## Exemples

Voici un exemple des opcodes wii. Il utilise le fichier *wii.csd* [examples/wii.csd].

### Exemple 1147. Exemple des opcodes wii.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
--rtaudio=alsa -o dac:hw:0
</CsOptions>
<CsInstruments>
nchnls = 2
ksmps = 400
```

```

#define WII_B          #3#
#define WII_A          #4#
#define WII_R_A        #304#
#define WII_PITCH      #20#
#define WII_ROLL       #21#
#define WII_BATTERY    #27#

#define WII_RUMBLE      #3#
#define WII_SET_LEDS   #4#

gkcnt init 1

instr 1
    i1 wiiconnect 3,1

    wiirange $WII_PITCH., -20, 0
    kb wiidata $WII_BATTERY.
    kt wiidata $WII_B.
    ka wiidata $WII_A.
    kra wiidata $WII_R_A.
    gka wiidata $WII_PITCH.
    gkp wiidata $WII_ROLL.
; If the B (trigger) button is pressed then activate a note
    if (kt==0) goto ee
    event "i", 2, 0, 5
    gkcnt = gkcnt + 1
    wiisend $WII_SET_LEDS., gkcnt
ee:
    if (ka==0) goto ff
    wiisend $WII_RUMBLE., 1
ff:
    if (kra==0) goto gg
    wiisend $WII_RUMBLE., 0
gg:
    printk2 kb
endin

instr 2
    a1 oscil ampdbs(gka), 440+gkp, 1
    outs a1, a1
endin

</CsInstruments>

<CsScore>
f1 0 4096 10 1
i1 0 300

</CsScore>

</CsoundSynthesizer>

```

## Voir aussi

*wiidata, wiirange, wiisend*

## Crédits

Auteur : John ffitich  
 Codemist Ltd  
 2009

Nouveau dans la version 5.11

# wiidata

wiidata — Lit des données provenant de l'un des contrôleurs externes Wiimote de Nintendo.

## Description

Opcode du greffon wiimote.

Lit des données provenant de un à quatre contrôleurs externes Wiimote de Nintendo.

## Syntaxe

```
kres wiidata kcontrol[, knum]
```

## Initialisation

Cet opcode doit être utilisé de pair avec un opcode *wiiconnect* actif.

## Exécution



### Note

Prière de noter que ces opcodes ne sont actuellement supportés que sous Linux.

*kcontrol* -- le code du contrôle à lire

*knun* -- le numéro de la Wiimote à interroger, qui est par défaut la première.

A chaque accès, un type de donnée particulier de la Wiimote est lu. Les contrôles actuellement implémentés sont donnés ci-dessous, avec le nom de macro défini dans le fichier *wii\_mac* :

0 (WII\_BUTTONS) : retourne une combinaison de bits représentant tous les boutons enfoncés.

1 (WII\_TWO) : retourne 1 si le bouton vient d'être enfoncé, 0 sinon.

2 (WII\_ONE) : comme ci-dessus.

3 (WII\_B) : comme ci-dessus.

4 (WII\_A) : comme ci-dessus.

5 (WII\_MINUS) : comme ci-dessus.

8 (WII\_HOME) : comme ci-dessus.

9 (WII\_LEFT) : comme ci-dessus.

10 (WII\_RIGHT) : comme ci-dessus.

11 (WII\_DOWN) : comme ci-dessus.

12 (WII\_UP) : comme ci-dessus.

13 (WII\_PLUS) : comme ci-dessus.

Si le numéro du contrôle vaut 100 plus un de ces codes de bouton, l'état courant du bouton est retourné. Les macros telles que `WII_S_TWO`, etc sont définies pour cela.

Si le numéro du contrôle vaut 200 plus un de ces codes de bouton, la valeur retournée est 1 si le bouton est enfoncé, et 0 sinon. Les macros telles que `WII_H_TWO`, etc sont définies pour cela.

Si le numéro du contrôle vaut 300 plus un de ces codes de bouton, la valeur retournée est 1 si le bouton vient d'être relâché, et 0 sinon. Les macros telles que `WII_R_TWO`, etc sont définies pour cela.

20 (`WII_PITCH`) : L'inclinaison de la Wiimote. La valeur en degrés est comprise entre -90 et +90, à moins d'une modification de l'intervalle par un appel à *wiirange*.

21 (`WII_ROLL`) : La rotation de la Wiimote. La valeur en degrés est comprise entre -90 et +90, à moins d'une modification de l'intervalle par un appel à *wiirange*.

23 (`WII_FORCE_X`) : La force appliquée à la Wiimote selon les trois axes.

24 (`WII_FORCE_Y`) :

25 (`WII_FORCE_Z`) :

26 (`WII_FORCE_TOTAL`) : L'intensité totale de la force appliquée à la Wiimote.

27 (`WII_BATTERY`) : Le pourcentage de la charge des piles restante.

28 (`WII_NUNCHUK_ANG`) : L'angle du joystick du nunchuk en degrés.

29 (`WII_NUNCHUK_MAG`) : Le déplacement du joystick du nunchuk par rapport à sa position centrale.

30 (`WII_NUNCHUK_PITCH`) : L'inclinaison du nunchuk en degrés, comprise entre -90 et +90, à moins d'une modification de l'intervalle par un appel à *wiirange*.

31 (`WII_NUNCHUK_ROLL`) : La rotation du nunchuk en degrés, comprise entre -90 et +90, à moins d'une modification de l'intervalle par un appel à *wiirange*.

33 (`WII_NUNCHUK_Z`): L'état du bouton Z du nunchuk.

34 (`WII_NUNCHUK_C`): L'état du bouton C du nunchuk.

35 (`WII_IR1_X`): Le pointage infrarouge de la Wiimote.

36 (`WII_IR1_Y`):

37 (`WII_IR1_Z`):

## Exemples

Voir l'exemple de *wiiconnect*.

## Voir aussi

*wiiconnect*, *wiirange*, *wiisend*,

## Crédits

Auteur : John ffitich

Codemist Ltd  
2009

Nouveau dans la version 5.11

# wiirange

wiirange — Fixe l'échelle et les limites de l'intervalle de certains des paramètres de la Wiimote.

## Description

Opcodes du greffon wiimote.

Fixe l'échelle et les limites de l'intervalle de certains des paramètres de la Wiimote.

## Syntaxe

```
wiirange icontrol, iminimum, imaximum[, inum]
```

## Initialisation

Cet opcode doit être utilisé de pair avec un opcode *wiiconnect* actif.

*icontrol* -- numéro du contrôle à pondérer. C'est l'un des suivants : 20 (WII\_PITCH), 21 (WII\_ROLL), 30 (WII\_NUNCHUK\_PITCH), 31 (WII\_NUNCHUK\_ROLL).

*iminimum* -- valeur minimale du contrôle.

*imaximum* -- valeur maximale du contrôle.



### Note

Prière de noter que ces opcodes ne sont actuellement supportés que sous Linux.

## Exemples

Voir l'exemple de *wiiconnect*.

## Voir aussi

*wiiconnect*, *wiidata*, *wiisend*,

## Crédits

Auteur : John ffitich  
Codemist Ltd  
2009

Nouveau dans la version 5.11



# wiisend

wiisend — Envoie des données à l'un des contrôleurs externes Wiimote de Nintendo.

## Description

Opcodes du greffon wiimote.

Envoie des données à l'un des contrôleurs externes Wiimote de Nintendo.

## Syntaxe

```
kres wiisend kcontrol, kvalue[, knum]
```

## Initialisation

Cet opcode doit être utilisé de pair avec un opcode *wiiconnect* actif.

## Exécution



### Note

Prière de noter que ces opcodes ne sont actuellement supportés que sous Linux.

*kcontrol* -- le code du contrôle à écrire.

*kvalue* -- la valeur à écrire dans le contrôle.

*knum* -- le numéro de la Wiimote de destination, qui est par défaut la première (zéro).

A chaque accès, un élément de donnée particulier de la Wiimote est écrit. Les contrôles actuellement implémentés sont donnés ci-dessous, avec le nom de macro défini dans le fichier *wii\_mac* :

3 (WII\_RUMBLE) : démarre ou arrête le vibreur de la Wiimote, selon la valeur de *kvalue* (0 pour arrêter, 1 pour démarrer).

4 (WII\_SET\_LEDS) : positionne les quatre LEDs de la Wiimote selon la représentation binaire de *kvalue*.

## Exemples

Voir l'exemple de *wiiconnect*.

## Voir aussi

*wiiconnect*, *wiidata*, *wiirange*,

## Crédits

Auteur : John ffitch  
Codemist Ltd

2009

Nouveau dans la version 5.11

# window

window — Applique une fenêtre à un tableau.

## Description

Applique une forme de fenêtre donnée à un vecteur stocké dans un tableau. La sortie est un tableau contenant le vecteur fenêtré.

## Syntaxe

```
kout[] window kin[],koff, itype]
```

## Initialisation

*itype* -- type de la fenêtre (facultatif) : 0 = Hamming, 1 = Hanning (von Hann) (vaut 1 par défaut).

## Exécution

*kout[]* -- tableau contenant la sortie fenêtrée. Créé s'il n'existe pas.

*kin[]* -- tableau contenant le vecteur d'entrée.

*koff* -- décalage pour faire commencer la fenêtre à la position *koff* (positif ou nul, 0 par défaut).

## Exemples

Voici un exemple de l'opcode window. Il utilise le fichier *window.csd* [examples/window.csd].

### Exemple 1148. Exemple de l'opcode window.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>

<CsOptions>
-d -o dac
</CsOptions>

<CsInstruments>
;ksmps needs to be an integer div of hopsize
ksmps = 64

instr 1

  ihopsize = 256 ; hopsize
  ifftsize = 1024 ; FFT size
  iolaps = ifftsize/ihopsize ; overlaps
  ibw = sr/ifftsize ; bin bandwidth
  kcmt init 0 ; counting vars
  krow init 0

  kOla[] init ifftsize ; overlap-add buffer
  kIn[] init ifftsize ; input buffer
```

```
kOut[][] init iolaps, ifftsize ; output buffers

a1 diskin2 "fox.wav",1,0,1 ; audio input

/* every hopsize samples */
if kcnt == ihopsize then
  /* window and take FFT */
  kWin[] window kIn,krow*ihopsize
  kSpec[] rfft kWin

  /* filter between high and low freqs */
  ilow = 0
  ihigh = 1000
  ki = int(ilow/ibw)
  until ki == int(ihigh/ibw) do
    kSpec[ki] = 0
    ki += 1
  od

  /* IFFT + window */
  kRow[] rifft kSpec
  kWin window kRow, krow*ihopsize
  /* place it on out buffer */
  kOut setrow kWin, krow

  /* zero the ola buffer */
  kOla = 0
  /* overlap-add */
  ki = 0
  until ki == iolaps do
    kRow getrow kOut, ki
    kOla = kOla + kRow
    ki += 1
  od

  /* update counters */
  krow = (krow+1)%iolaps
  kcnt = 0
endif

/* shift audio in/out of buffers */
kIn shiftin a1
a2 shiftout kOla
  out a2/iolaps

/* increment counter */
kcnt += ksmps

endin

</CsInstruments>

<CsScore>
i1 0 10
</CsScore>

</CsoundSynthesizer>
```

## Voir aussi

Opcodes vectoriels, Opcodes de tableaux.

## Crédits

Auteur : Victor Lazzarini

NUI Maynooth  
2014

Nouveau dans la version 6.04

# wrap

wrap — Enroule le signal qui dépasse les limites inférieure ou supérieure.

## Description

Enroule le signal qui dépasse les limites inférieure ou supérieure.

## Syntaxe

```
ares wrap asig, klow, khigh  
ires wrap isig, ilow, ihigh  
kres wrap ksig, klow, khigh
```

## Initialisation

*isig* -- signal d'entrée

*ilow* -- limite inférieure

*ihigh* -- limite supérieure

## Exécution

*xsig* -- signal d'entrée

*klow* -- limite inférieure

*khigh* -- limite supérieure

*wrap* enroule le signal qui dépasse les limites inférieure ou supérieure.

Cet opcode est utile dans plusieurs situations, telles que l'indexation de table ou pour l'écêtage et le modelage de signaux de taux-a, de taux-i ou de taux-k. *wrap* est aussi utile pour le parcours cyclique de tables quand l'indice maximum n'est pas une puissance de deux (voir *table* et *tablei*). Une autre utilisation de *wrap* consiste à répéter des événements de manière cyclique, avec une longueur de cycle arbitraire.

## Exemples

Voici un exemple de l'opcode *wrap*. Il utilise le fichier *wrap.csd* [examples/wrap.csd].

### Exemple 1149. Exemple de l'opcode *wrap*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
-odac      ;;realtime audio out  
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
```

```

; For Non-realtime ouput leave only the line below:
; -o wrap.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
Odbfs = 1
nchnls = 2

instr 1 ; Limit / Mirror / Wrap

igain = p4 ;gain
ilevl1 = p5 ; + level
ilevl2 = p6 ; - level
imode = p7 ;1 = limit, 2 = mirror, 3 = wrap

ain soundin "fox.wav"
ain = ain*igain

if imode = 1 goto limit
if imode = 2 goto mirror

asig wrap ain, ilevl2, ilevl1
goto outsignal

limit:
asig limit ain, ilevl2, ilevl1
goto outsignal

mirror:
asig mirror ain, ilevl2, ilevl1
outsignal:

outs asig*.5, asig*.5 ;mind your speakers

endin

</CsInstruments>
<CsScore>

; Gain +Levl -Levl Mode
i1 0 3 4.00 .25 -1.00 1 ;limit
i1 4 3 4.00 .25 -1.00 2 ;mirror
i1 8 3 4.00 .25 -1.00 3 ;wrap
e
</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*limit, mirror*

## Crédits

Auteur : Gabriel Maldonado  
Italie

Nouveau dans la version 3.49 de Csound

# writescratch

writescratch — Écrit une valeur dans le bloc-notes de l'instance d'un instrument.

## Description

L'opcode *writescratch* écrit l'une des quatre valeurs scalaires que l'on peut enregistrer dans l'instance d'un instrument.

## Syntaxe

```
writescratchival[, index]
```

## Initialisation

*ival* -- variable à écrire.

*index* -- quelle valeur écrire ; vaut zéro par défaut.

## Exemples

Voici un exemple de l'opcode writescratch. Il utilise le fichier *readscratch.csd* [examples/readscratch.csd].

### Exemple 1150. Exemple de l'opcode writescratch.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ilast readscratch
writescratch p2
ilen readscratch 1
writescratch p3, 1
printf_i "last run at %f for %f\n", ilen, ilast, ilen
endin

</CsInstruments>
<CsScore>
i 1 0 1
i 1 2 3
i 1 6 10
e
</CsScore>
```



`</CsoundSynthesizer>`

## Voir aussi

*readscratch,*

## Crédits

Auteur : John ffitch

Mars 2013

Nouveau dans la version 6.00 de Csound.

# wterrain

wterrain — Un opcode simple de synthèse par terrain d'onde.

## Description

Un opcode simple de synthèse par terrain d'onde.

## Syntaxe

```
about wterrain kamp, kpch, k_xcenter, k_ycenter, k_xradius, k_yradius, \  
      itabx, itaby
```

## Initialisation

*itabx, itaby* -- Les deux tables qui définissent le terrain.

## Exécution

La sortie est le résultat du dessin d'une ellipse dont les axes *k\_xradius* et *k\_yradius* centrés en (*k\_xcenter*, *k\_ycenter*), et de sa traversée à la fréquence *kpch*.

## Exemples

Voici un exemple de l'opcode wterrain. Il utilise le fichier *wterrain.csd* [examples/wterrain.csd].

### Exemple 1151. Exemple de l'opcode wterrain.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
; Audio out  No messages  
-odac          -d          ;;RT audio I/O  
; For Non-realtime ouput leave only the line below:  
; -o wterrain.wav -W ;; for file output any platform  
</CsOptions>  
<CsInstruments>  
  
; Initialize the global variables.  
sr = 44100  
kr = 4410  
ksmps = 10  
nchnls = 1  
  
instr 1  
kdclk  linseg 0, 0.01, 1, p3-0.02, 1, 0.01, 0  
kcx    line    0.1, p3, 1.9  
krx    linseg 0.1, p3/2, 0.5, p3/2, 0.1  
kpch   line    cpspch(p4), p3, p5 * cpspch(p4)  
a1     wterrain 10000, kpch, kcx, kcx, -krx, krx, p6, p7  
a1     dcblock a1
```

```

        out      a1*kdclk
    endin

</CsInstruments>
<CsScore>

f1      0      8192      10      1 0 0.33 0 0.2 0 0.14 0 0.11
f2      0      4096      10      1

i1      0      4      7.00 1 1 1
i1      4      4      6.07 1 1 2
i1      8      8      6.00 1 2 2
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Matthew Gillard  
Nouveau dans la version 4.19

# xadsr

`xadsr` — Calcule l'enveloppe ADSR classique.

## Description

Calcule l'enveloppe ADSR classique.

## Syntaxe

```
ares xadsr iatt, idec, islev, irel [, idel]  
kres xadsr iatt, idec, islev, irel [, idel]
```

## Initialisation

*iatt* -- durée de l'attaque (attack)

*idec* -- durée de la première chute (decay)

*islev* -- niveau d'entretien (sustain)

*irel* -- durée de la chute (release)

*idel* -- délai de niveau zéro avant le démarrage de l'enveloppe

## Exécution

L'enveloppe générée évolue dans l'intervalle de 0 à 1 et peut nécessiter un changement d'échelle par la suite, en fonction de l'amplitude demandée. Si l'on utilise *Odbfs* = 1, il sera probablement nécessaire de diminuer l'amplitude de l'enveloppe car plusieurs notes simultanées peuvent provoquer un écrêtage. Si l'on utilise pas *Odbfs*, une mise à l'échelle à une grande amplitude (par exemple 32000) sera peut-être nécessaire.

Voici une description de l'enveloppe :

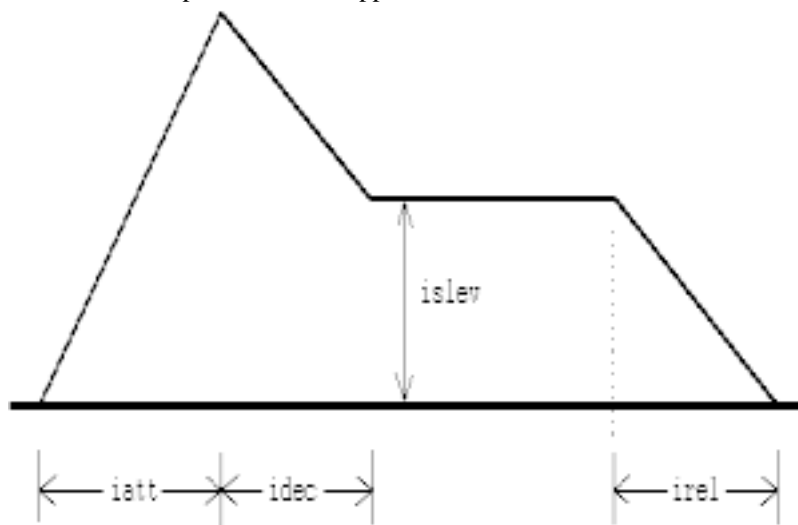


Image d'une enveloppe ADSR.

La longueur de la période d'entretien est calculée à partir de la longueur de la note. C'est pourquoi *xadsr* n'est pas adapté au traitement des événements MIDI, pour lesquels il faut plutôt utiliser *mxadsr*. L'opcode *xadsr* est identique à *adsr* sauf qu'il utilise des segments exponentiels plutôt que linéaires.

## Exemples

Voici un exemple de l'opcode *xadsr*. Il utilise le fichier *xadsr.csd* [examples/xadsr.csd].

### Exemple 1152. Exemple de l'opcode *xadsr*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o xadsr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

iatt = p5
idec = p6
islev = p7
irel = p8

kenv xadsr iatt, idec, islev, irel
kcps = cpspch(p4) ;frequency

asig vco2 kenv * 0.8, kcps
outs asig, asig

endin

</CsInstruments>
<CsScore>

i 1 0 1 7.00 .0001 1 .01 .001 ; short attack
i 1 2 1 7.02 1 .5 .01 .001 ; long attack
i 1 4 2 6.09 .0001 1 .1 .7 ; long release

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*adsr*, *madsr*, *mxadsr*

## Crédits

Auteur : John ffitth *xadsr* est nouveau dans la version 3.51 de Csound.

# xin

xin — Passe des variables à un bloc d'opcode défini par l'utilisateur.

## Description

Les opcodes *xin* et *xout* copient des variables vers et depuis la définition de l'opcode, permettant la communication avec l'instrument appelant.

Les types des variables d'entrée et de sortie sont définis par les paramètres *intypes* et *outtypes*.



### Notes

- *xin* et *xout* ne doivent être appelés qu'une fois, et *xin* doit précéder *xout*, sinon il pourra y avoir une erreur d'initialisation et une désactivation de l'instrument courant.
- Ces opcodes ne sont exécutés que pendant l'initialisation. La copie pendant l'exécution se fait par l'appel de l'opcode défini par l'utilisateur. Cela veut dire que si l'on veut ignorer *xin* ou *xout* avec *kgoto*, cela ne marche pas alors que *igoto* affecte à la fois les opérations de l'initialisation et de l'exécution.

## Syntaxe

```
xinarg1 [, xinarg2] ... [xinargN] xin
```

## Exécution

*xinarg1*, *xinarg2*, ... - arguments d'entrée. Le nombre et le type des variables doit concorder avec la déclaration *intypes* de l'opcode défini par l'utilisateur. Cependant *xin* ne vérifie pas si l'utilisation des variables d'initialisation et du taux de contrôle est correcte.

La syntaxe d'un bloc d'opcode défini par l'utilisateur est la suivante :

```
opcode name, outtypes, intypes
xinarg1 [, xinarg2] [, xinarg3] ... [xinargN] xin
[setksmps iksmps]
... the rest of the instrument's code.
xout xoutarg1 [, xoutarg2] [, xoutarg3] ... [xoutargN]
endop
```

On peut alors utiliser le nouvel opcode avec la syntaxe usuelle :

```
[xinarg1] [, xinarg2] ... [xinargN] name [xoutarg1] [, xoutarg2] ... [xoutargN] [, iksmps]
```

## Exemples

Voir l'exemple de l'opcode *opcode*.

## Voir aussi

*endop*, *opcode*, *setksmps*, *xout*

## Crédits

Auteur : Istvan Varga, 2002 ; basé sur du code par Matt J. Ingalls

Nouveau dans la version 4.22

## xout

*xout* — Récupère les variables d'un bloc d'opcode défini par l'utilisateur.

## Description

Les opcodes *xin* et *xout* copient des variables vers et depuis la définition de l'opcode, permettant la communication avec l'instrument appelant.

Les types des variables d'entrée et de sortie sont définis par les paramètres *intypes* et *outtypes*.



### Notes

- *xin* et *xout* ne doivent être appelés qu'une fois, et *xin* doit précéder *xout*, sinon il pourra y avoir une erreur d'initialisation et une désactivation de l'instrument courant.
- Ces opcodes ne sont exécutés que pendant l'initialisation. La copie pendant l'exécution se fait par l'appel de l'opcode défini par l'utilisateur. Cela veut dire que si l'on veut ignorer *xin* ou *xout* avec *kgoto*, cela ne marche pas alors que *igoto* affecte à la fois les opérations de l'initialisation et de l'exécution.

## Syntaxe

```
xout xoutarg1 [, xoutarg2] ... [, xoutargN]
```

## Exécution

*xoutarg1*, *xoutarg2*, ... - arguments de sortie. Le nombre et le type des variables doit concorder avec la déclaration *outtypes* de l'opcode défini par l'utilisateur. Cependant *xout* ne vérifie pas si l'utilisation des variables d'initialisation et du taux de contrôle est correcte.

La syntaxe d'un bloc d'opcode défini par l'utilisateur est la suivante :

```
opcode name, outtypes, intypes
xinarg1 [, xinarg2] [, xinarg3] ... [xinargN] xin
[setksmps iksmps]
... the rest of the instrument's code.
xout xoutarg1 [, xoutarg2] [, xoutarg3] ... [xoutargN]
endop
```

On peut alors utiliser le nouvel opcode avec la syntaxe usuelle :

```
[xinarg1] [, xinarg2] ... [xinargN] name [xoutarg1] [, xoutarg2] ... [xoutargN] [, iksmps]
```

## Exemples

Voir l'exemple de l'opcode *opcode*.

## Voir aussi

*endop*, *opcode*, *setksmps*, *xin*



## Crédits

Auteur : Istvan Varga, 2002; basé sur du code par Matt J. Ingalls

Nouveau dans la version 4.22

# xscanmap

xscanmap — Permet de lire la position et la vitesse d'un noeud dans une procédure de balayage.

## Description

Opcode du greffon scansyn.

Permet de lire la position et la vitesse d'un noeud dans une procédure de balayage.

## Syntaxe

```
kpos, kvel xscanmap iscan, kamp, kvamp [, iwhich]
```

## Initialisation

*iscan* -- la procédure de balayage à lire

*iwhich* (facultatif) -- le noeud à tester. 0 par défaut.

## Exécution

*kamp* -- facteur d'amplification de la valeur *kpos*.

*kvamp* -- facteur d'amplification de la valeur *kvel*.

L'état interne d'un noeud est lu. Cela comprend sa position et sa vitesse. Ils sont amplifiés par les valeurs *kamp* et *kvamp*.

## Exemples

Voici un exemple de l'opcode xscanmap. Il utilise le fichier *xscanmap.csd* [examples/xscanmap.csd].

### Exemple 1153. Exemple de l'opcode xscanmap.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o xscanmap.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
nchnls_i = 1
0dbfs = 1
;the matrices can be found in /manual/examples
```

```

instr 1 ; Plain scanned syntnesis
; note - scanu display is turned off
a0 = 0
    xscanu 1, .01, 6, 2, "128-stringcircularX", 4, 5, 2, .1, .1, -.01, .1, .5, 0, 0, a0, 0, 0
a1 xscans p4, cpspch(p5), 7, 0, 3
k1,k2 xscanmap 0, 1000, 1000, 64
    display k1, .25 ; note - display is updated every second
    outs a1, a1
endin

instr 2 ; Scan synthesis with audio injection and dual scan paths
; note - scanu display is turned off
ain disk2 "fox.wav",1,0,1
ain in
a0 = ain/10000
    xscanu 1, .01, 6, 2, "128,8-gridX", 14, 5, 2, .01, .05, -.05, .1, .5, 0, 0, a0, 0, 0
a1 xscans p4, cpspch(p5), 7, 0, 2
a2 xscans p4, cpspch(p6), 77, 0, 3
k1,k2 xscanmap 0, 1000, 1000, 127
    display k2, .5 ; note - display is updated ever 500ms
    outs a1,a2
endin

</CsInstruments>
<CsScore>
; Initial condition
;f1 0 16 7 0 8 1 8 0
f1 0 128 7 0 64 1 64 0

; Masses
f2 0 128 -7 1 128 1

; Centering force
f4 0 128 -7 0 128 2
f14 0 128 -7 2 64 0 64 2

; Damping
f5 0 128 -7 1 128 1

; Initial velocity
f6 0 128 -7 -.0 128 .0

; Trajectories
f7 0 128 -5 .001 128 128
f77 0 128 -23 "128-spiral-8,16,128,2,lover2"

; Sine
f9 0 1024 10 1
;-----
; Note list
i1 0 10 .9 7.00
s
i2 0 10 1 8.00 6.00
i2 0 10 1 7.00 8.05
e
</CsScore>
</CsSoundSynthesizer>

```

## Voir aussi

On peut trouver plus d'information sur la synthèse par balayage (de même que d'autres matrices) sur la page *Scanned Synthesis* [<http://www.csounds.com/scanned/>] du site Csound.com.

Il y a aussi un article sur ces opcodes : [http://www.csounds.com/stevenyi/scanned/yi\\_scannedSynthesis.html](http://www.csounds.com/stevenyi/scanned/yi_scannedSynthesis.html), écrit par Steven Yi.

## Crédits

Auteur : John ffitch

Nouveau dans la version 4.20

# xscansmap

xscansmap — Permet de lire la position et la vitesse d'un noeud dans une procédure de balayage.

## Description

Opcodes du greffon scansyn.

Permet de lire la position et la vitesse d'un noeud dans une procédure de balayage.

## Syntaxe

```
xscansmap kpos, kvel, iscan, kamp, kvamp [, iwhich]
```

## Initialisation

*iscan* -- la procédure de balayage à lire

*iwhich* (facultatif) -- le noeud à tester. 0 par défaut.

## Exécution

*kpos* -- la position du noeud.

*kvel* -- la vitesse du noeud.

*kamp* -- facteur d'amplification de la valeur *kpos*.

*kvamp* -- facteur d'amplification de la valeur *kvel*.

L'état interne d'un noeud est lu. Cela comprend sa position et sa vitesse. Ils sont amplifiés par les valeurs *kamp* et *kvamp*.

## Voir aussi

On peut trouver plus d'information sur la synthèse par balayage (de même que d'autres matrices) sur la page *Scanned Synthesis* [<http://www.csounds.com/scanned/>] du site Csounds.com.

Il y a aussi un article sur ces opcodes : [http://www.csounds.com/stevenyi/scanned/yi\\_scannedSynthesis.html](http://www.csounds.com/stevenyi/scanned/yi_scannedSynthesis.html), écrit par Steven Yi.

## Crédits

Nouveau dans la version 4.21

Novembre 2002. Merci à Rasmus Ekman pour avoir précisé cet opcode.

## xscans

xscans — Générateur rapide de forme d'onde et de la table d'onde de la synthèse par balayage.

## Description

Opcode du greffon scansyn.

Version expérimentale de *scans*. Autorise des matrices bien plus grandes, est plus rapide et plus compact, mais supprime une certaine flexibilité (non utilisée ?). S'il est apprécié, il remplacera l'ancien opcode car sa syntaxe est compatible bien qu'étendue.

## Syntaxe

```
ares xscans kamp, kfreq, ifntraj, id [, iorder]
```

## Initialisation

*ifntraj* -- table contenant la trajectoire du balayage. C'est une série de nombres qui contiennent les adresses des masses. L'ordre de ces adresses est utilisé comme chemin de balayage. Ne doit pas contenir de valeurs supérieures au nombre de masses, ou des nombres négatifs. Voir l'*introduction à la section sur la synthèse par balayage*.

*id* -- s'il est positif, c'est l'ID de l'opcode. Il est utilisé pour relier l'opcode de balayage au bon générateur de forme d'onde. S'il est négatif, sa valeur absolue indique la table d'onde dans laquelle sera écrite la forme d'onde. Cette forme d'onde peut être utilisée par la suite par un autre opcode pour générer du son. Le contenu initial de cette table sera écrasé.

*iorder* (facultatif, 0 par défaut) -- ordre de l'interpolation utilisée en interne. Peut prendre n'importe quelle valeur comprise entre 1 et 4, et vaut 4 par défaut, qui est l'interpolation quartique. 2 est l'interpolation quadratique et 1 l'interpolation linéaire. Les nombres les plus élevés donnent un traitement plus lent, mais pas nécessairement meilleur.

## Exécution

*kamp* -- amplitude de la sortie. Noter que l'amplitude résultante dépend aussi des valeurs instantanées de la table d'onde. Ce nombre est en fait la facteur de pondération de la table d'onde.

*kfreq* -- fréquence de balayage

## Format de Matrice

Le nouveau format de matrice est une liste de connexions, une par ligne reliant le point x au point y. Aucun poids n'est affecté au lien ; il est supposé valoir l'unité. La liste est précédée par la ligne <MATRIX> et se termine par une ligne </MATRIX>

Par exemple, une corde circulaire de 8 sera codée par

```
<MATRIX>
0 1
1 0
1 2
2 1
2 3
```

```

3 2
3 4
4 3
4 5
5 4
5 6
6 5
6 7
7 6
0 7
</MATRIX>

```

## Exemples

Voici un exemple de l'opcode `xscans`. Il utilise le fichier `xscans.csd` [examples/xscans.csd].

### Exemple 1154. Exemple de l'opcode `xscans`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o xscans.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
;the matrices can be found in /manual/examples

instr 1 ; Plain scanned syntnesis

a0      =      0
        xscanu    1, .01, 6, 2, "128,8-cylinderX", 4, 5, 2, .1, .1, -.01, .1, .5, 0, 0, a0, 0, 0
a1      xscans    .7, cpspch(p4), 7, 0, 1
        outs      a1, a1
endin

instr 2 ; Scan synthesis with audio injection and dual scan paths
; tap the mic or sing to inject audio into the resonators

a0,aa    ins
a0      =      a0/.8
        xscanu    1, .01, 6, 2, "128,8-gridX", 14, 5, 2, .01, .05, -.05, .1, .5, 0, 0, a0, 0, 0
a1      xscans    .5, cpspch(7.00), 7, 0, 1
a2      xscans    .5, cpspch(7.001), 77, 0, 1
        outs      a1+a2,a1+a2
endin

instr 3 ; Vibrating structure with audio injection
; Tap the MIC - to inject audio into the resonators

a0,aa    ins
a0      =      a0/.8

```

```

                                xscanu      1, .01, 6, 2, "128-stringcircularX", 14, 5, 2, .01, .05, -.05, .25, .75, 0, 0, a0
endin

instr 4 ; Modulated scanners

i1      bexprnd      5
i2      bexprnd      1
ko      oscil        i1, i2, 9
ka1     oscili        .5, .15*8, p7
ka2     oscili        .5, .15*8, p8
kf      oscili        1, .15, p4
kf      =            2^(kf/12)*p6*440+ko
a1      xscans        p9*ka1, kf+i1, 777, 1, 1
a2      xscans        p9*ka2, (kf+i1)*2.1, 77, 1, 1
                                outs        a1+a2, a1+a2
endin

</CsInstruments>
<CsScore>
; Initial condition
f1 0 128 7 0 64 1 64 0
; Masses
f2 0 128 -7 1 128 1
; Centering force
f4 0 128 -7 0 128 2
f14 0 128 -7 2 64 0 64 2
; Damping
f5 0 128 -7 1 128 1
; Initial velocity
f6 0 128 -7 -.0 128 .0
; Trajectories
f7 0 128 -5 .001 128 128
f777 0 128 -23 "128-stringcircular"
f77 0 128 -23 "128-spiral-8,16,128,2,lover2"
; Sine
f9 0 16384 10 1

; Pitch tables
f100 0 1024 -7 +3 128 +3 128 -2 128 -2 128 +0 128 +0 128 -4 128 -4 128 +3
f101 0 1024 -7 -2 128 -2 128 -2 128 -2 128 -5 128 -5 128 -4 128 -4 128 -2
f102 0 1024 -7 +3 128 +3 128 +2 128 +2 128 +0 128 +0 128 +0 128 +0 128 +3
f103 0 1024 -7 +7 128 +7 128 +5 128 +5 128 +3 128 +3 128 +3 128 +3 128 +7

; Amplitude tables
f200 0 1024 7 1 128 0 128 0 127 0 1 1 128 0 128 0 127 0 1 1 128 0 127 0 1 1
f201 0 1024 7 0 127 0 1 1 127 0 1 1 128 0 127 0 1 1 127 0 1 1 128 0 127 0 1 1
f202 0 1024 7 1 127 0 1 1 127 0 1 1 127 0 1 1 127 0 1 1 127 0 1 1 127 0 1 1
f203 0 1024 7 1 1024 0

;-----

; Note list
i1 0 10 6.00
s
i2 1 10
s
i3 1 23
i4 1 23 101 1 .5 200 202 1.5
i4 . . 102 0 .5 200 201 1
i4 . . 103 0 .5 200 201 1
i4 . . 100 0 .25 200 200 2
e
</CsScore>
</CsoundSynthesizer>

```

Pour des exemples similaires, voir la documentation sur *scans*.



## Voir aussi

On peut trouver plus d'information sur la synthèse par balayage (de même que d'autres matrices) sur la page *Scanned Synthesis* [<http://www.csounds.com/scanned/>] du site Csounds.com.

Il y a aussi un article sur ces opcodes : [http://www.csounds.com/stevenyi/scanned/yi\\_scannedSynthesis.html](http://www.csounds.com/stevenyi/scanned/yi_scannedSynthesis.html), écrit par Steven Yi.

*scans, xscanu*

## Crédits

Ecrit par John ffitich.

Nouveau dans la version 4.20

# xscanu

xscanu — Calcule la forme d'onde et la table d'onde à utiliser dans la synthèse par balayage.

## Description

Opcodes du greffon scansyn.

Version expérimentale de *scanu*. Autorise des matrices bien plus grandes, est plus rapide et plus compact, mais supprime une certaine flexibilité (non utilisée ?). S'il est apprécié, il remplacera l'ancien opcode car sa syntaxe est compatible bien qu'étendue.

## Syntaxe

```
xscanu init, irate, ifnvel, ifnmass, ifnstif, ifncentr, ifndamp, kmass, \
      kstif, kcentr, kdamp, ileft,  iright, kpos, kstrngth, ain, idisp, id
```

## Initialisation

*init* -- la position initiale des masses. Si c'est un nombre négatif, alors la valeur absolue de *init* indique la table à utiliser pour la forme du marteau. Si *init* > 0, il doit représenter le nombre de masses attendu, sinon sa valeur est sans importance.

*irate* -- taux de mise à jour.

*ifnvel* -- ftable contenant la vitesse initiale de chaque masse. Sa taille est le nombre de masses attendu.

*ifnmass* -- ftable contenant la valeur de chaque masse. Sa taille est le nombre de masses attendu.

*ifnstif* --

- *soit* une ftable contenant la raideur du ressort de chaque connexion. Sa taille est le carré du nombre de masses attendu. Ses données sont ordonnées selon la succession des lignes de la matrice de connexion du système.
- *soit* une chaîne de caractères donnant le nom d'un fichier au format MATRIX

*ifncentr* -- ftable contenant la force de centrage de chaque masse. Sa taille est le nombre de masses attendu.

*ifndamp* -- ftable contenant le facteur d'amortissement de chaque masse. Sa taille est le nombre de masses attendu.

*ileft* -- si *init* < 0, position du marteau de gauche (*ileft* = 0 frappe complètement à gauche, *ileft* = 1 frappe complètement à droite).

*iright* -- si *init* < 0, position du marteau de droite (*iright* = 0 frappe complètement à gauche, *iright* = 1 frappe complètement à droite).

*idisp* -- s'il vaut 0, il n'y a pas d'affichage des masses.

*id* -- s'il est positif, c'est l'ID de l'opcode. Il est utilisé pour relier l'opcode de balayage au bon générateur de forme d'onde. S'il est négatif, sa valeur absolue indique la table d'onde dans laquelle sera écrite la forme

d'onde. Cette forme d'onde peut être utilisée par la suite par un autre opcode pour générer du son. Le contenu initial de cette table sera écrasé.

## Exécution

*kmass* -- pondère les masses

*kstif* -- pondère la raideur des ressorts

*kcentr* -- pondère la force de centrage

*kdamp* -- pondère l'amortissement

*kpos* -- position d'un marteau actif le long de la corde (*kpos* = 0 est complètement à gauche, *kpos* = 1 est complètement à droite). La forme du marteau est déterminée par *init* et sa puissance de percussion est *kstrngth*.

*kstrngth* -- puissance utilisée par le marteau actif

*ain* -- entrée audio qui s'ajoute à la vitesse des masses. L'amplitude ne doit pas être trop grande.

## Format de Matrice

Le nouveau format de matrice est une liste de connexions, une par ligne reliant le point x au point y. Aucun poids n'est affecté au lien ; il est supposé valoir l'unité. La liste est précédée par la ligne <MATRIX> et se termine par une ligne </MATRIX>

Par exemple, une corde circulaire de 8 sera codée par

```
<MATRIX>
0 1
1 0
1 2
2 1
2 3
3 2
3 4
4 3
4 5
5 4
5 6
6 5
6 7
7 6
0 7
</MATRIX>
```

## Exemples

Voici un exemple de l'opcode *xscanu*. Il utilise le fichier *xscanu.csd* [examples/xscanu.csd].

### Exemple 1155. Exemple de l'opcode *xscanu*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsSoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o xscanu.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
Odbfs = 1
;the matrices can be found in /manual/examples

instr 1 ; Plain scanned syntnesis
; Note Also that I am using quadratic interpolation on these.
a0 = 0
xscanu 1, .01, 6, 2, "128,8-gridX", 4, 5, 2, .1, .1, -.01, .1, .5, 0, 0, a0, 0, 0
a1 xscans .5, cpspch(p4), 333, 0, p6 ; NOTE LEFT RIGHT TRAJECTORY (f333) IS CLEAN!
a1 dcblock a1
outs a1, a1
endin

instr 2 ; Scan synthesis with audio injection and dual scan paths

a0 diskin2 "fox.wav",1,0,1
; a0,aa ins
a0 = a0/.8
xscanu 1, .01, 6, 2, "128,8-torusX", 14, 5, 2, .01, .05, -.05, .1, .5, 0, 0, a0, 0, 0
a1 xscans .3, cpspch(7.00), 333, 0, 2 ; NOTE LEFT RIGHT TRAJECTORY (f333) IS CLEAN!
a2 xscans .3, cpspch(6.00), 77, 0, 2
a1 dcblock a1
a2 dcblock a2
outs a1*.5,a2*.1
endin

</CsInstruments>
<CsScore>

; Initial condition
;f1 0 16 7 0 8 1 8 0
f1 0 128 7 0 64 1 64 0

; Masses
f2 0 128 -7 1 128 1

; Centering force
f4 0 128 -7 0 128 2
f14 0 128 -7 2 64 0 64 2

; Damping
f5 0 128 -7 1 128 1

; Initial velocity
f6 0 128 -7 -.0 128 .0

; Trajectories
f7 0 128 -5 .001 128 128
f77 0 128 -23 "128-spiral-8,16,128,2,lover2"
f777 0 128 -23 "128,8-torusX"

; Spring matrices

```

```
f3 0 128 -23 "128-stringX"
f33 0 128 -23 "128-stringcircularX"
f333 0 128 -23 "128-left_rightX"
f3333 0 128 -23 "128,8-torusX"
f33333 0 128 -23 "128,8-cylinderX"
f333333 0 128 -23 "128,8-gridX"

; Sine
f9 0 1024 10 1

; Pitch tables
f100 0 1024 -7 +3 128 +3 128 -2 128 -2 128 +0 128 +0 128 -4 128 -4 128 +3
f101 0 1024 -7 -2 128 -2 128 -2 128 -2 128 -5 128 -5 128 -4 128 -4 128 -2
f102 0 1024 -7 +3 128 +3 128 +2 128 +2 128 +0 128 +0 128 +0 128 +0 128 +3
f103 0 1024 -7 +7 128 +7 128 +5 128 +5 128 +3 128 +3 128 +3 128 +3 128 +7

; Amplitude tables
f200 0 1024 7 1 128 0 128 0 127 0 1 1 128 0 128 0 127 0 1 1 128 0 127 0 1 1
f201 0 1024 7 0 127 0 1 1 127 0 1 1 128 0 127 0 1 1 127 0 1 1 128 0 127 0 1 1
f202 0 1024 7 1 127 0 1 1 127 0 1 1 127 0 1 1 127 0 1 1 127 0 1 1 127 0 1 1
f203 0 1024 7 1 1024 0
;;f204 0 1024 7 1 512 0 511 0 1 1

;-----

; Note list
i1 0 10 6.00 1 2
s
i2 0 15
e
</CsScore>
</CsoundSynthesizer>
```

Pour des exemples similaires, voir la documentation sur *scans*.

## Voir aussi

On peut trouver plus d'information sur la synthèse par balayage (de même que d'autres matrices) sur la page *Scanned Synthesis* [<http://www.csounds.com/scanned/>] du site Csound.com.

Il y a aussi un article sur ces opcodes : [http://www.csounds.com/stevenyi/scanned/yi\\_scannedSynthesis.html](http://www.csounds.com/stevenyi/scanned/yi_scannedSynthesis.html), écrit par Steven Yi.

*scanu*, *xscans*

## Crédits

Ecrit par John ffitich.

Nouveau dans la version 4.20

# xtratim

xtratim — Allonge la durée d'évènements générés en temps réel.

## Description

Allonge la durée d'évènements générés en temps réel et gère cet allongement. (Habituellement on l'utilise avec *release* au lieu de *linenr*, *linsegr*, etc).

## Syntaxe

**xtratim** iextradur

## Initialisation

*iextradur* -- durée additionnelle pour l'instance courante de l'instrument.

## Exécution

*xtratim* allonge la durée de la note MIDI courante de *iextradur* secondes après que le message note off correspondant ait désactivé cette note. On l'utilise habituellement avec *release*. Cet opcode n'a pas d'arguments en sortie.

Cet opcode est utile pour implémenter des enveloppes complexes avec relâchement, dont la durée n'est pas connue lors du démarrage de l'enveloppe (par exemple pour des évènements MIDI en temps réel).

## Exemples

Voici un exemple de l'opcode xtratim. Il utilise le fichier *xtratim.csd* [examples/xtratim.csd].

### Exemple 1156. Exemple de l'opcode xtratim.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

Cet exemple montre comment générer un segment de relâchement pour une enveloppe ADSR après qu'un note off MIDI ait été reçu, en allongeant la durée avec *xtratim* et en utilisant *release* pour tester si la note est dans sa phase de relâchement.

```
<CsoundSynthesizer>

<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  Silent MIDI in
-odac        -iadc      -d        -MO    ;;realtime I/O
</CsOptions>

<CsInstruments>
;Simple usage of the xtratim opcode
sr = 44100
ksmps = 10
nchnls = 2

; sine wave for oscillators
gisin      ftgen      1, 0, 4096, 10, 1
```

```
instr 1

  inum notnum
  icps cpsmidi
  iamp ampmidi 4000
;
;----- complex envelope block -----
xtratism 1 ;extra-time, i.e. release dur
krel init 0
krel release ;outputs release-stage flag (0 or 1 values)
if (krel == 1) kgoto rel ;if in release-stage goto release section
;
;***** attack and sustain section *****
kmp1 linseg 0, .03, 1, .05, 1, .07, 0, .08, .5, 4, 1, 50, 1
kmp = kmp1*iamp
kgoto done
;
;----- release section -----
rel:
kmp2 linseg 1, .3, .2, .7, 0
kmp = kmp1*kmp2*iamp
done:
;-----
a1 oscili kmp, icps, gisin
outs a1, a1
endin

</CsInstruments>

<CsScore>
f 0 3600 ;dummy table to wait for realtime MIDI events
e
</CsScore>

</CsoundSynthesizer>
```

Voici un exemple plus élaboré de l'opcode xtratism. Il utilise le fichier *xtratism-2.csd* [exemples/xtratism-2.csd].

### Exemple 1157. Exemple plus complexe de l'opcode xtratism.

Cet exemple montre comment générer un segment de relâchement pour une enveloppe ADSR après qu'un note off MIDI ait été reçu, en allongeant la durée avec *xtratism* et en utilisant *release* pour tester si la note est dans sa phase de relâchement. Deux enveloppes sont générées simultanément pour les canaux gauche et droit.

```
<CsoundSynthesizer>

<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  Silent  MIDI in
-odac          -iadc    -d      -M0   ;;realtime I/O
</CsOptions>

<CsInstruments>
;xtratism example by Jonathan Murphy Dec. 2006
sr = 44100
ksmps = 10
nchnls = 2

; sine wave for oscillators
gisin ftgen 1, 0, 4096, 10, 1
; set volume initially to midpoint
ctrlinit 1, 7,64
```

```

;;; simple two oscil, two envelope synth
instr 1

    ; frequency
    kcps      cpsmidib
    ; initial velocity (noteon)
    ivel      veloc

    ; master volume
    kamp      ctrl17 1, 7, 0, 127
    kamp      = kamp * ivel

    ; parameters for aenv1
    iatt1     = 0.03
    idec1     = 1
    isus1     = 0.25
    irel1     = 1
    ; parameters for aenv2
    iatt2     = 0.06
    idec2     = 2
    isus2     = 0.5
    irel2     = 2

    ; extra (release) time allocated
    xtratim   (irel1>irel2 ? irel1 : irel2)
    ; krel is used to trigger envelope release
    krel      init 0
    krel      release
    ; if noteoff received, krel == 1, otherwise krel == 0
    if (krel == 1) kgoto rel

    ; attack, decay, sustain segments
    atmp1     linseg 0, iatt1, 1, idec1, isus1 , 1, isus1
    atmp2     linseg 0, iatt2, 1, idec2, isus2 , 1, isus2
    aenv1     = atmp1
    aenv2     = atmp2
    kgoto     done

    ; release segment
rel:
    atmp3     linseg 1, irel1, 0, 1, 0
    atmp4     linseg 1, irel2, 0, 1, 0
    aenv1     = atmp1 * atmp3 ;to go from the current value (in case
    aenv2     = atmp2 * atmp4 ;the attack hasn't finished) to the release.

    ; control oscillator amplitude using envelopes
done:
    aosc1     oscil aenv1, kcps, gisin
    aosc2     oscil aenv2, kcps * 1.5, gisin
    aosc1     = aosc1 * kamp
    aosc2     = aosc2 * kamp

    ; send aosc1 to left channel, aosc2 to right,
    ; release times are noticeably different

    outs      aosc1, aosc2

    endin

</CsInstruments>

<CsScore>

f 0 3600 ;dummy table to wait for realtime MIDI events

</CsScore>

```



</CsoundSynthesizer>

## Voir aussi

*linenr, release*

## Crédits

Auteur : Gabriel Maldonado

Italie

Exemples par Gabriel Maldonado et Jonathan Murphy

Nouveau dans la version 3.47 de Csound.

# xyin

xyin — Détecte la position du curseur dans une fenêtre de sortie.

## Description

Détecte la position du curseur dans une fenêtre de sortie. Lorsque *xyin* est appelé, la position de la souris dans la fenêtre de sortie est utilisée pour répondre à la requête. En raison de la simplicité de ce mécanisme, on ne peut utiliser de manière précise qu'un seul *xyin* à la fois. La position de la souris est rapportée dans la fenêtre de sortie.

## Syntaxe

```
kx, ky xyin iprd, ixmin, ixmax, iymn, iymax [, ixinit] [, iyinit]
```

## Initialisation

*iprd* -- période de détection du curseur(en secondes). Typiquement 0.1 seconde.

*xmin, xmax, ymin, ymax* -- valeurs limites des coordonnées x-y du curseur dans la fenêtre d'entrée.

*ixinit, iyinit* (facultatif) -- coordonnées x-y initiales rapportées ; les valeurs par défaut sont 0, 0. Si ces valeurs ne sont pas dans l'intervalle min-max donné, elles seront déplacées dans cet intervalle.

## Exécution

*xyin* échantillonne la position x-y du curseur dans une fenêtre d'entrée toutes les *iprd* secondes. Les valeurs retournées sont répétées (pas interpolées) au taux-k, et restent fixes jusqu'à ce qu'un nouveau changement intervienne dans la fenêtre. Il peut y avoir n'importe quel nombre de fenêtres en entrée. Cette unité est utile pour le contrôle en , mais il vaut mieux éviter un mouvement continu si *iprd* est anormalement petit.



### Note

Vous pouvez être amenés à activer les affichages au moyen de l'option de ligne de commande *--displays* en fonction de votre plate-forme et de votre distribution.

## Exemples

Voici un exemple de l'opcode *xyin*. Il utilise le fichier *xyin.csd* [examples/xyin.csd].

### Exemple 1158. Exemple de l'opcode *xyin*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out      Audio in      No messages
-odac             -iadc         --displays ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
```

```
; -o xyin.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 10
nchnls = 2

; Instrument #1.
instr 1
; Print and capture values every 0.1 seconds.
iprd = 0.1
; The x values are from 1 to 30.
ixmin = 1
ixmax = 30
; The y values are from 1 to 30.
iymin = 1
iymax = 30
; The initial values for X and Y are both 15.
ixinit = 15
iyinit = 15

; Get the values kx and ky using the xyin opcode.
kx, ky xyin iprd, ixmin, ixmax, iymin, iymax, ixinit, iyinit

; Print out the values of kx and ky.
printks "kx=%f, ky=%f\\n", iprd, kx, ky

; Play an oscillator, use the x values for amplitude and
; the y values for frequency.
kamp = kx * 1000
kcps = ky * 220
a1 poscil kamp, kcps, 1

outs a1, a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 30 seconds.
i 1 0 30
e

</CsScore>
</CsoundSynthesizer>
```

Lorsque les valeurs de *kx* et de *ky* changent, elles sont affichées comme ceci :

```
kx=8.612036, ky=22.677933
kx=10.765685, ky=15.644135
```

## Crédits

Exemple écrit par Kevin Conder.

# xyscale

xyscale — Interpolation linéaire 2D.

## Description

Opcode du greffon emugens.

Interpolation linéaire 2D entre quatre points à (0,0), (1,0), (0,1) et (1,1)

## Syntaxe

```
kout xyscale kx, ky, k00, k10, k01, k11
```

## Exécution

*kx, ky* -- Coordonnées pour évaluer l'interpolation. Valeurs entre 0 et 1, où :

*k00* -- valeur du point de coordonnées (x=0, y=0)

*k10* -- valeur du point de coordonnées (x=1, y=0)

*k01* -- valeur du point de coordonnées (x=0, y=1)

*k11* -- valeur du point de coordonnées (x=1, y=1)

(0,1)          (1,1)

(0,0)          (1,0)

Etant données quatre valeurs placées aux sommets d'un carré, la valeur interpolée au point (x, y) est trouvée, où x et y sont compris entre 0 et 1.

## Exemples

Voici un exemple de l'opcode xyscale. Il utilise le fichier *xyscale.csd* [examples/xyscale.csd].

### Exemple 1159. Exemple de l'opcode xyscale.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc    -d      ;;RT audio I/O
</CsOptions>
<CsInstruments>

ksmps=128
```

```

nchnls=2

giwidth = 400
giheight = 300
FLpanel "FLmouse", giwidth, giheight, 10, 10
FLpanelEnd

FLrun

Odbfs = 1

instr 1
; We define four chords for bottom-left, bottom-right, top-left and top-right
; Use the mouse to interpolate between them
ibl[] fillarray ntom:i("4C"), ntom:i("4Eb"), ntom:i("4G")
itl[] fillarray ntom:i("4E"), ntom:i("4G#"), ntom:i("4B")
ibr[] fillarray ntom:i("4G"), ntom:i("4A"), ntom:i("4B")
itr[] fillarray ntom:i("4Eb"), ntom:i("4Eb+"), ntom:i("4F")

kmousex, kmousey, kb1, kb2, kb3 FLmouse 2
kx = limit(kmousex/giwidth, 0, 1)
ky = 1 - limit(kmousey/giheight, 0, 1)

printf "x: %f y: %f \n", changed(kx, ky), kx, ky

iamp = 0.1
a0 oscili iamp, mtof(xyscale(kx, ky, ibl[0], itl[0], ibr[0], itr[0]))
a1 oscili iamp, mtof(xyscale(kx, ky, ibl[1], itl[1], ibr[1], itr[1]))
a2 oscili iamp, mtof(xyscale(kx, ky, ibl[2], itl[2], ibr[2], itr[2]))
aout = sum(a0, a1, a2)
outs aout, aout
endin

</CsInstruments>
<CsScore>
i 1 0 120

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*scale, ntrpol, linlin*

## Crédits

Par : Eduardo Moguillansky 2017

# zACL

zACL — Efface une ou plusieurs variables dans l'espace za.

## Description

Efface une ou plusieurs variables dans l'espace za.

## Syntaxe

```
zACL kfirst [, klast]
```

## Exécution

*kfirst* -- Première position za de l'intervalle à effacer.

*klast* -- Dernière position za de l'intervalle à effacer. Si elle n'est pas donnée, seule la position *kfirst* est effacée.

zACL efface une ou plusieurs variables dans l'espace za. Ceci est utile pour les variables utilisées comme accumulateur pour mélanger des signaux de taux-a à chaque cycle, mais qui doivent être effacés avant le prochain groupe de calculs.

## Exemples

Voici un exemple de l'opcode zACL. Il utilise le fichier *zACL.csd* [examples/zACL.csd].

### Exemple 1160. Exemple de l'opcode zACL.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc          -d          ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o zACL.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 32
nchnls = 1
0dbfs = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zACLinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
; Generate a simple sine waveform.
```

```

    asin oscili 0.2, 440

    ; Declick
    asin *= linsegr(0, 0.05, 1, 0.05, 0)

    ; Send the sine waveform to za variable #1.
    zaw asin, 1
    endin

    ; Instrument #2 -- generates audio output.
    instr 2
    ; Send za location #1 to channel 1
    a1 zar 1
    out a1

    ; Clear the za variables, get them ready for
    ; another pass.
    zACL 0, 1
    endin

</CsInstruments>
<CsScore>

    ; Play Instrument #1 for one second.
    i 1 0 1
    ; Play Instrument #2 until end of performance
    i 2 0 -1
    e 1.5

</CsScore>
</CsSoundSynthesizer>

```

## Voir aussi

*zamod, zar, zaw, zawm, ziw, ziwM*

## Crédits

Auteur : Robin Whittle  
 Australie  
 Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zakinit

zakinit — Etablit l'espace zak.

## Description

Etablit l'espace zak. Ne doit être appelé qu'une seule fois.

## Syntaxe

```
zakinit isizea, isizek
```

## Initialisation

*isizea* -- le nombre de positions de taux audio pour les patch de taux-a. Chaque position est un tableau de longueur ksmmps.

*isizek* -- le nombre de positions à réserver pour les nombres en virgule flottante dans l'espace zk. On peut lire et écrire dans celles-ci au taux-i et au taux-k.

## Exécution

Il y a au moins une position d'allouée pour chaque espace za et zk. Il peut y avoir des milliers ou des dizaines de milliers de positions za et zk, mais la plupart des pièces n'en nécessitent probablement que quelques douzaines pour patcher les signaux. Ces positions de patch sont référencées par un numéro dans les autres opcodes zak.

Pour n'exécuter *zakinit* qu'une seule fois, on le place en dehors de toute définition d'instrument, dans l'entête de l'orchestre, après *sr*, *kr*, *ksmps*, et *nchnls*.



### Note

Les canaux zak se comptent à partir de 0, si bien que si l'on définit un canal, le seul canal valide est le canal 0.

## Exemples

Voici un exemple de l'opcode *zakinit*. Il utilise le fichier *zakinit.csd* [examples/zakinit.csd].

### Exemple 1161. Exemple de l'opcode *zakinit*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zakinit.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```



```

sr = 44100
ksmps = 4410
nchnls = 1

; Initialize the ZAK space.
; Create 3 a-rate variables and 5 k-rate variables.
zakinit 2, 3

instr 1 ;a simple waveform.
    ; Generate a simple sine waveform.
    asin oscil 20000, 440, 1

    ; Send the sine waveform to za variable #1.
    zaw asin, 1
endin

instr 2 ;generates audio output.
    ; Read za variable #1.
    a1 zar 1

    ; Generate audio output.
    out a1

    ; Clear the za variables, get them ready for
    ; another pass.
    zac1 0, 2
endin

instr 3 ;increments k-type channels
    k0 zkr 0
    k1 zkr 1
    k2 zkr 2

    zkw k0+1, 0
    zkw k1+5, 1
    zkw k2+10, 2
endin

instr 4 ;displays values from k-type channels
    k0 zkr 0
    k1 zkr 1
    k2 zkr 2

    ; The total count for k0 is 30, since there are 10
    ; control blocks per second and instruments 3 and 4
    ; are on for 3 seconds.
    printf "k0 = %i\n", k0, k0
    printf "k1 = %i\n", k1, k1
    printf "k2 = %i\n", k2, k2
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

i 1 0 1
i 2 0 1

i 3 0 3
i 4 0 3
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Robin Whittle

Australie

Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zamod

zamod — Module un signal de taux-a par un autre.

## Description

Module un signal de taux-a par un autre.

## Syntaxe

```
ares zamod asig, kzamod
```

## Exécution

*asig* -- Le signal d'entrée

*kzamod* -- Contrôle quelle variable *za* sera utilisée pour la modulation. Une valeur positive indique une modulation additive, une valeur négative indique une modulation multiplicative. Une valeur de 0 ne fait aucun changement à *asig*.

*zamod* Module un signal de taux-a par un autre, qui provient d'une variable *za*. La position de la variable modulante est contrôlée par la variable de taux-i ou de taux-k *kzamod*. Ceci est la version de taux-a de *zkmod*.

## Exemples

Voici un exemple de l'opcode *zamod*. Il utilise le fichier *zamod.csd* [examples/zamod.csd].

### Exemple 1162. Exemple de l'opcode *zamod*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac        -iadc      -d          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zamod.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 2 a-rate variables and 2 k-rate variables.
zakinit 2, 2

; Instrument #1 -- a simple waveform.
instr 1
```

```

; Vary an a-rate signal linearly from 20,000 to 0.
asig line 20000, p3, 0

; Send the signal to za variable #1.
zaw asig, 1
endin

; Instrument #2 -- generates audio output.
instr 2
; Generate a simple sine wave.
asin oscil 1, 440, 1

; Modify the sine wave, multiply its amplitude by
; za variable #1.
a1 zamod asin, -1

; Generate the audio output.
out a1

; Clear the za variables, prepare them for
; another pass.
zacl 0, 2
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
; Play Instrument #2 for 2 seconds.
i 2 0 2
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*zacr, ziw, ziwm*

## Crédits

Auteur : Robin Whittle  
 Australie  
 Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zar

zir — Lecture à partir d'une position dans l'espace za au taux-a.

## Description

Lecture à partir d'une position dans l'espace za au taux-a.

## Syntaxe

ares **zar** kndx

## Exécution

*kndx* -- pointe sur la position za à lire.

*zar* lit la suite de nombres décimaux à *kndx* dans l'espace za, qui sont les ksmps nombres décimaux de taux-a à traiter dans un cycle-k.

## Exemples

Voici un exemple du opcode zar. Il utilise le *zar.csd* [examples/zar.csd].

### Exemple 1163. Exemple de l'opcode zar.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zar.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
; Generate a simple sine waveform.
asin oscil 20000, 440, 1

; Send the sine waveform to za variable #1.
zaw asin, 1
endin
```

```

; Instrument #2 -- generates audio output.
instr 2
  ; Read za variable #1.
  a1 zar 1

  ; Generate audio output.
  out a1

  ; Clear the za variables, get them ready for
  ; another pass.
  zac1 0, 1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*zarg, zir, zkr*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zarg

**zarg** — Lecture à partir d'une position dans l'espace *za* au taux-*a* avec application d'un gain.

## Description

Lecture à partir d'une position dans l'espace *za* au taux-*a* avec application d'un gain.

## Syntaxe

ares **zarg** *kndx*, *kgain*

## Initialisation

*kndx* -- pointe sur la position *za* à lire.

*kgain* -- Multiplicateur pour le signal taux-*a*.

## Exécution

*zarg* lit la suite de nombres décimaux à *kndx* dans l'espace *za*, qui sont les *ksmps* nombres décimaux de taux-*a* à traiter dans un cycle-*k*. *zarg* multiplie aussi le signal de taux-*a* par la valeur de taux-*k* *kgain*.

## Exemples

Voici un exemple de l'opcode *zarg*. Il utilise le fichier *zarg.csd* [examples/zarg.csd].

### Exemple 1164. Exemple de l'opcode *zarg*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in   No messages
-odac         -iadc      -d          ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zarg.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
```

```

; Generate a simple sine waveform, with an amplitude
; between 0 and 1.
asin oscil 1, 440, 1

; Send the sine waveform to za variable #1.
zaw asin, 1
endin

; Instrument #2 -- generates audio output.
instr 2
; Read za variable #1, multiply its amplitude by 20,000.
a1 zarg 1, 20000

; Generate audio output.
out a1

; Clear the za variables, get them ready for
; another pass.
zACL 0, 1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e

</CsScore>
</CsSoundSynthesizer>

```

## Voir aussi

*zar, zir, zkr*

## Crédits

Auteur : Robin Whittle  
 Australie  
 Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.



# zaw

zaw — Écrit dans une variable *za* au taux-*a* sans mixage.

## Description

Écrit dans une variable *za* au taux-*a* sans mixage.

## Syntaxe

**zaw** *asig*, *kndx*

## Exécution

*asig* -- Valeur à écrire dans la position *za*.

*kndx* -- Pointe sur la position *za* vers laquelle écrire.

*zaw* écrit *asig* dans la variable *za* spécifiée par *kndx*.

Ces opcodes sont rapides, et vérifient toujours que l'indexation est à l'intérieur des limites des espaces *zk* ou *za*. Sinon, une erreur est rapportée, la valeur 0 est retournée, et il n'y a aucune écriture.

## Exemples

Voici un exemple de l'opcode *zaw*. Il utilise le fichier *zaw.csd* [examples/zaw.csd].

### Exemple 1165. Exemple de l'opcode *zaw*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zaw.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
; Generate a simple sine waveform.
```

```

    asin oscil 20000, 440, 1

    ; Send the sine waveform to za variable #1.
    zaw asin, 1
endin

; Instrument #2 -- generates audio output.
instr 2
    ; Read za variable #1.
    a1 zar 1

    ; Generate the audio output.
    out a1

    ; Clear the za variables, get them ready for
    ; another pass.
    zawl 0, 1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*zawm, ziw, ziwm, zkw, zkwm*

## Crédits

Auteur : Robin Whittle  
 Australie  
 Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zawm

zawm — Ecrit dans une variable za au taux-a avec mixage.

## Description

Ecrit dans une variable za au taux-a avec mixage.

## Syntaxe

```
zawm asig, kndx [, imix]
```

## Initialisation

*imix* (facultatif, par défaut=1) -- indique si le mixage sera fait.

## Exécution

*asig* -- valeur à écrire dans l'espace za.

*kndx* -- Pointe sur la position za vers laquelle écrire.

Ces opcodes sont rapides, et vérifient toujours que l'indexation est à l'intérieur des limites des espaces zk ou za. Sinon, une erreur est rapportée, la valeur 0 est retournée et il n'y a aucune écriture.

*zawm* est un opcode de mixage, il ajoute le signal à la valeur actuelle de la variable. Si aucun *imix* n'est spécifié, le mixage aura toujours lieu. *imix* = 0 provoquera l'écrasement des données comme dans *ziw*, *zkw*, et *zaw*. Toute autre valeur entraînera un mixage.

*Avertissement* : lors de l'utilisation des opcodes de mixage *ziwm*, *zkwm*, et *zawm*, il faut faire attention à ce que les variables qui reçoivent le mixage soient remises à zéro à la fin (ou au début) de chaque cycle-k ou -a. Leur ajouter indéfiniment des signaux peut engendrer des valeurs astronomiques.

Une approche possible serait d'établir certains intervalles de variables zk ou za à utiliser pour le mixage, puis d'utiliser ensuite *zkcl* ou *zacl* pour effacer ces variables.

## Exemples

Voici un exemple de l'opcode *zawm*. Il utilise le fichier *zawm.csd* [examples/zawm.csd].

### Exemple 1166. Exemple de l'opcode *zawm*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o zawm.wav -W ;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a basic instrument.
instr 1
; Generate a simple sine waveform.
asin oscil 15000, 440, 1

; Mix the sine waveform with za variable #1.
zawm asin, 1
endin

; Instrument #2 -- another basic instrument.
instr 2
; Generate another waveform with a different frequency.
asin oscil 15000, 880, 1

; Mix this sine waveform with za variable #1.
zawm asin, 1
endin

; Instrument #3 -- generates audio output.
instr 3
; Read za variable #1, containing both waveforms.
a1 zar 1

; Generate the audio output.
out a1

; Clear the za variables, get them ready for
; another pass.
zacr 0, 1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
; Play Instrument #3 for one second.
i 3 0 1
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*zaw, ziw, ziw, zkw, zkwm*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zdf\_1pole

zdf\_1pole — Implémentation d'un filtre à 1 pôle avec rétroaction sans retard.

## Description

Implémentation d'un filtre à 1 pôle avec rétroaction sans retard (6 dB/oct). Propose les modes passe-bas (par défaut), passe-haut et passe-tout.

## Syntaxe

```
asig zdf_1pole ain, xcf [, kmode, istor]
```

## Initialisation

*istor* -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal de sortie.

*asig* -- signal d'entrée.

*xcf* -- fréquence de coupure du filtre (taux-i-, k ou a).

*kmode* -- type de sortie du filtre. La valeur par défaut est 0. Le mode de sortie peut être :

- 0 - passe-bas
- 1 - passe-haut
- 2 - passe-tout

## Exemples

Voici un exemple de l'opcode `zdf_1pole`. Il utilise le fichier `zdf_1pole.csd` [examples/zdf\_1pole.csd].

### Exemple 1167. Exemple de l'opcode `zdf_1pole`.

```
<CsoundSynthesizer>
<CsOptions>
-o dac
</CsOptions>
<CsInstruments>

sr = 48000
ksmps = 1
nchnls = 2
0dbfs = 1
```

```
instr 1
  asig = random:a(-1.0, 1.0)
  asig = zdf_lpole(asig, line(220, p3, 10000), p4)
  outc(asig, asig)
endin

instr 2
  asig = vco2(0.5, 220)
  asig = zdf_lpole(asig, line(220, p3, 10000), p4)
  outc(asig, asig)
endin

</CsInstruments>
<CsScore>
i1 0 4 0
i1 5 4 1
i1 10 4 2
i2 15 4 0
i2 20 4 1
i2 25 4 2
</CsScore>
</CsoundSynthesizer>
```

## Références

Ce filtre est basé sur les travaux de Will Pirkle qui emploie le travail de Vadim Zavalishin pour créer des implémentations de filtres analogiques à transformation préservant la topologie (TPT), avec des transformations bilinéaires.

1. Pirkle, Will. Designing Software Synthesizer Plug-ins in C++: For RackAFX, VST3, and Audio Units. CRC Press, 2014.
2. Pirkle, Will. AN-4: Virtual Analog (VA) Filter Implementation. 2013.
3. Zavalishin, Vadim. "The Art of VA filter design." Native Instruments, 2012.

## Crédits

Auteur : Steven Yi  
Avril 2017

Nouveau dans Csound 6.09.0

# zdf\_1pole\_mode

zdf\_1pole\_mode — Implémentation d'un filtre à 1 pôle avec rétroaction sans retard et sortie multimodale.

## Description

Implémentation d'un filtre à 1 pôle avec rétroaction sans retard (6 dB/oct). Propose une sortie passe-bas et passe-haut.

## Syntaxe

```
alp, ahp zdf_1pole_mode ain, xcf [, istor]
```

## Initialisation

*istor* -- état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*alp* -- signal de sortie passe-bas.

*ahp* -- signal de sortie passe-haut.

*asig* -- signal d'entrée.

*xcf* -- fréquence de coupure du filtre (taux-i-, k ou a).

## Exemples

Voici un exemple de l'opcode `zdf_1pole_mode`. Il utilise le fichier `zdf_1pole_mode.csd` [exemples/zdf\_1pole\_mode.csd].

### Exemple 1168. Example of the `zdf_1pole_mode` opcode.

```
<CsoundSynthesizer>
<CsOptions>
-o dac
</CsOptions>
<CsInstruments>

sr = 48000
ksmps = 1
nchnls = 2
0dbfs = 1

instr 1
  asig = random:a(-1.0, 1.0)
  alp, ahp zdf_1pole_mode asig, line(220, p3, 10000)
  asig = (p4 == 0) ? alp : ahp
```



```
    outc(asig, asig)
endin

instr 2
    asig = vco2(0.5, 220)
    alp, ahp zdf_lpole_mode asig, line(220, p3, 10000)
    asig = (p4 == 0) ? alp : ahp
    outc(asig, asig)
endin

</CsInstruments>
<CsScore>
i1 0 4 0
i1 5 4 1
i2 10 4 0
i2 15 4 1
</CsScore>
</CsoundSynthesizer>
```

## Références

Ce filtre est basé sur les travaux de Will Pirkle qui emploie le travail de Vadim Zavalishin pour créer des implémentations de filtres analogiques à transformation préservant la topologie (TPT), avec des transformations bilinéaires.

1. Pirkle, Will. Designing Software Synthesizer Plug-ins in C++: For RackAFX, VST3, and Audio Units. CRC Press, 2014.
2. Pirkle, Will. AN-4: Virtual Analog (VA) Filter Implementation. 2013.
3. Zavalishin, Vadim. "The Art of VA filter design." Native Instruments, 2012.

## Crédits

Auteur : Steven Yi  
Avril 2017

Nouveau dans Csound 6.09.0

# zdf\_2pole

*zdf\_2pole* — Implémentation d'un filtre à 2 pôles avec rétroaction sans retard.

## Description

Implémentation d'un filtre à 2 pôles avec rétroaction sans retard (12 dB/oct). Propose les modes passe-bas (par défaut), passe-haut et passe-tout.

## Syntaxe

```
asig zdf_2pole ain, xcf, xQ [, kmode, istor]
```

## Initialisation

*istor* --état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal de sortie.

*ain* -- signal d'entrée.

*xcf* -- fréquence de coupure du filtre (taux-i-, k ou a).

*xQ* -- valeur Q du filtre (taux-i-, k ou a). Dans l'intervalle 0.5-25.0.

*kmode* -- type de sortie du filtre. La valeur par défaut est 0. Le mode de sortie peut être :

- 0 - passe-bas
- 1 - passe-haut
- 2 - passe-bande
- 3 - passe-bande à gain unitaire
- 4 - coupe-bande
- 5 - passe-tout
- 6 - écrêteur

## Exemples

Voici un exemple de l'opcode *zdf\_2pole*. Il utilise le fichier *zdf\_2pole.csd* [examples/zdf\_2pole.csd].

**Exemple 1169. Exemple de l'opcode *zdf\_2pole*.**

```
<CsoundSynthesizer>
<CsOptions>
-o dac
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 1
nchnls = 2
0dbfs = 1

instr 1
  asig = random:a(-1.0, 1.0)
  asig = zdf_2pole(asig, line(20, p3, 10000), 4, p4)

  outc(asig, asig)
endin

instr 2
  asig = vco2(0.5, 220)
  asig = zdf_2pole(asig, line(20, p3, 10000), 4, p4)
  outc(asig, asig)
endin

</CsInstruments>
<CsScore>
i1 0 4 0
i1 5 4 1
i1 10 4 2
i1 15 4 3
i1 20 4 4
i1 25 4 5
i1 30 4 6

i2 40 4 0
i2 45 4 1
i2 50 4 2
i2 55 4 3
i2 60 4 4
i2 65 4 5
i2 70 4 6

</CsScore>
</CsoundSynthesizer>
```

## Références

Ce filtre est basé sur les travaux de Will Pirkle qui emploie le travail de Vadim Zavalishin pour créer des implémentations de filtres analogiques à transformation préservant la topologie (TPT), avec des transformations bilinéaires.

1. Pirkle, Will. Designing Software Synthesizer Plug-ins in C++: For RackAFX, VST3, and Audio Units. CRC Press, 2014.
2. Pirkle, Will. AN-4: Virtual Analog (VA) Filter Implementation. 2013.
3. Zavalishin, Vadim. "The Art of VA filter design." Native Instruments, 2012.

## Crédits

Auteur : Steven Yi  
Avril 2017

Nouveau dans Csound 6.09.0

# zdf\_2pole\_mode

`zdf_2pole_mode` — Implémentation d'un filtre à 2 pôles avec rétroaction sans retard et sortie multimodale.

## Description

Implémentation d'un filtre à 2 pôles avec rétroaction sans retard (12 dB/oct). Propose une sortie passe-bas, passe-bande et passe-haut.

## Syntaxe

```
alp, abp, ahp zdf_2pole_mode ain, xcf, Q [, istor]
```

## Initialisation

*istor* --état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*alp* -- signal de sortie passe-bas.

*abp* -- signal de sortie passe-bande.

*ahp* -- signal de sortie passe-haut.

*ain* -- signal d'entrée.

*xcf* -- fréquence de coupure du filtre (taux-i-, k ou a).

*Q* -- valeur Q du filtre (taux-i-, k ou a). Dans l'intervalle 0.5-25.0.

## Exemples

Voici un exemple de l'opcode `zdf_2pole_mode`. Il utilise le fichier `zdf_2pole_mode.csd` [examples/zdf\_2pole\_mode.csd].

### Exemple 1170. Example of the `zdf_2pole_mode` opcode.

```
<CsoundSynthesizer>
<CsOptions>
-o dac
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 1
nchnls = 2
0dbfs = 1
```

```
instr 1
  asig = random:a(-1.0, 1.0)
  asigs[] init 3
  asigs[0], asigs[1], asigs[2] zdf_2pole_mode asig, line(20, p3, 10000), 4

  asig = asigs[p4]
  outc(asig, asig)
endin

instr 2
  asig = vco2(0.5, 220)
  asigs[] init 3
  asigs[0], asigs[1], asigs[2] zdf_2pole_mode asig, line(20, p3, 10000), 4
  outc(asigs[p4], asigs[p4])
endin

</CsInstruments>
<CsScore>
i1 0 4 0
i1 5 4 1
i1 10 4 2

i2 15 4 0
i2 20 4 1
i2 25 4 2

</CsScore>
</CsoundSynthesizer>
```

## Références

Ce filtre est basé sur les travaux de Will Pirkle qui emploie le travail de Vadim Zavalishin pour créer des implémentations de filtres analogiques à transformation préservant la topologie (TPT), avec des transformations bilinéaires.

1. Pirkle, Will. Designing Software Synthesizer Plug-ins in C++: For RackAFX, VST3, and Audio Units. CRC Press, 2014.
2. Pirkle, Will. AN-4: Virtual Analog (VA) Filter Implementation. 2013.
3. Zavalishin, Vadim. "The Art of VA filter design." Native Instruments, 2012.

## Crédits

Auteur : Steven Yi  
Avril 2017

Nouveau dans Csound 6.09.0

# zdf\_ladder

`zdf_ladder` — Implémentation d'un filtre en échelle à 4 pôles avec rétroaction sans retard.

## Description

Implémentation d'un filtre en échelle à 4 pôles avec rétroaction sans retard (24 dB/oct) basé sur le filtre en échelle de Moog.

## Syntaxe

```
asig zdf_ladder ain, xcf, xQ [, istor]
```

## Initialisation

*istor* --état initial de l'espace de données interne. Comme le filtrage comprend une boucle de rétroaction sur la sortie précédente, l'état initial de l'espace de stockage utilisé est significatif. Une valeur nulle provoquera l'effacement de cet espace ; une valeur non nulle autorisera la persistance de l'information précédente. La valeur par défaut est 0.

## Exécution

*asig* -- signal de sortie.

*asig* -- signal d'entrée.

*xcf* -- fréquence de coupure du filtre (taux-i-, k ou a).

*xQ* -- valeur Q du filtre (taux-i-, k ou a). Dans l'intervalle 0.5-25.0.

## Exemples

Voici un exemple de l'opcode `zdf_ladder`. Il utilise le fichier *zdf\_ladder.csd* [examples/zdf\_ladder.csd].

### Exemple 1171. Exemple de l'opcode `zdf_ladder`.

```
<CsoundSynthesizer>
<CsOptions>
-o dac
</CsOptions>
<CsInstruments>

sr = 48000
ksmps = 1
nchnls = 2
0dbfs = 1

instr 1
  asig = random:a(-1.0, 1.0)
  asig = zdf_ladder(asig, expon(220, p3, 5000), p4)
  outc(asig, asig)
endin
```

```
instr 2
  asig = vco2(0.5, 220)
  asig = zdf_ladder(asig, expon(220, p3, 5000), p4)
  outc(asig, asig)
endin

instr 3
  asig = vco2(0.5, p4)
  asig = zdf_ladder(asig, expon(5000, p3, 200), 0.5 + p5* 24)
  asig *= madsr(0.05, 0, 1, 0.25)
  outc(asig, asig)
endin

instr play_instr3
  schedule(3, 0, 0.25, mtof(48 + (p4 % 2) * 12), p4 / 16)

  if(p4 < 16) then
    schedule("play_instr3", 0.25, 0.25, p4 + 1)
  else
    event_i("e", 0.5, 0)
  endif
  turnoff
endin

</CsInstruments>
<CsScore>
i1 0 2 0.5
i1 + . 1
i1 + . 4
i1 + . 10
i1 + . 18
i1 + . 24.5
i2 12 2 0.5
i2 + . 1
i2 + . 4
i2 + . 10
i2 + . 18
i2 + . 24.5

s
i "play_instr3" 0 0.25 0
f0 60
</CsScore>
</CsoundSynthesizer>
```

## Références

Ce filtre est basé sur les travaux de Will Pirkle qui emploie le travail de Vadim Zavalishin pour créer des implémentations de filtres analogiques à transformation préservant la topologie (TPT), avec des transformations bilinéaires.

1. Pirkle, Will. Designing Software Synthesizer Plug-ins in C++: For RackAFX, VST3, and Audio Units. CRC Press, 2014.
2. Pirkle, Will. AN-4: Virtual Analog (VA) Filter Implementation. 2013.
3. Zavalishin, Vadim. "The Art of VA filter design." Native Instruments, 2012.

## Crédits

Auteur : Steven Yi



Avril 2017

Nouveau dans Csound 6.09.0

## zfilter2

*zfilter2* — Réalise un filtrage au moyen d'un bloc de filtre numérique de forme transposée II avec déplacement radial et déformation angulaire des pôles.

### Description

Filtre configurable à usage général avec contrôle variable des pôles. Les coefficients du filtre implémentent l'équation aux différences suivante :

$$(1)*y(n) = b0*x[n] + b1*x[n-1] + \dots + bM*x[n-M] - a1*y[n-1] - \dots - aN*y[n-N]$$

the system function for which is represented by:

$$H(Z) = \frac{B(Z)}{A(Z)} = \frac{b0 + b1*Z^{-1} + \dots + bM*Z^{-M}}{1 + a1*Z^{-1} + \dots + aN*Z^{-N}}$$

### Syntaxe

```
ares zfilter2 asig, kdamp, kfreq, iM, iN, ib0, ib1, ..., ibM, \  
      ia1, ia2, ..., iaN
```

### Initialisation

A l'initialisation, les nombres de zéros et de pôles du filtres sont spécifiés ainsi que leurs valeurs. Les coefficients doivent être obtenus par une application externe de conception de filtre telle que Matlab et sont spécifiés directement ou bien chargés dans une table via *GEN01*. Avec *zfilter2*, les racines du polynôme caractéristique sont calculées à l'initialisation pour une implémentation efficace des opérations de contrôle des pôles.

### Exécution

L'opcode *filter2* réalise un filtrage au moyen d'un bloc de filtre numérique de forme transposée II sans contrôle variable. *zfilter2* utilise en plus les opérations de déplacement radial et de déformation angulaire des pôles dans le plan des Z.

Le déplacement radial des pôles augmente la magnitude des pôles le long des lignes radiales dans le plan des Z. Cela modifie les durées de suroscillation du filtre. La variable de taux-k *kdamp* est le paramètre d'amortissement. Les valeurs positives (0.01 to 0.99) augmentent la durée de suroscillation du filtre (Q élevé), les valeurs négatives (-0.01 to -0.99) diminuent la durée de suroscillation du filtre (Q faible).

La déformation des pôles modifie leur fréquence en les déplaçant le long de chemins angulaires dans le plan des Z. Cette opération ne change pas la forme de l'amplitude de la réponse mais modifie les fréquences d'un facteur constant (préservant 0 et p). La variable de taux-k *kfreq* détermine le facteur de déformation fréquentielle. Les valeurs positives (0.01 to 0.99) augmentent les fréquences vers p et les valeurs négatives (-0.01 to -0.99) diminuent les fréquences vers 0.

Comme *filter2* implémente des filtres récursifs généralisés, on peut l'utiliser pour définir une grande variété d'algorithmes généraux de traitement numérique du signal. Par exemple, on peut implémenter un guide

d'onde numérique pour modéliser un instrument de musique au moyen d'une paire d'opcodes *delayr* et *delayw* conjointement à l'opcode *filter2*.

## Exemples

Un filtre RII du second ordre contrôlable opérant sur un signal de taux-a :

```
a1 zfilter2 asig, kdamp, kfreq, 1, 2, 1, ia1, ia2 ;; filtre RII contrôlable de taux-a
```

## Voir aussi

*filter2*

## Crédits

Auteur : Michael A. Casey  
M.I.T.  
Cambridge, Mass.  
1997

Nouveau dans la version 3.47

# zir

zir — Lecture à partir d'une position dans un espace zk au taux-i.

## Description

Lecture à partir d'une position dans un espace zk au taux-i.

## Syntaxe

```
ir zir indx
```

## Initialisation

*indx* -- pointe vers la position zk à lire.

## Exécution

*zir* lit le signal à la position *indx* dans l'espace zk.

## Exemples

Voici un exemple de l'opcode zir. Il utilise le fichier *zir.csd* [examples/zir.csd].

### Exemple 1172. Exemple de l'opcode zir.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac         -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zir.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple instrument.
instr 1
; Set the zk variable #1 to 32.594.
ziw 32.594, 1
endin
```

```
; Instrument #2 -- prints out zk variable #1.
instr 2
; Read the zk variable #1 at i-rate.
i1 zir 1

; Print out the value of zk variable #1.
print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*zar, zarg, zkr*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# ziw

ziw — Ecrit dans une variable zk au taux-i sans mixage.

## Description

Ecrit dans une variable zk au taux-i sans mixage.

## Syntaxe

```
ziw isig, indx
```

## Initialisation

*isig* -- Initialise la valeur de la position zk.

*indx* -- Pointe sur la position za vers laquelle écrire.

## Exécution

*ziw* écrit *isig* dans la variable zk spécifié par *indx*.

Ces opcodes sont rapides, et vérifient toujours que l'indexation est à l'intérieur des limites des espaces zk ou za. Sinon, une erreur est rapportée, la valeur 0 est retournée, et il n'y a aucune écriture.

## Exemples

Voici un exemple de l'opcode *ziw*. Il utilise le fichier *ziw.csd* [examples/ziw.csd].

### Exemple 1173. Exemple de l'opcode *ziw*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ziw.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1
```

```
; Instrument #1 -- a simple instrument.
instr 1
  ; Set zk variable #1 to 64.182.
  ziw 64.182, 1
endin

; Instrument #2 -- prints out zk variable #1.
instr 2
  ; Read zk variable #1 at i-rate.
  i1 zir 1

  ; Print out the value of zk variable #1.
  print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*zaw, zawm, ziwm, zkw, zkwm*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# ziwm

ziwm — Ecrit dans une variable zk au taux-i avec mixage.

## Description

Ecrit dans une variable zk au taux-i avec mixage.

## Syntaxe

```
ziwm isig, indx [, imix]
```

## Initialisation

*isig* -- initialise la valeur à la position zk.

*indx* -- pointe sur la position zk vers laquelle écrire.

*imix* (facultatif, par défaut=1) -- indique si le mixage doit avoir lieu.

## Exécution

*ziwm* est un opcode de mixage, il ajoute le signal à la valeur actuelle de la variable. Si aucun *imix* n'est spécifié, le mixage aura toujours lieu. *imix* = 0 provoquera l'écrasement des données comme dans *ziw*, *zkw* et *zaw*. Toute autre valeur entraînera un mixage.

*Attention* : lors de l'utilisation des opcodes de mixage *ziwm*, *zkwm* et *zawm*, il faut faire attention à ce que les variables qui reçoivent le mixage soient remises à zéro à la fin (ou au début) de chaque cycle-k ou -a. Leur ajouter indéfiniment des signaux peut engendrer des valeurs astronomiques.

Une approche serait d'établir certains intervalles de variables zk et za à utiliser pour le mixage, puis d'utiliser *zkcl* ou *zacl* pour effacer ces intervalles.

## Exemples

Voici un exemple de l'opcode *ziwm*. Il utilise le fichier *ziwm.csd* [examples/ziwm.csd].

### Exemple 1174. Exemple de l'opcode *ziwm*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o ziwm.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```



```

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple instrument.
instr 1
; Add 20.5 to zk variable #1.
ziwm 20.5, 1
endin

; Instrument #2 -- another simple instrument.
instr 2
; Add 15.25 to zk variable #1.
ziwm 15.25, 1
endin

; Instrument #3 -- prints out zk variable #1.
instr 3
; Read zk variable #1 at i-rate.
i1 zir 1

; Print out the value of zk variable #1.
; It should be 35.75 (20.5 + 15.25)
print i1
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
; Play Instrument #3 for one second.
i 3 0 1
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*zaw, zawm, ziw, zkw, zkwm*

## Crédits

Auteur : Robin Whittle  
 Australie  
 Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zkcl

zkcl — Efface une ou plusieurs variable dans l'espace zk.

## Description

Efface une ou plusieurs variable dans l'espace zk.

## Syntaxe

```
zkcl kfirst, klast
```

## Exécution

*kfirst* -- Première position zk de l'intervalle à effacer.

*klast* -- Dernière position zk de l'intervalle à effacer.

*zkcl* efface une ou plusieurs variables dans l'espace zk. Ceci est utile pour les variables utilisées comme accumulateur pour mélanger des signaux de taux-k à chaque cycle, mais qui doivent être effacés avant le prochain groupe de calculs.

## Exemples

Voici un exemple de l'opcode zkcl. Il utilise le fichier *zkcl.csd* [examples/zkcl.csd].

### Exemple 1175. Exemple de l'opcode zkcl.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zkcl.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
; Linearly vary a k-rate signal from 220 to 1760.
kline line 220, p3, 1760
```

```

; Add the linear signal to zk variable #1.
zkw kline, 1
endin

; Instrument #2 -- generates audio output.
instr 2
; Read zk variable #1.
kfreq zkr 1

; Use the value of zk variable #1 to vary
; the frequency of a sine waveform.
a1 oscil 20000, kfreq, 1

; Generate the audio output.
out a1

; Clear the zk variables, get them ready for
; another pass.
zkcl 0, 1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for three seconds.
i 1 0 3
; Play Instrument #2 for three seconds.
i 2 0 3
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*zacl, zkwm, zkw, zkmod, zkr*

## Crédits

Auteur : Robin Whittle  
 Australie  
 Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zkmod

zkmod — Facilite la modulation d'un signal par un autre.

## Description

Facilite la modulation d'un signal par un autre.

## Syntaxe

```
kres zkmod ksig, kzkmod
```

## Exécution

*k*sig -- Le signal d'entrée

*kzkmod* -- contrôle quelle variable zk est utilisée pour la modulation. Une valeur positive signifie une modulation additive, une valeur négative une modulation multiplicative. La valeur 0 ne fait aucun changement à *k*sig. *kzkmod* peut être de taux-i ou de taux-k.

*zkmod* Facilite la modulation d'un signal par un autre, le signal de modulation provenant d'une variable zk. La modulation spécifiée peut être additive ou multiplicative.

## Exemples

Voici un exemple de l'opcode zkmod. Il utilise le fichier *zkmod.csd* [examples/zkmod.csd].

### Exemple 1176. Exemple de l'opcode zkmod.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zkmod.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

; Initialize the ZAK space.
; Create 2 a-rate variables and 2 k-rate variables.
zakinit 2, 2

; Instrument #1 -- a signal with jitter.
instr 1
; Generate a k-rate signal goes from 30 to 2,000.
```

```

kline line 30, p3, 2000

; Add the signal into zk variable #1.
zkw kline, 1
endin

; Instrument #2 -- generates audio output.
instr 2
; Create a k-rate signal modulated the jitter opcode.
kamp init 20
kcpsmin init 40
kcpsmax init 60
kjtr jitter kamp, kcpsmin, kcpsmax

; Get the frequency values from zk variable #1.
kfreq zkr 1
; Add the the frequency values in zk variable #1 to
; the jitter signal.
kjfreq zkmod kjtr, 1

; Use a simple sine waveform for the left speaker.
aleft oscil 20000, kfreq, 1
; Use a sine waveform with jitter for the right speaker.
aright oscil 20000, kjfreq, 1

; Generate the audio output.
outs aleft, aright

; Clear the zk variables, prepare them for
; another pass.
zkcl 0, 2
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 2 seconds.
i 1 0 2
; Play Instrument #2 for 2 seconds.
i 2 0 2
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*zamod, zkcl, zkr, zkwm, zkw*

## Crédits

Auteur : Robin Whittle  
 Australie  
 Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zkr

zkr — Lecture à partir d'une position dans l'espace zk au taux-k.

## Description

Lecture à partir d'une position dans l'espace zk au taux-k.

## Syntaxe

```
kres zkr kndx
```

## Initialisation

*kndx* -- pointe sur la position za à lire.

## Exécution

*zkr* lit la suite de nombres décimaux à *kndx* dans l'espace zk.

## Exemples

Voici un exemple de l'opcode zkr. Il utilise le fichier *zkr.csd* [examples/zkr.csd].

### Exemple 1177. Exemple de l'opcode zkr.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zkr.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
; Linearly vary a k-rate signal from 440 to 880.
kline line 440, p3, 880

; Add the linear signal to zk variable #1.
```

```

    zkw kline, 1
    endin

; Instrument #2 -- generates audio output.
instr 2
    ; Read zk variable #1.
    kfreq zkr 1

    ; Use the value of zk variable #1 to vary
    ; the frequency of a sine waveform.
    a1 oscil 20000, kfreq, 1

    ; Generate the audio output.
    out a1

    ; Clear the zk variables, get them ready for
    ; another pass.
    zkcl 0, 1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for one second.
i 1 0 1
; Play Instrument #2 for one second.
i 2 0 1
e

</CsScore>
</CsoundSynthesizer>

```

## Voir aussi

*zar, zarg, zir, zkcl, zkmod, zkwm, zkw*

## Crédits

Auteur : Robin Whittle  
 Australie  
 Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zkw

zkw — Ecrit dans une variable zk au taux-k sans mixage.

## Description

Ecrit dans une variable zk au taux-k sans mixage.

## Syntaxe

```
zkw kval, kndx
```

## Exécution

*kval* -- valeur à écrire dans la position zk.

*kndx* -- pointe sur la position zk vers laquelle écrire.

*zkw* écrit *kval* dans la variable zk spécifiée par *kndx*.

## Exemples

Voici un exemple de l'opcode zkw. Il utilise le fichier *zkw.csd* [examples/zkw.csd].

### Exemple 1178. Exemple de l'opcode zkw.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o zkw.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a simple waveform.
instr 1
; Linearly vary a k-rate signal from 100 to 1,000.
kline line 100, p3, 1000

; Add the linear signal to zk variable #1.
zkw kline, 1
```



```
endin

; Instrument #2 -- generates audio output.
instr 2
  ; Read zk variable #1.
  kfreq zkr 1

  ; Use the value of zk variable #1 to vary
  ; the frequency of a sine waveform.
  a1 oscil 20000, kfreq, 1

  ; Generate the audio output.
  out a1

  ; Clear the zk variables, get them ready for
  ; another pass.
  zkcl 0, 1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for two seconds.
i 1 0 2
; Play Instrument #2 for two seconds.
i 2 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*zaw, zawm, ziw, ziwm, zkr, zkwm*

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

# zkwm

zkwm — Ecrit dans une variable zk au taux-k avec mixage.

## Description

Ecrit dans une variable zk au taux-k avec mixage.

## Syntaxe

```
zkwm ksig, kndx [, imix]
```

## Initialisation

*imix* (facultatif) -- indique si le mixage sera fait.

## Exécution

*ksig* -- valeur à écrire dans l'espace zk.

*kndx* -- pointe sur la position zk vers laquelle écrire.

*zkwm* est un opcode de mixage, il ajoute le signal à la valeur courante de la variable. Si aucun *imix* n'est spécifié, le mixage aura toujours lieu. *imix* = 0 provoquera l'écrasement des données comme dans *ziw*, *zkw*, et *zaw*. Toutes autres valeurs entraînera un mixage.

*Avertissement* : lors de l'utilisation des opcodes de mixage *ziwm*, *zkwm*, et *zawm*, il faut faire attention à ce que les variables qui reçoivent le mixage soient remises à zéro à la fin (ou au début) de chaque cycle-k ou -a. Leur ajouter indéfiniment des signaux peut engendrer des valeurs astronomiques.

Une approche possible serait d'établir certains intervalles de variables zk ou za à utiliser pour le mixage, puis d'utiliser ensuite *zkcl* ou *zacl* pour effacer ces variables.

## Exemples

Voici un exemple de l'opcode *zkwm*. Il utilise le fichier *zkwm.csd* [examples/zkwm.csd].

### Exemple 1179. Exemple de l'opcode *zkwm*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o zkwm.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
```

```

ksmps = 10
nchnls = 1

; Initialize the ZAK space.
; Create 1 a-rate variable and 1 k-rate variable.
zakinit 1, 1

; Instrument #1 -- a basic instrument.
instr 1
    ; Generate a k-rate signal.
    ; The signal goes from 30 to 20,000 then back to 30.
    kramp linseg 30, p3/2, 20000, p3/2, 30

    ; Mix the signal into the zk variable #1.
    zkwm kramp, 1
endin

; Instrument #2 -- another basic instrument.
instr 2
    ; Generate another k-rate signal.
    ; This is a low frequency oscillator.
    klfo lfo 3500, 2

    ; Mix this signal into the zk variable #1.
    zkwm klfo, 1
endin

; Instrument #3 -- generates audio output.
instr 3
    ; Read zk variable #1, containing a mix of both signals.
    kamp zkr 1

    ; Create a sine waveform. Its amplitude will vary
    ; according to the values in zk variable #1.
    a1 oscil kamp, 880, 1

    ; Generate the audio output.
    out a1

    ; Clear the zk variable, get it ready for
    ; another pass.
    zkcl 0, 1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; Play Instrument #1 for 5 seconds.
i 1 0 5
; Play Instrument #2 for 5 seconds.
i 2 0 5
; Play Instrument #3 for 5 seconds.
i 3 0 5
e

</CsScore>
</CsSoundSynthesizer>

```

## Voir aussi

zaw, zawm, ziw, ziwm, zkcl, zkw, zkr

## Crédits

Auteur : Robin Whittle  
Australie  
Mai 1997

Nouveau dans la version 3.45

Exemple écrit par Kevin Conder.

---

# Instructions de partition et routines GEN

## Instructions de partition

Les instructions utilisées dans les partitions sont :

- *a* - Avance le temps de la partition d'une quantité spécifiée
- *b* - Réinitialise l'horloge
- *C* - Contrôle le report automatique des p-args
- *d* - Efface un instrument infini
- *e* - Marque la fin de la dernière section de la partition
- *f* - Appelle une *routine GEN* pour placer des valeurs dans une table de fonction stockée
- *i* - Active un instrument à une date spécifique et pour une certaine durée
- *m* - Positionne une marque nommée dans la partition
- *n* - Répète une section marquée
- *q* - Rend un instrument silencieux
- *r* - Commence une section répétée
- *s* - Marque la fin d'une section
- *t* - Fixe le tempo
- *v* - Permet une modification temporelle variable localement des événements de la partition
- *x* - Ignore le reste de la section courante
- *y* - Fixe la "graine" pour les nombres aléatoires, soit la valeur de p1, soit la valeur de l'horloge système si p1 est omis.
- *{* - Commence une boucle imbriquable, sans section
- *}* - Termine une boucle imbriquable, sans section

# Instruction a (ou instruction avancer)

a — Avancer le temps de la partition de la quantité spécifiée.

## Description

Provoque l'avancement du temps de la partition de la quantité spécifiée sans produire d'échantillons sonores.

## Syntaxe

a p1 p2 p3

## Exécution

p1 Non significatif. Habituellement zéro.  
p2 Date en pulsations à laquelle l'avance doit commencer.  
p3 Nombre de pulsations duquel il faut avancer sans produire de son.  
p4 |  
p5 | Non significatifs.  
p6 |  
.  
.

## Considérations Spéciales

Cette instruction permet d'avancer le compteur de pulsations dans une partition sans générer les échantillons sonores correspondants. On peut l'utiliser quand une section de la partition est incomplète (le début ou le milieu sont manquants) et que l'on ne souhaite pas générer et écouter une longue période de silence.

p2, date d'activation, et p3, nombre de pulsations, sont traités comme dans l'*instruction i*, en tenant compte du tri et des modifications par les *instructions t*.

Une *instruction a* sera insérée temporairement dans la partition par la fonction Score Extract lorsque l'extrait commence après le début de la Section. Ceci afin de conserver le compte de pulsations de la partition originale pour les messages de pic d'amplitude qui sont rapportés sur la console de l'utilisateur.

A chaque exécution d'un orchestre lorsqu'une *instruction a* est rencontrée, sa présence et son effet sont rapportés sur la console de l'utilisateur.

## Exemples

Voici un exemple de l'instruction a. Il utilise le fichier *a.csd* [examples/a.csd].

### Exemple 1180. Exemple de l'instruction a.

```
<CsoundSynthesizer>  
<CsOptions>
```

```

; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o a.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

aenv expseg .01, p3*0.25, 1, p3*0.75, 0.01
asig poscil3 .8*aenv, p4
outs asig, asig

endin
</CsInstruments>
<CsScore>
;two sections
s
a 0 0 6 ;advance score 6 seconds
i 1 0 2 110 ;these first 2 notes
i 1 3 2 220 ;will not sound
i 1 6 2 440
i 1 9 2 880
s
a 0 3 6 ;advance score 6 seconds, but do this after 3 seconds
i 1 0 2 110 ;this will sound, because action time (p2) from a statement = 3
i 1 3 2 220 ;so these 2 notes
i 1 6 2 440 ;will not sound
i 1 9 2 880 ;and this one will
e
</CsScore>
</CsoundSynthesizer>

```

# Instruction b

b — Cette instruction réinitialise l'horloge.

## Description

Cette instruction réinitialise l'horloge.

## Syntaxe

b p1

## Exécution

p1 -- Spécifie comment l'horloge doit être réglée.

## Considérations Spéciales

p1 est le nombre de pulsations par lequel les valeurs p2 des *instructions i* suivantes sont modifiées. Si p1 est positif, l'horloge est avancée, et les notes suivantes apparaissent plus tard, le nombre de pulsations spécifié par p1 étant ajouté au p2 des notes. Si p1 est négatif, l'horloge est retardée, et les notes suivantes apparaissent plus tôt, le nombre de pulsations spécifié par p1 étant soustrait du p2 des notes. L'effet n'est pas cumulatif. L'horloge est réinitialisée avec chaque *instruction b*. Si p1 = 0, l'horloge revient à sa position initiale, et les notes suivantes apparaissent à leur position spécifiée en p2.

## Exemples

Voici un exemple de l'instruction b. Il utilise le fichier *b.csd* [examples/b.csd].

### Exemple 1181. Exemple de l'instruction b.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o b.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

aenv expseg .01, p3*0.25, 1, p3*0.75, 0.01
asig poscil3 .4*aenv, 220, 1
outs asig, asig

endin
```



```
instr 2

asig pluck 0.7, p4, 220, 0, 1
outs asig, asig

endin

instr 3

asig loscil .8, 1, 2, 1
outs asig, asig

endin

instr 4
asig bamboo .8, 0.01
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave
f 2 0 0 1 "fox.wav" 0 0 0 ;sample

i1 0 2
i1 10 2

b 5 ; set the clock "forward"
i2 1 2 220 ; start time = 6
i2 2 2 110 ; start time = 7

b -1
i3 3 2 ; start time = 2
i3 5.5 1 ; start time = 4.5

b 0 ; reset clock to normal
i4 10 2 ; start time = 10

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Explication suggérée et exemple fourni par Paul Winkler. (Version 4.07 de Csound)

# Instruction C

C Statement — Cette instruction active ou désactive la fonction automatique de report des arguments (carry).

## Description

Cette instruction active ou désactive la fonction automatique de report (carry) des arguments manquants.

## Syntaxe

`C p1`

## Exécution

*p1* -- Spécifie s'il faut désactiver (valeur 0) ou activer (valeur non nulle).

## Exemples

Voici un exemple de l'instruction C. Il utilise le fichier *C.csd* [examples/b.csd].

### Exemple 1182. Exemple de l'instruction C.

```
<CsoundSynthesizer>

<CsInstruments>
0dbfs = 1

instr 1
ii pcount
print ii
a1 oscil 0.5,A4
outs a1,a1
endin
</CsInstruments>

<CsScore>

i1 0 .5 100
i . +
i . . . !
i

s
i1 0 .5 100
i . + .
C 0
i . . .
i . .
i .

e
</CsScore>
```

</CsoundSynthesizer>

## Crédits

Auteur : John ffitch  
Alta Sounds  
Bath, UK  
2016

Nouveau dans la version 6.08

# Instruction d (instruction de note)

d — Supprime un instrument actif à une date précise.

## Description

Cette instruction provoque la suppression d'un instrument tenu à une date spécifique.

## Syntaxe

i p1 p2 p3 p4

## Initialisation

*p1* -- Numéro d'instrument. Une partie décimale facultative permet d'ajouter une étiquette indiquant des liaisons entre des notes particulières d'aggrégats consécutifs

*p2* -- Date de début en unités arbitraires appelées pulsations.

*p3* -- Ignoré mais nécessaire. Habituellement zéro.

## Exécution

Une pulsation vaut une seconde, à moins qu'il n'y ait une *instruction t* dans cette section de la partition ou une *option -t* dans la ligne de commande.

Les dates d'action sont relatives au début d'une section (voir l'*instruction s*), qui reçoit la date 0.

Dans une section, les instructions de note peuvent être placées dans n'importe quel ordre. Avant d'être envoyées à l'orchestre, les instructions non triées de la partition doivent être traitées par la fonction Sort, qui les ordonnera par valeurs de *p2* croissantes. Les notes ayant la même valeur en *p2* seront triées par *p1* croissants ; si elles ont le même *p1*, alors par *p3* croissants.

## Exemples

Voici un exemple de l'instruction d. Il utilise le fichier *d\_statement.csd* [examples/d\_statement.csd].

### Exemple 1183. Exemple de l'instruction d.

```
<CsoundSynthesizer>

<CsInstruments>
instr 1
  a1 oscil 10000, 440
  out a1
endin
instr sound
  a1 oscil 10000, 440
  out a1
endin
</CsInstruments>
```

```
<CsScore>
f 0 10
i 1 0 -1
d 1 1 0

i "sound" 3 -1
i "-sound" 4 0
e
</CsScore>

</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch, nouveau dans la version 6.09.

# Instruction e

e — On peut utiliser cette instruction pour marquer la fin de la dernière section de la partition.

## Description

On peut utiliser cette instruction pour marquer la fin de la dernière section de la partition.

## Syntaxe

e [0 temps]

## Exécution

Le second p-champ *temps* est facultatif et s'il est présent, il détermine la date de fin (en pulsations) de la dernière section de la partition. Cette date doit être après le dernier événement sinon elle n'aura pas d'effet. Les instruments "actifs en permanence" se termineront à cette date. Cette manière d'allonger la section est utile pour éviter les coupures prématurées de chute de réverbération ou d'autres effets.

## Considérations Spéciales

L'instruction *e* est contextuellement identique à une instruction *s*. De plus, l'instruction *e* termine toute génération de signal (y compris une exécution indéfinie) et ferme tous les fichiers d'entrée et de sortie.

Si une instruction *e* intervient avant la fin de la partition, toutes les lignes suivantes de la partition seront ignorées.

Dans un fichier de partition pas encore trié, l'instruction *e* est facultative. Si un fichier de partition n'a pas d'instruction *e*, alors la fonction Sort en fournira une.

## Exemples

Voici un exemple de l'instruction *e*. Il utilise le fichier *e.csd* [examples/e.csd].

### Exemple 1184. Exemple de l'instruction e.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime output leave only the line below:
; -o e.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gal init 0
```

```

instr 1

aenv expseg .01, p3*0.1, 1, p3*0.9, 0.01
gal poscil3 .5*aenv, cpspch(p4), 1
outs gal,gal
endin

instr 99

aL, aR reverbsc gal, gal, 0.85, 12000, sr, 0.5, 1
outs aL,aR

gal = 0

endin
</CsInstruments>
<CsScore>
f 1 0 128 10 1 ;sine wave

i 1 1 0.1 8.00
i 1 2 0.1 8.02
i 1 3 0.1 8.04
i 1 4 0.1 9.06

i 99 0 6 ;remains active for 6 seconds

e10
</CsScore>
</CsoundSynthesizer>

```

# Instruction f (ou instruction de table de fonction)

f — Provoque l'écriture de valeurs dans une table de fonction en mémoire par une routine GEN.

## Description

Provoque l'écriture de valeurs dans une table de fonction en mémoire par une routine GEN pour utilisation par des instruments.

## Syntaxe

**f** p1 p2 p3 p4 p5 ... PMAX

## Exécution

p1 -- Numéro de table sous lequel la fonction mémorisée sera connue. Un nombre négatif signifie une demande de destruction de la table.

p2 -- Date d'activation de la génération de la fonction (ou de sa destruction) en pulsations.

p3 -- Taille de la table de la fonction (c'est-à-dire nombre de points). Doit être une puissance de 2, ou une puissance de 2 plus 1 si ce nombre est positif. La taille de table maximale est de 16777216 ( $2^{24}$ ) points.

p4 -- Numéro de la routine GEN à appeler (voir *ROUTINES GEN*). Une valeur négative supprimera la normalisation.

p5 ... PMAX -- Paramètres dont la signification est déterminée par la routine GEN particulière.

## Considérations spéciales

Les tables de fonction sont des tableaux de valeurs en virgule flottante. On peut créer une simple onde sinusoïdale avec cette ligne :

```
f 1 0 1024 10 1
```

Cette table utilise *GEN10* pour son remplissage.

Historiquement, à cause des contraintes des anciennes plates-formes, Csound ne pouvait accepter que des tables dont la taille était une puissance de deux. Cette limitation a été levée dans les récentes versions, et l'on peut créer des tables de n'importe quelle taille. Cependant, pour créer une table dont la taille n'est pas une puissance de deux (ou une puissance de deux plus un), il faut spécifier la taille comme un nombre négatif.



### Note

Il y a des opcodes qui n'accepteront pas des tables dont la taille n'est pas une puissance de deux, car ils comptent sur cela pour leur optimisation interne.

Pour les tableaux dont la longueur est une puissance de 2, l'allocation d'espace mémoire est toujours prévue pour  $2^n$  points plus un *point de garde*. La valeur du point de garde, utilisée pour la lecture avec interpolation, peut être fixée automatiquement selon le but de la table : si la *taille* est une puissance de 2 exacte, le point



de garde sera une copie du premier point ; cela convient pour la *lecture cyclique avec interpolation* comme dans *oscili*, etc., et devrait même être utilisé pour la version sans interpolation *oscl* pour rester consistant. Si la *taille* est fixée à  $2^n + 1$ , le point de garde prolongera automatiquement le contour des valeurs de la table ; cela convient pour les fonctions à lecture non-cyclique comme dans *envplx*, *oscill*, *oscilli*, etc.

La taille de la table est utilisée comme un code pour indiquer à Csound comment remplir ce point de garde. Si la taille est exactement une puissance de deux, alors le point de garde contient une copie du premier point de la table. Si la taille est une puissance-de-deux plus un, Csound étend le contour de la fonction stockée dans la table pour un point supplémentaire.

Les tables sont allouées dans la mémoire primaire, avec les données d'instrument. Le nombre maximum de tables était limité à 200. Ceci a changé et il n'est plus limité que par la quantité de mémoire disponible. (Actuellement il y a une limitation logicielle de 300, qui est augmentée automatiquement selon les besoins).

On peut supprimer une table de fonction existante par une *instruction f* contenant un p1 négatif et une date d'activation adéquate. Une table de fonction est également supprimée par la génération d'une autre table avec le même p1. Les fonctions ne sont pas automatiquement effacées à la fin d'une section de partition.

La date p2 est traitée de la même manière que dans l'*instruction i* en tenant compte du tri et des modifications par les *instructions t*. Si une *instruction f* et une *instruction i* ont le même p2, le tri donnera la priorité à l'*instruction f* afin que le table de fonction soit disponible pendant l'initialisation de la note.



## Avertissement

Le nombre maximum de p-champs acceptés dans la partition est déterminé par PMAX (une variable de compilation). PMAX vaut actuellement 1000. Cela peut éliminer des valeurs entrées au moyen de *GEN02*. Pour contourner cette limitation, utiliser *GEN23* ou *GEN28* pour lire les valeurs à partir d'un fichier.

On peut utiliser une *instruction f 0* (avec zéro en p1 et p2 positif) pour créer une date sans action associée. De tels marqueurs temporels sont utiles pour remplir une section de partition (voir l'*instruction s*) et pour lancer une exécution de Csound à partir d'événements en temps réel (par exemple en n'utilisant que des entrées MIDI sans événements de partition). La durée indique le nombre de secondes de l'exécution de Csound. Si l'on veut que Csound tourne pendant 10 heures, on utilisera :

```
f0 36000
```

La manière la plus simple de remplir une table (f1) avec des 0 est :

```
f1 0 xx 2 0
```

where xx = table size.

La manière la plus simple de remplir une table (f1) avec n'importe quelle valeur unique est :

```
f1 0 xx -7 yy xx yy
```

où xx = taille de la table et yy = n'importe quelle valeur unique

Dans les deux exemple ci-dessus, la taille de la table (p3) doit être une puissance de 2 ou une puissance-de-2 + 1.

## Voir aussi

*ROUTINES GEN*

## Crédits

Mise à jour en août 2002 grâce à une note de Rasmus Ekman. Il n'y a plus de limite codée en dur à 200 tables de fonction.

# Instruction i (instruction d'instrument ou de note)

i — Active un instrument à une date précise et pour une certaine durée.

## Description

Cette instruction est nécessaire pour activer un instrument à une date précise et pour une certaine durée. Les valeurs des champs de paramètre sont passées à cet instrument avant son initialisation, et demeurent valides durant toute son exécution.

## Syntaxe

i p1 p2 p3 p4 ...

## Initialisation

*p1* -- Numéro d'instrument, habituellement un nombre entier non négatif. Une partie décimale facultative permet d'ajouter une étiquette indiquant des liaisons entre des notes particulières d'aggrégats consécutifs. Un *p1* négatif (incluant une étiquette) peut être utilisé pour faire cesser une note « tenue » particulière. Dans le cas d'un instrument nommé, on peut obtenir le même effet en ayant un '-' comme premier caractère du nom.

*p2* -- Date de début en unités arbitraires appelées pulsations.

*p3* -- Durée en pulsations (habituellement positive). Une valeur négative démarre une note tenue (voir aussi *ihold*). On peut aussi utiliser une valeur négative pour les instruments 'toujours actifs' comme la réverbération. Ces notes ne sont pas terminées par des *instruction s*. Une valeur nulle provoquera une passe d'initialisation sans exécution (voir aussi *instr*).

*p4* ... -- Paramètres dont la signification est déterminée par l'instrument.

## Exécution

Une pulsation vaut une seconde, à moins qu'il n'y ait une *instruction t* dans cette section de la partition ou une *option -t* dans la ligne de commande.

Les dates de début ou d'action sont relatives au début d'une section (voir l'*instruction s*), qui reçoit la date 0.

Dans une section, les instructions de note peuvent être placées dans n'importe quel ordre. Avant d'être envoyées à l'orchestre, les instructions non triées de la partition doivent être traitées par la fonction Sort, qui les ordonnera par valeurs de *p2* croissantes. Les notes ayant la même valeur en *p2* seront triées par *p1* croissants ; si elles ont le même *p1*, alors par *p3* croissants.

Les notes peuvent être superposées, c'est-à-dire qu'un seul instrument peut jouer n'importe quel nombre de notes simultanément. (Les copies nécessaires de l'espace de données de l'instrument seront allouées dynamiquement par le chargeur de l'orchestre). Chaque note se termine normalement à la fin de sa durée en *p3*, ou à la réception d'un signal MIDI *noteoff*. Un instrument peut modifier sa propre durée en changeant la valeur de son *p3* pendant l'initialisation de la note, ou en se prolongeant lui-même par l'action d'une unité *linenr* ou *xtratim*.

Un instrument peut être activé et réglé pour une durée indéfinie soit en lui donnant un p3 négatif soit en incluant un *ihold* dans le code de son temps-i. Si une note tenue est active, une *instruction i* avec un p1 correspondant ne provoquera pas une nouvelle allocation mais prendra l'espace de données de la note tenue. Les nouveaux p-champs (y compris p3) seront maintenant effectifs, et une passe de temps-i sera exécutée pendant laquelle les unités peuvent être soit initialisées à nouveau soit autorisées à continuer comme requis pour une note liée (voir *tigoto*). Une note tenue peut être suivie soit par une autre note tenue soit par une note de durée finie. Une note tenue continuera à être jouée au-delà des fins de section (voir l'*instruction s*). Elle est arrêtée seulement par un *turnoff* ou par une *instruction i* avec un p1 négatif correspondant ou par une *instruction e*.

Il est possible d'avoir plusieurs instances (habituellement, mais pas forcément, des notes de hauteurs différentes) du même instrument, tenues simultanément, via des valeurs négatives de p3. L'instrument peut ensuite recevoir de nouveaux paramètres de la partition. C'est utile pour éviter de longs *linseg* codés en dur, et peut être accompli en ajoutant une partie décimale au numéro de l'instrument.

Par exemple, pour tenir trois copies de l'instrument 10 dans un accord :

```
i10.1  0  -1  7.00
i10.2  0  -1  7.04
i10.3  0  -1  7.07
```

Les instructions *i* suivantes peuvent faire référence aux mêmes instances de note active, et si la définition de l'instrument est faite proprement, les nouveaux p-champs peuvent servir à changer le caractère des notes jouées. Par exemple, pour faire glisser l'accord précédent d'une octave vers le haut et le laisser résonner :

```
i10.1  1  1  8.00
i10.2  1  1  8.04
i10.3  1  1  8.07
```



## Astuce

Pour la terminaison des notes, il faut tenir compte du fait que  $i\ 1.1 == i\ 1.10$  et que  $i\ 1.1 ! = i\ 1.01$ . Le nombre maximum de positions décimales que l'on peut utiliser dépend de la précision avec laquelle Csound a été compilé (Csound Double (64 bit) ou Float (32 bit))

La définition de l'instrument doit prendre ceci en compte, cependant, spécialement si l'on veut éviter les clics (voir l'exemple ci-dessous).

Noter que la notation décimale du numéro d'instrument ne peut pas être utilisée en conjonction avec le MIDI en temps réel. Dans ce cas, l'instrument serait monodique tant qu'une note est tenue.

Les notes liées à des instances précédentes du même instrument, devraient éviter la plus grande partie de l'initialisation au moyen de *tigoto*, sauf pour les valeurs entrées dans la partition. Par exemple, tous les opcodes de lecture de table dans l'instrument, seront habituellement sautés en initialisation, car ils mémorisent en interne leur phase. Si celle-ci est brutalement modifiée, on entendra des clics en sortie.

Noter que plusieurs opcodes (comme *delay* et *reverb*) sont prévus pour une initialisation facultative. Pour utiliser cette possibilité, l'opcode *tival* est approprié. Ainsi, il n'y a pas besoin de les escamoter par un saut *tigoto*.

A partir de la version 3.53 de Csound, les chaînes sont reconnues dans les p-champs des opcodes qui les acceptent (*convolve*, *adsyn*, *diskin*, etc.). Il ne peut y avoir qu'une seule chaîne par ligne de la partition.

On peut aussi terminer une note depuis la partition en utilisant un nombre négatif pour l'instrument (p1). Cela équivaut à utiliser l'opcode *turnoff2*. Lorsqu'une note est terminée depuis la partition, elle peut avoir

un relâchement (si *xtratim* ou des opcodes avec une section de relâchement tels que *linenr* sont utilisés) et seules les notes ayant la même partie fractionnaire sont arrêtées. De plus, seule la dernière instance de l'instrument est arrêtée, si bien qu'il faut autant de numéros d'instrument négatifs que de numéros positifs pour que toutes les notes soient arrêtées.

```
i 1.1 1 300 8.00
i 1.2 1 300 8.04
i 1.3 1 -300 8.07
i 1.3 1 -300 8.09

; noter que les p-champs suivant p2 sont ignorés
; si le numéro d'instrument est négatif
i -1.1 3 1 4.00
i -1.2 4 51 4.04
i -1.3 5 1 4.07
i -1.3 6 10 4.09
```

## Considérations Spéciales

Le numéro d'instrument maximum était de 200. Cela a changé et il n'est plus limité que par la capacité mémoire (actuellement, il y a une limite logicielle de 200 ; celle-ci est étendue automatiquement si nécessaire).

## Exemples

Voici un exemple de l'instruction i. Il utilise le fichier *i\_statement.csd* [examples/i\_statement.csd].

### Exemple 1185. Exemple de l'instruction i.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o i_statement.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 10

icps      init      cpspch(p4)      ; Get target pitch from score event
iporime   init      abs(p3)/7      ; Portamento time dep on note length
iamp0     init      p5              ; Set default amps
iamp1     init      p5
iamp2     init      p5

itie      tival
if itie == 1      igoto nofadein      ; Check if this note is tied,
; if not fade in
iamp0     init      0

nofadein:
if p3 < 0      igoto nofadeout      ; Check if this note is held, if not fade out
iamp2     init      0

nofadeout:
```

```

; Now do amp from the set values:
kamp      linseg      iamp0, .03, iamp1, abs(p3)-.03, iamp2

; Skip rest of initialization on tied note:
          tigoto      tieskip

kcps      init        icps                ; Init pitch for untied note
kcps      port        icps, iportime, icps  ; Drift towards target pitch

kpw       oscil       .4, rnd(1), 1, rnd(.7) ; A simple triangle-saw oscil
ar        vco         kamp, kcps, 3, kpw+.5, 1, 1/icps

; (Used in testing - one may set ipch to cpspch(p4+2)
;          and view output spectrum)
;          ar oscil kamp, kcps, 1

          outs        ar, ar

tieskip:; Skip some initialization on tied note

endin
</CsInstruments>
<CsScore>
f1  0 8192 10 1          ; Sine

i10.1    0    -1    7.00    .15
i10.2    0    -1    7.04
i10.3    0    -1    7.07
i10.1    1    -1    8.00
i10.2    1    -1    8.04
i10.3    1    -1    8.07
i10.1    2     1    7.11
i10.2    2     1    8.04
i10.3    2     1    8.07
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Texte supplémentaire (Version 4.07 de Csound) expliquant les notes liées, publié par Rasmus Ekman d'après une note de David Kirsh, postée sur la liste de courrier électronique de Csound. Instrument en exemple par Rasmus Ekman.

Mise à jour Août 2002 grâce à une note de Rasmus Ekman. Il n'y a plus de limite codée en dur à 200 instruments.

# Instruction m (instruction de marquage)

m — Positionne une marque nommée dans la partition.

## Description

Positionne une marque nommée dans la partition, qui peut être utilisée par une *instruction n*.

## Syntaxe

m p1

## Initialisation

p1 -- Nom de la marque.

## Exécution

Peut être utile pour construire une structure couplet refrain dans la partition. Les noms peuvent contenir des lettres et des chiffres.

Par exemple, la partition suivante :

```
m foo
il 0 1
il 1 1.5
il 2.5 2
s
il 0 10
s
n foo
e
```

sera passée au préprocesseur de Csound comme :

```
il 0 1
il 1 1.5
il 2.5 2
s
il 0 10
s
;; ceci est la section nommée répétée
il 0 1
il 1 1.5
il 2.5 2
s
;; fin de la section nommée
e
```

## Exemples

Voici un exemple de l'instruction m. Il utilise le fichier *m.csd* [examples/m.csd].

**Exemple 1186. Exemple de l'instruction m.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o m.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

aenv expseg .01, p3*0.25, 1, p3*0.75, 0.01
asig poscil3 .8*aenv, p4, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave

m foo ;mark section
i 1 0 1 110
i 1 1.5 1 220
i 1 3 1 440
i 1 4.5 1 880
s ;second section
i 1 0 2 110
i 1 2 2 220
s
n foo ;repeat marked section
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Avril 1998

Nouveau dans la version 3.48 de Csound



# Instruction n

n — Répète une section.

## Description

Répète une section depuis l'*instruction m* référencée.

## Syntaxe

n p1

## Initialisation

p1 -- Nom de la marque à répéter.

## Exécution

Peut-être utile pour construire une structure couplet refrain dans la partition. Les noms peuvent contenir des lettres et des chiffres.

Par exemple, la partition suivante :

```
m foo
il 0 1
il 1 1.5
il 2.5 2
s
il 0 10
s
n foo
e
```

Sera transmise par le préprocesseur à Csound comme :

```
il 0 1
il 1 1.5
il 2.5 2
s
il 0 10
s
;; ceci est la section nommée répétée
il 0 1
il 1 1.5
il 2.5 2
s
;; fin de la section nommée
e
```

## Exemples

Voir l'exemple pour l'*instruction m*.

## Crédits

Auteur : John ffitch  
Université de Bath/Codemist Ltd.  
Bath, UK  
Avril 1998

Nouveau dans la version 3.48 de Csound

# Instruction q

q — Cette instruction peut être utilisée pour rendre un instrument silencieux.

## Description

Cette instruction peut être utilisée pour rendre un instrument silencieux.

## Syntaxe

q p1 p2 p3

## Exécution

p1 -- Numéro de l'instrument à rendre muet/sonore.

p2 -- Date d'action en pulsations.

p3 -- Détermine si l'instrument doit être rendu silencieux ou sonore. La valeur 0 signifie silencieux, toute autre valeur signifie sonore.

Noter que ceci n'affecte pas les instruments déjà actifs à la date p2. Ça bloque toute tentative d'en démarrer un après cette date.

## Exemples

Voici un exemple de l'instruction q. Il utilise le fichier *q.csd* [examples/q.csd].

### Exemple 1187. Exemple de l'instruction q.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime output leave only the line below:
; -o q.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

aenv expseg .01, p3*0.25, 1, p3*0.75, 0.01
asig poscil3 .8*aenv, p4, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1;sine wave
```

```
s
q 1 6 0 ;mute at 6 seconds in this section
i 1 0 2 110
i 1 3 2 220
i 1 6 2 440
i 1 9 2 880

s
q 1 6 1 ;unmute again at 6 seconds in this section
i 1 0 2 110
i 1 3 2 220
i 1 6 2 440
i 1 9 2 880
e
</CsScore>
</CsoundSynthesizer>
```

# Instruction r (instruction répéter)

r — Débute une section répétée.

## Description

Débute une section répétée, qui dure jusqu'à la prochaine instruction *s*, *r* ou *e*.

## Syntaxe

**r** p1 p2

## Initialisation

*p1* -- Nombre de répétitions de la section demandé.

*p2* -- Macro(nom) pour indexer chaque répétition (facultatif).

## Exécution

Afin de rendre les sections plus souples qu'une simple édition, la macro nommée en *p2* reçoit la valeur 1 à la première boucle dans la section, 2 à la seconde, 3 à la troisième, etc. On peut l'utiliser pour changer la valeur des p-champs, ou l'ignorer.



### Avertissement

A cause de sérieux problèmes d'interaction avec l'expansion de macro, les sections doivent commencer et finir dans le même fichier, à l'extérieur de toute macro.

## Exemples

Voici un exemple d'instruction r. Il utilise le fichier *r.csd* [examples/r.csd].

### Exemple 1188. Exemple d'instruction r.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o r.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

; Instrument #1.
```

```
instr 1
; The score's p4 parameter has the number of repeats.
kreps = p4
; The score's p5 parameter has our note's frequency.
kcps = p5

; Print the number of repeats.
printks "Repeated %i time(s).\n", 1, kreps

; Generate a nice beep.
a1 oscil 20000, kcps, 1
out a1
endin

</CsInstruments>
<CsScore>

; Table #1, a sine wave.
f 1 0 16384 10 1

; We'll repeat this section 6 times. Each time it
; is repeated, its macro REPS_MACRO is incremented.
r6 REPS_MACRO

; Play Instrument #1.
; p4 = the r statement's macro, REPS_MACRO.
; p5 = the frequency in cycles per second.
i 1 00.10 00.10 $REPS_MACRO 1760
i 1 00.30 00.10 $REPS_MACRO 880
i 1 00.50 00.10 $REPS_MACRO 440
i 1 00.70 00.10 $REPS_MACRO 220

; Marks the end of the section.
s

e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK  
Avril 1998

Nouveau dans la version 3.48 de Csound

Exemple écrit par Kevin Conder

# Instruction s

s — Marque le fin d'une section.

## Description

L'*instruction s* marque le fin d'une section.

## Syntaxe

**s** [ temps ]

## Initialisation

Le premier p-champ *temps* est facultatif et s'il est présent, il détermine la date de fin (en pulsations) de la section. Cette date doit être après la fin du dernier évènement de la section sinon elle n'aura pas d'effet. On peut l'utiliser pour créer une pause avant le début de la section suivante ou pour permettre aux instruments "actifs en permanence" tels que les effets de jouer seuls pendant une certaine durée.

## Exécution

Le tri des *instructions i*, des *instructions f* et des *instructions a* par date d'action est effectué section par section.

La modification temporelle par l'*instruction t* est faite section par section.

Toutes les dates d'action à l'intérieur d'une section sont relatives à son début. Une instruction de section établit un nouveau temps relatif de 0, mais n'a pas d'autres effets de réinitialisation (par exemple les tables de fonction mémorisées sont préservées par delà les limites de section).

On considère qu'une section est complète lorsque toutes les dates d'action et toutes les durées finies ont été satisfaites. (C'est-à-dire que la "longueur" d'une section est déterminée par la dernière action apparue ou par l'arrêt du système). Une section peut être allongée par l'utilisation d'une *instruction f0* ou en fournissant la valeur de *p1* facultative à l'*instruction s*.

À la fin d'une section, le système provoque automatiquement le nettoyage des instruments inactifs et de leur espace de données.



### Note

- Puisque les instructions de partition sont traitées section par section, la quantité de mémoire requise dépend du nombre maximum d'instructions de partition dans une section. L'allocation de mémoire est dynamique, et l'utilisateur sera informé chaque fois que des blocs de mémoire supplémentaires sont demandés pendant le traitement de la partition.
- Pour la dernière section d'une partition, l'*instruction s* est facultative ; l'*instruction e* peut être utilisée à la place.

## Exemples

Voici un exemple de l'*instruction s*. Il utilise le fichier *s.csd* [examples/s.csd].

### Exemple 1189. Exemple de l'instruction s.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o s.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

aenv expseg .01, p3*0.25, 1, p3*0.75, 0.01
asig poscil3 .8*aenv, p4, 1
      outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1;sine wave

s ;first section
i 1 0 2 110
i 1 3 2 220
i 1 6 2 440
i 1 9 2 880

s ;second section
i 1 0 2 880
i 1 3 2 440
i 1 6 2 220
i 1 9 2 110
e
</CsScore>
</CsoundSynthesizer>
```



# Instruction t (instruction de tempo)

t — Fixe le tempo.

## Description

Cette instruction fixe le tempo et spécifie les accelerando et les ritardando de la section courante. Ceci est réalisé en convertissant les pulsations en secondes.

## Syntaxe

t p1 p2 p3 p4 ... (illimité)

## Initialisation

p1 -- Doit être zéro.

p2 -- Tempo initial en pulsations par minute.

p3, p5, p7,... -- Dates en pulsations (en ordre non décroissant).

p4, p6, p8,... -- Tempi pour les dates en pulsations référencées.

## Exécution

Les dates et le Tempo pour chaque date sont donnés en couples ordonnés qui définissent des points sur un graphe « date, tempo ». (L'axe du temps est ici en pulsations et n'est donc pas nécessairement linéaire). Le taux de pulsations d'une section peut être pensé comme un mouvement d'un point à un autre de ce graphe : un mouvement entre deux points à la même hauteur signifie un tempo constant, tandis qu'un mouvement entre deux points de hauteurs différentes traduit un accelerando ou un ritardando selon le cas. Le graphe peut contenir des discontinuités : deux points ayant la même date mais des tempi différents provoqueront un changement de tempo instantané.

Le mouvement entre différents tempi sur des durées non nulles est inversement linéaire. Cela veut dire qu'un accelerando entre deux tempi M1 et M2 procède par interpolation linéaire des durées de chaque pulsation entre 60/M1 et 60/M2.

Le premier tempo doit être donné pour la pulsation 0.

Une fois assigné, un tempo sera effectif à partir de cette date à moins d'être influencé par un tempo suivant, ainsi, le dernier tempo spécifié sera actif jusqu'à la fin de la section.

Une *instruction t* ne s'applique que dans la section dans laquelle elle apparaît. Une seule *instruction t* est pertinente dans une section ; elle peut être placée n'importe où dans la section. Si une section de partition ne contient pas d'*instruction t*, les pulsations sont alors interprétées comme des secondes (c'est-à-dire avec une *instruction t 0 60* implicite).

Nota Bene. Si la commande de Csound comprend une *option -t*, le tempo interprété de toutes les *instruction t* de la partition sera remplacé par le tempo de la ligne de commande.

## Exemples

Voici un exemple de l'instruction t. Il utilise le fichier *t.csd* [examples/t.csd].

### Exemple 1190. Exemple de l'instruction t.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o t.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

aenv expseg .01, p3*0.25, 1, p3*0.75, 0.01
asig poscil3 .8*aenv, p4, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave

t 0 240 12 30 15 240 ;start tempo = 240

i 1 0 2 110 ;tempo = 240
i 1 3 2 220 ;slow down &
i 1 6 2 440 ;slow down &
i 1 9 2 880 ;slow down &
i 1 12 2 110 ;slow down to 30 at 12 seconds
i 1 15 2 220 ;speed up to 240 again
i 1 18 2 440 ;stay at tempo 240
i 1 21 2 880
e
</CsScore>
</CsoundSynthesizer>
```

# Instruction v

v — Permet une modification temporelle variable localement des évènements de la partition.

## Description

L'*instruction v* permet une modification temporelle variable localement des évènements de la partition.

## Syntaxe

v p1

## Initialisation

p1 -- facteur de modification temporelle (doit être positif).

## Exécution

L'*instruction v* prend effet avec l'*instruction i* qui la suit, et reste effective jusqu'à la prochaine *instruction v*, *instruction s*, ou *instruction e*.

## Exemples

La valeur de p1 est utilisée comme un coefficient multiplicatif de la date de début (p2) des *instructions i* suivantes.

```
i1  0 1  ; note1
v2
i1  1 1  ; note2
```

Dans cet exemple, la deuxième note apparaît deux pulsations après la première note, et elle est deux fois plus longue.

Bien que l'*instruction v* soit semblable à l'*instruction t*, l'*instruction v* agit localement. Cela veut dire que v n'affecte que les notes suivantes, et que son effet peut être annulé ou changé par une autre *instruction v*.

Les valeurs reportées ne sont pas affectées par l'*instruction v* (voir *Carry*).

```
i1  0 1
v2
i.  + .
i.  . .
```

Dans cet exemple, l'*instruction v* n'a aucun effet.

Voici un exemple de l'*instruction v*. Il utilise le fichier *v.csd* [examples/v.csd].

### Exemple 1191. Exemple de l'*instruction v*.

<CsoundSynthesizer>

```

<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o v.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

aenv expseg .01, p3*0.25, 1, p3*0.75, 0.01
asig poscil3 .4*aenv, p4, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave
;because note 3 and 5 are played simultaneously and are nearly of the same frequency,
;played together they will create a "beating" sound.

i 1 0 2 110 ; note1
v2
i 1 3 . 220 ; note2
i 1 6 . 110 ; note3
v1
i 1 9 . 880 ; note4
i 1 12 . 100 ; note5
e
</CsScore>
</CsSoundSynthesizer>

```

Dans cet exemple, note3 et note5 sont jouées simultanément, tandis que note4 est jouée avant note3, c'est-à-dire à sa place initiale. Les durées sont inchangées.

# Instruction x

x — Ignore le reste de la section courante.

## Description

On peut utiliser cette instruction pour ignorer le reste de la section courante.

## Syntaxe

**x** valeurbidon

## Initialisation

Tous les p-champs sont ignorés.

## Exemples

Voici un exemple de l'instruction x. Il utilise le fichier *x.csd* [examples/x.csd].

### Exemple 1192. Exemple de l'instruction x.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o x.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

aenv expseg .01, p3*0.25, 1, p3*0.75, 0.01
asig poscil3 .8*aenv, p4, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave

s ;first section
i 1 0 2 110
i 1 3 2 220
i 1 6 2 440
i 1 9 2 880
s ;second section
x ;skip the rest
i 1 0 2 110 ;of this section
```

```
i 1 3 2 220
i 1 6 2 440
i 1 9 2 880
s ;but continue with this one
i 1 0 2 880
i 1 3 2 440
i 1 6 2 220
i 1 9 2 110
e
</CsScore>
</CsoundSynthesizer>
```

# Instruction y (ou instruction graine)

a — Fixe la "graine" pour les nombres aléatoires.

## Description

Fixe la "graine" pour les nombres aléatoires, soit la valeur de p1, soit la valeur de l'horloge système si p1 est omis.

## Syntaxe

**y** [p1]

## Initialisation

*p1* -- Un entier compris entre 0 et  $2^{32}$  utilisé comme nouvelle graine pour les nombres aléatoires. S'il est omis, la valeur de l'horloge système est utilisée.

## Exécution

On peut utiliser le symbole tilde ~ dans une expression chaque fois qu'un nombre est permis. Chaque ~ sera remplacé par un nombre aléatoire compris entre zéro (0) et un (1). S'il n'y a pas d'instruction y dans la partition, le générateur de nombres pseudo-aléatoires fournira les mêmes nombres à chaque exécution. Si une graine fixe est donnée, une suite prévisible de nombres pseudo-aléatoires sera générée à partir de cette graine. S'il y a une instruction y sans p1, l'horloge système sera utilisée comme graine, fournissant une suite de nombres pseudo-aléatoires différente à chaque exécution.

## Exemples

Voici un exemple de l'instruction y. Il utilise le fichier *y\_statement.csd* [examples/y\_statement.csd].

### Exemple 1193. Exemple de l'instruction y.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o 0dbfs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1
  print p4
endin
</CsInstruments>
```

```
<CsScore>
```

```
y
```

```
i 1 0 1 [~ * 10 + 1]
```

```
i 1 + 1 [~ * 10 + 1]
```

```
i 1 + 1 [~ * 10 + 1]
```

```
i 1 + 1 [~ * 10 + 1]
```

```
e
```

```
</CsScore>
```

```
</CsoundSynthesizer>
```

A chaque exécution de cette exemple, différentes valeurs comprises entre 1 et 10 seront affichées. La graine utilisée est également affichée.

## Crédits

Auteur : John ffitch, 2014

Nouveau dans la version 6.03



# Instruction {

{ — Commence une boucle imbriquable, sans section.

## Description

On peut utiliser les *instructions { et }* pour répéter un groupe d'instructions de partition. Ces boucles ne constituent pas des sections de partition indépendantes et peuvent ainsi répéter des événements dans la même section. Plusieurs boucles peuvent se chevaucher dans le temps ou être imbriquées.

## Syntaxe

{ p1 p2

## Initialisation

p1 -- Nombre de répétitions de la boucle.

p2 -- Un nom de macro qui est automatiquement défini au début de la boucle et dont la valeur est incrémentée à chaque répétition (facultatif). La valeur initiale est zéro et la valeur finale est (p1 - 1).

## Exécution

L'*instruction {* est utilisée conjointement avec l'*instruction }* pour définir des groupes d'événements de partition qui se répètent. Une boucle de partition commence par l'*instruction {* qui définit le nombre de répétitions et un nom de macro unique qui contiendra le compteur de boucle. Le corps d'une boucle peut contenir n'importe quel nombre d'événements (y compris des sauts de section) et il se termine par une *instruction }* ayant sa propre ligne. L'*instruction }* ne prend pas de paramètre.

Le terme "boucle" n'implique aucune sorte de succession temporelle pour les itérations de la boucle. Autrement dit, les valeurs p2 des événements à l'intérieur de la boucle ne sont pas incrémentées automatiquement de la longueur de la boucle à chaque répétition. C'est un avantage car cela permet de définir facilement des groupes d'événements simultanés. La macro de boucle peut être utilisée avec des *expressions de partition* pour incrémenter les dates de début d'événements ou pour faire varier les événements de toute autre manière désirée à chaque répétition. Noter que à la différence de l'*instruction r*, la valeur de la macro au premier passage dans la boucle est zéro (0), pas un (1). Ainsi la valeur finale est inférieure d'une unité au nombre de répétitions.

Les boucles de partition sont un outil très puissant. Bien que semblables à l'outil de répétition de section (l'*instruction r*), leur principal avantage est que les événements de partition dans les itérations successives de la boucle ne sont pas séparés par une fin de section. Ainsi, il est possible de créer plusieurs boucles qui se chevauchent dans le temps. Les boucles peuvent aussi être imbriquées jusqu'à une profondeur de 39 niveaux.



### Avertissement

En raison de sérieux problèmes d'interaction avec l'expansion de macro, les boucles doivent commencer et se terminer dans le même fichier, et pas à l'intérieur d'une macro.

## Exemples

Voici quelques exemples des *instructions { et }*.

### Exemple 1194. Répétition séquentielle d'une phrase de trois notes, quatre fois.

```
{ 4 CNT
i1 [0.00 + 0.75 * $CNT.] 0.2 220
i1 [0.25 + 0.75 * $CNT.] . 440
i1 [0.50 + 0.75 * $CNT.] . 880
}
```

interprété comme

```
i1 0.00 0.2 220
i1 0.25 . 440
i1 0.50 . 880

i1 0.75 0.2 220
i1 1.00 . 440
i1 1.25 . 880

i1 1.50 0.2 220
i1 1.75 . 440
i1 2.00 . 880

i1 2.25 0.2 220
i1 2.50 . 440
i1 2.75 . 880
```

### Exemple 1195. Création d'un groupe d'harmoniques simultanés.

Dans cet exemple,  $p4$  contient la fréquence de la note.

```
{ 8 PARTIAL
i1 0 1 [100 * ($PARTIAL. + 1)]
}
```

interprété comme

```
i1 0 1 100
i1 0 1 200
i1 0 1 300
i1 0 1 400
i1 0 1 500
i1 0 1 600
i1 0 1 700
i1 0 1 800
```

Voici un exemple complet des *instructions* { et }. Il utilise le fichier *leftbrace.csd* [examples/leftbrace.csd].

### Exemple 1196. Un exemple de boucles imbriquées pour créer plusieurs clusters inharmoniques de sinus.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in
```

```

-odac          -iadc      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o abs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
nchnls = 2

gaReverbSend init 0

; a simple sine wave partial
instr 1
    idur  =      p3
    iamp  =      p4
    ifreq =      p5
    aenv  linseg 0.0, 0.1*idur, iamp, 0.6*idur, iamp, 0.3*idur, 0.0
    aosc  oscili aenv, ifreq, 1
           vincr  gaReverbSend, aosc
endin

; global reverb instrument
instr 2
    al, ar reverb gaReverbSend, gaReverbSend, 0.85, 12000
           outs   gaReverbSend+al, gaReverbSend+ar
           clear  gaReverbSend
endin

</CsInstruments>
<CsScore>
f1 0 4096 10 1

{ 4 CNT
  { 8 PARTIAL
    ; start time      duration      amplitude      frequency

    i1 [0.5 * $CNT.] [1 + ($CNT * 0.2)] [500 + (~ * 200)] [800 + (200 * $CNT.) + ($PARTIAL. * 20,
  }
}

i2 0 6
e

</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 3.52 (?) de Csound. (Fixé dans la version 5.08).

# Instruction }

} — Termine une boucle imbriquable, sans section.

## Description

On peut utiliser les *instructions { et }* pour répéter un groupe d'instructions de partition. Ces boucles ne constituent pas des sections de partition indépendantes et peuvent ainsi répéter des événements dans la même section. Plusieurs boucles peuvent se chevaucher dans le temps ou être imbriquées.

## Syntaxe

}

## Initialisation

Tous les p-champs sont ignorés.

## Exécution

L'*instruction }* est utilisée conjointement avec l'*instruction {* pour définir des groupes d'événements de partition qui se répètent. Une boucle de partition commence par l'*instruction {* qui définit le nombre de répétitions et un nom de macro unique qui contiendra le compteur de boucle. Le corps d'une boucle peut contenir n'importe quel nombre d'événements (y compris des sauts de section) et il se termine par une *instruction }* ayant sa propre ligne. L'*instruction }* ne prend pas de paramètre.

Voir la documentation de l'*instruction {* pour plus de détails.

## Exemples

Voir les exemples de l'article sur l'*instruction {*.

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 3.52 (?) de Csound. (Fixé dans la version 5.08).

## Routines GEN

Les routines GEN sont utilisées comme générateurs de données pour les tables de fonction. Quand une table de fonction est créée au moyen de l'*instruction de partition f* la fonction GEN est donnée dans le quatrième argument. Un numéro de GEN négatif implique que la fonction ne sera pas normalisée et qu'elle gardera ses valeurs originales.

## Générateurs Sinus/Cosinus :

- *GEN09* - Formes d'ondes complexes obtenues par une somme pondérée de sinus.
- *GEN10* - Formes d'ondes complexes obtenues par une somme pondérée de sinus.

- *GEN11* - Ensemble additif de partiels cosinus.
- *GEN19* - Formes d'ondes complexes obtenues par une somme pondérée de sinus.
- *GEN30* - Génère des partiels harmoniques en analysant une table existante.
- *GEN33* - Génère des formes d'onde complexes en mélangeant des sinus.
- *GEN34* - Génère des formes d'onde complexes en mélangeant des sinus.

## Générateurs par morceaux de ligne/exponentielle

- *GEN05* - Construit des fonctions à partir de morceaux de courbes exponentielles.
- *GEN06* - Génère une fonction composée de morceaux de polynômes cubiques.
- *GEN07* - Construit des fonctions à partir de morceaux de lignes droites.
- *GEN08* - Génère une courbe spline cubique par morceaux.
- *GEN16* - Crée une table depuis une valeur initiale jusqu'à une valeur terminale.
- *GEN25* - Construit des fonctions à partir de morceaux de courbes exponentielles avec des points charnière (breakpoints).
- *GEN27* - Construit des fonctions à partir de morceaux de lignes droites avec des points charnière.

## Routines GEN d'accès fichier :

- *GEN01* - Transfère des données d'un fichier son dans une table de fonction.
- *GEN23* - Lit des valeurs numériques à partir d'un fichier texte.
- *GEN28* - Lit un fichier texte qui contient une trajectoire paramétrée par le temps.
- *GEN43* - Charge un fichier PVOCEX contenant une analyse de PV.
- *GEN49* - Transfère les données d'un fichier son MP3 dans une table de fonction.

## Routines GEN d'accès à des valeurs numériques

- *GEN02* - Transfère les données des p-champs dans une table de fonction.
- *GEN17* - Crée une fonction en escalier à partir des paires x-y données.
- *GEN52* - Crée une table multi-canaux entrelacés à partir des tables source indiquées, dans le format attendu par l'opcode *fconv*.

## Routines GEN de fonction fenêtre

- *GEN20* - Génère les fonctions de différentes fenêtres.

## Routines GEN de fonction aléatoire

- *GEN21* - Génère les tables de différentes distributions aléatoires.

- *GEN40* - Génère une distribution aléatoire à partir d'un histogramme.
- *GEN41* - Génère une liste aléatoire de paires numériques.
- *GEN42* - Génère une distribution aléatoire d'intervalles discrets de valeurs.

## Routines GEN de distorsion non-linéaire

- *GEN03* - Génère une table de fonction en évaluant un polynôme.
- *GEN13* - Mémoire un polynôme dont les coefficients sont dérivés des polynômes de Tchebychev de première espèce.
- *GEN14* - Mémoire un polynôme dont les coefficients sont dérivés des polynômes de Tchebychev de seconde espèce.
- *GEN15* - Crée deux tables de fonctions polynomiales mémorisées.

## Routines GEN de dimensionnement de l'amplitude

- *GEN04* - Génère une fonction de normalisation.
- *GEN12* - Génère le logarithme d'une fonction de Bessel de seconde espèce modifiée.
- *GEN24* - Lit les valeurs numériques d'une table de fonction déjà allouée en les reproportionnant.

## Routines GEN de mixage

- *GEN18* - Écrit des formes d'onde complexes construites à partir de formes d'ondes déjà existantes.
- *GEN31* - Mélange n'importe quelle forme d'onde définie dans une table existante.
- *GEN32* - Mélange n'importe quelle forme d'onde, rééchantillonnée soit par TFR soit par interpolation linéaire.

## Routines GEN de hauteur et d'accordage

- *GEN51* - Remplit une table avec une échelle micro-tonale entièrement personnalisée, à la manière des opcodes *cpstun*, *cpstuni* et *cpstmid*.

## Routines GEN nommées

On peut ajouter des routines GEN à Csound au moyen de greffons de fonction GEN. Il y a actuellement un seul greffon GEN qui fournit les fonctions exponentielle et tangente hyperbolique, ainsi que la fonction de sonie. Il y a aussi un générateur appelé *farey* pour les opérations sur les suites de Farey et un générateur de courbes de Bézier. Ces fonctions GEN ne sont pas appelées par un numéro, mais par un nom.

- *"tanh"* - remplit une table à partir d'une formule de tangente hyperbolique.
- *"exp"* - remplit une table à partir d'une formule d'exponentielle.
- *"sone"* - remplit une table à partir d'une formule de sonie.
- *"farey"* - remplit une table à partir d'une suite de Farey.

- *"quadbezier"* - remplit une table avec une courbe de Bézier quadratique.
- *"wave"* - remplit une table avec une transformée en ondelettes.
- *"padsynth"* - remplit une table au moyen de l'algorithme padsynth.

# GEN01

GEN01 — Transfère des données d'un fichier son dans une table de fonction.

## Description

Ce sous-programme transfère des données d'un fichier son dans une table de fonction.

## Syntaxe

```
f# date taille 1 codfic decal format canal
```

## Exécution

*taille* -- nombre de points dans la table. Ordinairement une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*) ; la taille de table maximale est de 16777216 ( $2^{24}$ ) points. L'allocation de mémoire pour la table peut être *différée* en mettant ce paramètre à 0 ; la taille allouée est alors le nombre de points dans le fichier (probablement pas une puissance de 2), et la table n'est pas utilisable par les oscillateurs normaux, mais par l'unité *loscil*. Le fichier son peut aussi être mono ou stéréo.

*codfic* -- entier ou chaîne de caractères dénotant le nom du fichier son source. Un entier dénote le fichier *soundin.codfic* ; une chaîne de caractères (entre apostrophes doubles, espaces autorisés) donne le nom du fichier lui-même, optionnellement un nom de chemin complet. Si le chemin n'est pas complet, le fichier est d'abord cherché dans le répertoire courant, ensuite dans celui qui est donné par la variable d'environnement *SSDIR* (si elle est définie) enfin par *SFDIR*. Voir aussi *soundin*.

*decal* -- commence à lire à *decal* secondes dans le fichier.

*canal* -- numéro du canal à lire. 0 indique de lire tous les canaux.

*format* -- s'il est positif, il est ignoré, mais s'il est négatif, il spécifie le format audio d'un fichier brut :

- 1 - 8-bit caractères signés
- 2 - 8-bit octets A-law
- 3 - 8-bit octets U-law
- 4 - 16-bit entiers courts
- 5 - 32-bit entiers longs
- 6 - 32-bit flottants
- 7 - 8-bit caractères non signés
- 8 - 24-bit entiers
- 9 - 64-bit doubles



### Note

Cette liste n'est pas la même que celle qui est dans *diskin2*.

Si *format* = 0 le format des échantillons est lu dans l'en-tête du fichier son.



### Note

- La lecture s'arrête à la fin du fichier ou lorsque la table est pleine. Les cellules de la table non remplies contiendront des zéros.



- Si p4 est positif, la table sera post-normalisée (reproportionnée avec une valeur absolue maximale de 1 après génération). Une valeur de p4 négative empêche cette opération.
- GEN01 fonctionne aussi avec les formats WAV et OGG ainsi que d'autres formats ; ceux-ci dépendent de libsndfile. Voir <http://www.mega-nerd.com/libsndfile/>

## Exemples

Voici un exemple de la routine GEN01. Il utilise le fichier *gen01.csd* [examples/gen01.csd] et plusieurs fichiers son.

### Exemple 1197. Un exemple de la routine GEN01.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen01.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;plays deferred and non-deferred sounds with loscil

ifn = p4
ibas = 1

asig loscil 1, 1, ifn, ibas
outs asig, asig

endin

instr 2 ;plays only non-deffered sound

isnd = p4
aread line sr*p3, p3, 0 ;play this backward
asig tablei aread, isnd ;use table 1
outs asig, asig

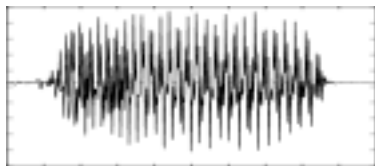
endin
</CsInstruments>
<CsScore>
f 1 0 131072 1 "beats.wav" 0 0 0 ;non-deferred sound
f 2 0 0 1 "flute.aiff" 0 0 0 ;& deferred sounds in
f 3 0 0 1 "beats.ogg" 0 0 0 ;different formats

i 1 0 1 1
i 1 + 1 2
i 1 + 1 3

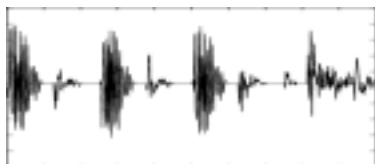
i 2 4 2 1 ;non-deffered sound for instr. 2
e
</CsScore>
</CsoundSynthesizer>
```

Voici les diagrammes des formes d'onde des routines GEN01 utilisées dans l'exemple :

f 1 0 131072 1 "beats.wav" 0 0 0 - son non-différé.



f 2 0 0 1 "flute.aiff" 0 0 0 - son différé



f 3 0 0 1 "beats.ogg" 0 0 0 - son différé

## Crédits

Septembre 2003. Remerciements au Dr. Richard Boulanger pour avoir signalé les références au format de fichier AIFF.

Revisé dans la v6.11 et dans la v6.12 pour clarifier le statut des en-têtes et des fichiers bruts et pour ajouter trois formats d'échantillons.

# GEN02

GEN02 — Transfère les données des p-champs dans une table de fonction.

## Description

Ce sous-programme transfère les données des p-champs dans une table de fonction.

## Syntaxe

```
f # date taille 2 v1 v2 v3 ...
```

## Initialisation

*taille* -- nombre de points dans la table. La taille de table maximale est de 16777216 ( $2^{24}$ ) points. On peut donner une taille de zéro ; dans ce cas le nombre de valeurs fixe la longueur de la table.

*v1*, *v2*, *v3*, etc. -- valeurs à copier directement dans l'espace de la table. Les valeurs copiées peuvent comprendre le point de garde de la table ; les cellules de la table non remplies contiendront des zéros.



### Note

Si *p4* (le numéro de la routine GEN) est positif, la table sera post-normalisée (reproportionnée avec une valeur absolue maximale de 1 après génération). Une valeur de *p4* négative empêche cette opération. On utilisera habituellement la valeur -2 avec cette fonction GEN, afin que les valeurs ne soient pas normalisées.

## Exemples

Voici un exemple de la routine GEN02. Il utilise le fichier *gen02.csd* [examples/gen02.csd].

### Exemple 1198. Exemple de la routine GEN02.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;;realtime audio out
;-iadc     ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen02.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

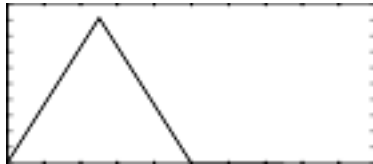
ifn = p4      ;choose different tables of GEN02
```

```
kcps init 1/p3      ;index over the length of entire note
kndx phasor kcps
ixmode = 1          ;normalize index data
kamp tablei kndx, ifn, ixmode
asig poscil kamp, 440, 1 ;use GEN02 as envelope for amplitude
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 8192 10 1 ;sine wave
f 2 0 5 2 0 2 0
f 3 0 5 2 0 2 10 0
f 4 0 9 2 0 2 10 100 0

i 1 0 2 2
i 1 3 2 3
i 1 6 2 4
e
</CsScore>
</CsoundSynthesizer>
```

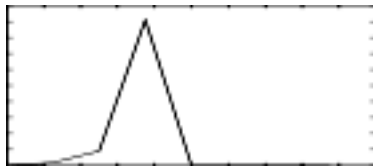
Voici les diagrammes des formes d'onde des routines GEN02 utilisées dans l'exemple :



f 2 0 5 2 0 2 0



f 3 0 5 2 0 2 10 0



f 4 0 9 2 0 2 10 100 0

## Voir aussi

*GEN17*

## Crédits

Décembre 2002. Merci à Rasmus Ekman, pour avoir corrigé la limite de la variable *PMAX*.

L'utilisation d'une taille nulle est nouvelle dans la version 6.12

# GEN03

GEN03 — Génère une table de fonction en évaluant un polynôme.

## Description

Ce sous-programme génère une table de fonction en évaluant un polynôme en  $x$  sur un intervalle fixe et avec des coefficients spécifiés.

## Syntaxe

```
f # date taille 3 xval1 xval2 c0 c1 c2 ... cn
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1.

*xval1*, *xval2* -- limites gauche et droite de l'intervalle  $x$  sur lequel le polynôme est défini ( $xval1 < xval2$ ). Celles-ci produiront la 1ère valeur stockée et la (puissance-de-2 plus 1)ème valeur stockée respectivement dans la table de la fonction générée.

*c0*, *c1*, *c2*, ..., *cn* -- coefficients du polynôme d'ordre  $n$

$$C_0 + C_1x + C_2x^2 + \dots + C_nx^n$$

Les coefficients peuvent être des nombres réels positifs ou négatifs ; un zéro dénote un terme manquant dans le polynôme. La liste de coefficients commence en p7, avec une limite maximale actuelle de 144 termes.



### Note

- Le segment défini  $[fn(xval1), fn(xval2)]$  est distribué également. Ainsi une table de 512 points sur l'intervalle  $[-1,1]$  aura son origine à la cellule 257 (au début de la seconde moitié). Si le point de garde est requis, les deux valeurs  $fn(-1)$  et  $fn(1)$  existeront dans la table.
- *GEN03* est utile en conjonction avec *table* ou *tablei* pour le waveshaping audio (modification du son par distortion non-linéaire). Les coefficients pour produire un formant particulier à partir d'un index de lecture sinusoïdal d'amplitude connue peuvent être déterminés avant le traitement en utilisant des algorithmes tels que les formules de Tchebychev. Voir aussi *GEN13*.

## Exemples

Voici un exemple de la routine GEN03. Il utilise le fichier *gen03.csd* [examples/gen03.csd].

### Exemple 1199. Exemple de la routine GEN03.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen03.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
;example by Russell Pinkston - Univ. of Texas (but slightly modified)

gisine   ftgen 1, 0, 16384, 10, 1      ;sine wave

instr    1

ihertz = cpspch(p4)
ipkamp = p5
iwsfn = p6           ;waveshaping function
inmfn = p7           ;normalization function
aenv   linen 1, .01, p3, .1           ;overall amp envelope
actrl  linen 1, 2, p3, 2             ;waveshaping index control
aindex poscil actrl/2, ihertz, gisine ;sine wave to be distorted
asignal tablei .5+aindex, iwsfn, 1    ;waveshaping
anormal tablei actrl, inmfn,1         ;amplitude normalization
asig   = asignal*anormal*ipkamp*aenv
asig   dcblock2 asig                 ;get rid of possible DC
outs   asig, asig

endin
</CsInstruments>
<CsScore>
; first four notes are specific Chebyshev polynomials using gen03. The values were obtained from Dodge

f4 0 513 3 -1 1 0 1   ; First-order Chebyshev: x
f5 0 257 4 4 1       ; Normalizing function for fn4

f6 0 513 3 -1 1 -1 0 2 ; Second-order Chebyshev: 2x2 - 1
f7 0 257 4 6 1       ; Normalizing function for fn6

f8 0 513 3 -1 1 0 -3 0 4 ; Third-order Chebyshev: 4x3 - 3x
f9 0 257 4 8 1       ; Normalizing function for fn8

f10 0 513 3 -1 10 0 -7 0 56 0 -112 0 64 ; Seventh-order Chebyshev: 64x7 - 112x5 + 56x3 - 7x
f11 0 257 4 10 1     ; Normalizing function for fn10

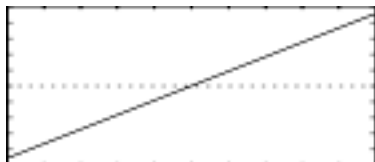
f12 0 513 3 -1 1 5 4 3 2 2 1 ; a 4th order polynomial function over the x-interval -1 to 1
f13 0 257 4 12 1     ; Normalizing function for fn12

; five notes with same fundamental, different waveshape & normalizing functions
;      pch  amp  wsfn  nmfn
i1 0 3 8.00 .7 4 5
i1 + . . . 6 7
i1 + . . . 8 9
i1 + . . . 10 11
i1 + . . . 12 13

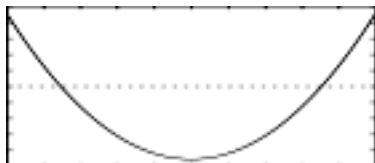
e
</CsScore>
</CsoundSynthesizer>

```

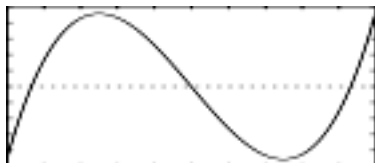
Voici les diagrammes des formes d'onde des routines GEN03 utilisées dans l'exemple :



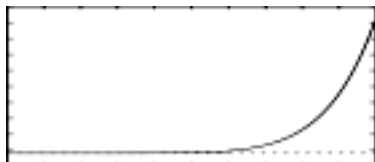
f4 0 513 3 1 1 0 1 - Tchebychev du premier ordre :  $x$



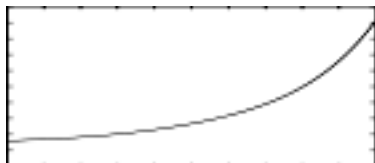
f6 0 513 3 -1 1 -1 0 2 - Tchebychev du second ordre :  $2x^2 - 1$



f8 0 513 3 -1 1 0 -3 0 4 - Tchebychev du troisième ordre :  $4x^3 - 3x$



f10 0 513 3 -1 10 0 -7 0 56 0 -112 0 64 - Tchebychev du septième ordre :  $64x^7 - 112x^5 + 56x^3 - 7x$



f12 0 513 3 -1 1 5 4 3 2 2 1 - une fonction polynomiale du quatrième ordre,  $x$  compris entre -1 et 1

## Voir aussi

*GEN13*, *GEN14* et *GEN15*.

Information au sujet des polynômes de Tchebychev sur Wikipedia : [http://fr.wikipedia.org/wiki/Polynôme\\_de\\_Tchebychev](http://fr.wikipedia.org/wiki/Polynôme_de_Tchebychev) [[http://fr.wikipedia.org/wiki/Polyn%C3%B4me\\_de\\_Tchebychev](http://fr.wikipedia.org/wiki/Polyn%C3%B4me_de_Tchebychev)]

# GEN04

GEN04 — Génère une fonction de normalisation.

## Description

Ce sous-programme génère une fonction de normalisation en examinant le contenu d'une table existante.

## Syntaxe

```
f # temps taille 4 source# modesource
```

## Initialisation

*taille* -- nombre de points dans la table. Une puissance-de-2 plus 1. Ne doit pas dépasser (sauf de 1) la taille de la table source examinée ; limitée à exactement la moitié de cette taille si *modesource* est de type décalage (voir ci-dessous).

*source #* -- numéro de table de la fonction stockée à examiner.

*modesource* -- une valeur codée, spécifiant comment la table source doit être parcourue pour obtenir la fonction de normalisation. Zéro indique que la source doit être parcourue de gauche à droite. Une valeur non nulle indique que la source a une structure bipolaire ; la lecture commencera au point médian et progressera vers les extrémités, par paires de points équidistants du centre.



### Note

- La fonction de normalisation dérive de la progression des maxima absolus de la table source parcourue. La nouvelle table est créée de gauche à droite, en stockant des valeurs égales à  $1/(\text{maximum absolu lu jusqu'à là})$ . Les valeurs stockées commenceront ainsi par  $1/(\text{première valeur lue})$ , et deviendront progressivement plus petites lorsque de nouveaux maxima seront rencontrés. Pour une table source normalisée (valeurs  $\leq 1$ ), les valeurs dérivées descendront de  $1/(\text{première valeur lue})$  jusqu'à 1. Si la première valeur lue est zéro, son inverse sera fixé à 1.
- la fonction de normalisation générée par *GEN04* n'est pas elle-même normalisée.
- *GEN04* est utile pour modifier l'échelle d'un signal dérivé d'une table afin qu'il ait une amplitude de crête consistante. On l'utilise particulièrement en waveshaping quand la porteuse (ou fonction d'indexation) a une amplitude inférieure à la moitié de l'échelle complète.

## Exemples

Voici un exemple de la routine GEN04. Il utilise le fichier *gen04.csd* [examples/gen04.csd].

### Exemple 1200. Un exemple de la routine GEN04.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform
```



```

-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen04.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisine   ftgen 0, 0, 16384, 10, 1      ;sine wave

instr    1

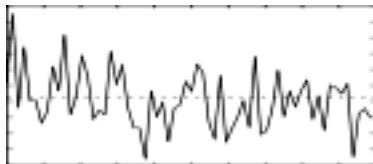
ihertz = cpspch(p4)
ipkamp = p5
iwsfn = p6      ;waveshaping function
inmfn = p7      ;normalization function
agate   linen   1, .01, p3, .1          ;overall amp envelope
kctrl   linen   .9, 2, p3, 2          ;waveshaping index control
aindex  poscil  kctrl/2, ihertz, gisine ;sine wave to be distorted
asignal tablei  .5*aindex, iwsfn, 1    ;waveshaping
knormal tablei  1/kctrl, inmfn, 1      ;amplitude normalization
outs    asignal*knormal*ipkamp*agate, asignal*knormal*ipkamp*agate

endin
</CsInstruments>
<CsScore>
f1 0 64 21 6 ;Gaussian (random) distribution
f2 0 33 4 1 1 ;normalizing function with midpoint bipolar offset

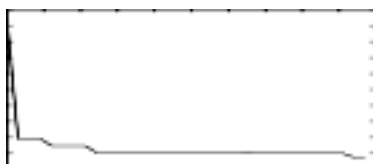
s
; st dur pch amp   wsfm inmfn
i1 0 4 6.00 .7 1 2
i1 4 . 7.00 .
i1 8 . 8.00 .
;-----
f3 0 1025 13 1 1 0 5 0 5 0 10 ;Chebyshev algorithm
f4 0 513 4 3 1 ;normalizing function with midpoint bipolar offset
s
; st dur pch amp   wsfm inmfn
i1 0 4 6.00 .9 3 4
i1 4 . 7.00 .
i1 8 . 8.00 .
e
</CsScore>
</CsoundSynthesizer>

```

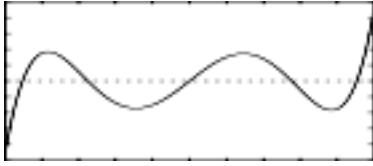
Voici les diagrammes des formes d'onde des routines GEN04 utilisées dans l'exemple :



f1 0 64 21 6 - distribution (aléatoire) gaussienne



f2 0 33 4 1 1 - et sa fonction de normalisation avec décalage bipolaire du point central



f3 0 1025 13 1 1 0 5 0 5 0 10 - algorithme de Tchebychev



f4 0 513 4 3 1 - et sa fonction de normalisation avec décalage bipolaire du point central

# GEN05

GEN05 — Construit des fonctions à partir de morceaux de courbes exponentielles.

## Description

Construit des fonctions à partir de morceaux de courbes exponentielles.

## Syntaxe

```
f # date taille 5 a n1 b n2 c ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*).

*a*, *b*, *c*, etc. -- valeurs d'ordonnée, dans les p-champs de numéros impairs p5, p7, p9, . . . Elle doivent être non nulles et de même signe.

*n1*, *n2*, etc. -- longueurs des morceaux (nombre de positions mémorisées), dans les p-champs de numéros pairs. Ne peuvent pas être négatives, mais un zéro est significatif pour spécifier des formes d'onde discontinues. La somme  $n1 + n2 + \dots$  sera normalement égale à *taille* pour les fonctions complètement spécifiées. Si la somme est inférieure, les positions de la fonction non comprises seront mises à zéro ; si la somme est supérieure, seules les premières *taille* positions seront stockées. Noter que les valeurs sont arrondies en nombres entiers avant leur utilisation.



### Note

- Si p4 est positif, les fonctions sont post-normalisées (reproportionnées avec une valeur absolue maximale de 1 après génération). Une valeur de p4 négative empêche cette opération.
- Une interpolation linéaire sur des points discrets implique une augmentation ou une diminution le long d'un segment par des sauts égaux entre des positions adjacentes ; une interpolation exponentielle implique une progression par rapports égaux. Dans les deux formes l'interpolation de *a* à *b* suppose que la valeur *b* sera atteinte à la (n + 1)ème position. Pour les fonctions discontinues, et pour les segments dépassant la dernière position, cette valeur ne sera pas atteinte, bien qu'elle puisse éventuellement apparaître comme résultat d'une mise à l'échelle finale.

## Exemples

Voici un exemple simple de la routine GEN05. Il utilise le fichier *gen05.csd* [examples/gen05.csd].

### Exemple 1201. Un exemple de la routine GEN05.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform
```

```

-odac      ;;;realtime audio out
;-iadc     ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen05.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifn = p4      ;choose different tables for GEN05
kcps init 1/p3      ;index over the length of entire note
kndx phasor kcps
ixmode = 1      ;normalize index data
kamp tablei kndx, ifn, ixmode
asig poscil kamp, 440, 1
      outs asig, asig

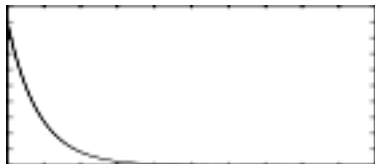
endin
</CsInstruments>
<CsScore>
f 1 0 8192 10 1 ;sine wave
f 2 0 129 5 1 100 0.0001 29 ;short attack
f 3 0 129 5 0.00001 87 1 22 .5 20 0.0001 ;long attack

i 1 0 2 2
i 1 3 2 3

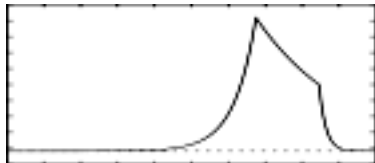
e
</CsScore>
</CsoundSynthesizer>

```

Voici les diagrammes des formes d'onde des routines GEN05 utilisées dans l'exemple :



f 2 0 129 5 1 100 0.0001 29 - forme d'onde allant de 1 à 0.0001 sur 100 points, puis restant sur cette valeur pendant 29 points



f 3 0 129 5 0.00001 87 1 22 .5 20 0.0001 - forme d'onde allant de 0.00001 à 1 sur 87 points, puis de 1 à 0.5 sur 22 points et enfin de 0.5 to 0.0001 sur 20 points

## Voir aussi

GEN06, GEN07 et GEN08

## GEN06

GEN06 — Génère une fonction composée de morceaux de polynômes cubiques.

### Description

Ce sous-programme génèrera une fonction composée de morceaux de polynômes cubiques, couvrant les points spécifiés trois par trois.

### Syntaxe

```
f #   date   taille   6   a   n1   b   n2   c   n3   d ...
```

### Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*).

*a, c, e, ...* -- les maxima ou les minima locaux des morceaux successifs, dépendant de la relation de ces points avec les inflexions adjacentes. Peuvent être positifs ou négatifs.

*b, d, f, ...* -- ordonnées des points d'inflexion aux extrémités des segments curvilignes successif. Peuvent être positifs ou négatifs.

*n1, n2, n3 ...* -- nombre de valeurs stockées entre les points spécifiés. Ne peuvent pas être négatifs, mais un zéro est significatif pour spécifier des discontinuités. La somme  $n1 + n2 + \dots$  sera normalement égale à *taille* pour les fonctions complètement spécifiées. (Pour des détails, voir *GEN05*).



#### Note

*GEN06* construit une fonction stockée à partir de fonctions polynomiales cubiques. Les morceaux groupent les valeurs d'ordonnée par groupes de 3 : point d'inflexion, maximum/minimum, point d'inflexion. Le premier segment complet comprend *b, c, d* et il a pour longueur  $n2 + n3$ , le suivant comprend *d, e, f* et il a pour longueur  $n4 + n5$ , etc. Le premier morceau (*a, b* de longueur *n1*) est incomplet avec seulement une inflexion ; le dernier morceau peut être incomplet aussi. Bien que les points d'inflexion *b, d, f ...* figurent chacun dans deux segments (un à gauche et un à droite), les pentes des deux segments restent indépendantes à ce point commun (c'est-à-dire que la dérivée première sera probablement discontinue). Quand *a, c, e...* sont alternativement maximum et minimum, les jointures des inflexions seront relativement douces ; pour des maxima successifs ou des minima successifs les inflexions seront en peigne.

### Exemples

Voici un exemple de la routine GEN06. Il utilise le fichier *gen06.csd* [examples/gen06.csd].

#### Exemple 1202. Un exemple de la routine GEN06.

```
<CsoundSynthesizer>  
<CsOptions>
```

```

; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen06.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifn = p4    ;choose between tables
kcps init 1/p3    ;create index over duration of note.
kndx phasor kcps
ixmode = 1
kval table kndx, ifn, ixmode    ;normalize mode

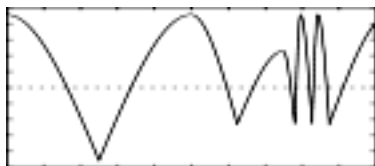
kfreq = kval * 30    ;scale frequency to emphasixe effect
asig poscil .7, 220 + kfreq, 1    ;add to frequency
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave.
f 2 0 513 6 1 128 -1 128 1 64 -.5 64 .5 16 -.5 8 1 16 -.5 8 1 16 -.5 84 1 16 -.5 8 .1 16 -.1 17 0
f 3 0 513 6 0 128 0.5 128 1 128 0 129 -1

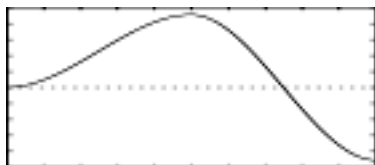
i 1 0 3 2
i 1 4 3 3
e
</CsScore>
</CsoundSynthesizer>

```

Voici les diagrammes des formes d'onde des routines GEN06 utilisées dans l'exemple :



f 2 0 513 6 1 128 -1 128 1 64 -.5 64 .5 16 -.5 8 1 16 -.5 8 1 16 -.5 84 1 16 -.5 8 .1 16 -.1 17 0 - une courbe pas trop lisse



f 3 0 513 6 0 128 0.5 128 1 128 0 129 -1 - une courbe allant de 0 à 1 puis à -1, avec respectivement un minimum, un maximum et un minimum à ces valeurs. Les points d'inflexion sont à 0.5 et à 0 et sont relativement lisses

## Voir aussi

GEN05, GEN07 et GEN08

# GEN07

GEN07 — Construit des fonctions à partir de morceaux de lignes droites.

## Description

Construit des fonctions à partir de morceaux de lignes droites.

## Syntaxe

`f # date taille 7 a n1 b n2 c ...`

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir *instruction f*).

*a, b, c*, etc. -- valeurs d'ordonnée, dans les p-champs de numéros impairs p5, p7, p9, ...

*n1, n2*, etc. -- longueur de segment (nombre de positions en mémoire), dans les p-champs de numéros pairs. Ne peuvent pas être négatifs, mais un zéro est significatif pour spécifier des formes d'onde discontinues (comme dans l'exemple ci-dessous). La somme  $n1 + n2 + \dots$  sera normalement égale à *taille* pour les fonctions complètement spécifiées. Si la somme est inférieure, les positions de la fonction non comprises seront mises à zéro ; si la somme est supérieure, seules les premières *taille* positions seront stockées.



### Note

- Si p4 est positif, les fonctions sont post-normalisées (reproportionnées avec une valeur absolue maximale de 1 après génération). Une valeur de p4 négative empêche cette opération.
- Une interpolation linéaire sur des points discrets implique une augmentation ou une diminution le long d'un segment par des sauts égaux entre des positions adjacentes ; une interpolation exponentielle implique une progression par rapports égaux. Dans les deux formes l'interpolation de *a* à *b* suppose que la valeur *b* sera atteinte à la (n + 1)ème position. Pour les fonctions discontinues, et pour les segments dépassant la dernière position, cette valeur ne sera pas atteinte, bien qu'elle puisse éventuellement apparaître comme résultat d'une mise à l'échelle finale.

## Exemples

Voici un exemple de la routine GEN07. Il utilise le fichier *gen07.csd* [examples/gen07.csd].

### Exemple 1203. Un exemple de la routine GEN07.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;;realtime audio out
;-iadc     ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen07.wav -W ;;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;use GEN07 to alter frequency

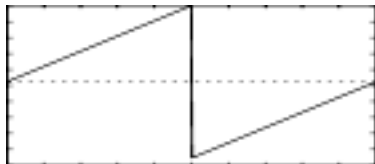
ifn = p4 ;use different GEN07 tables
kcps init 10/p3 ;index ftable 10 times over the duration of entire note
kndx phasor kcps
ixmode = 1 ;normalize index data
kfrq tablei kndx, ifn, ixmode
kfrq = kfrq*1000 ;scale
asig poscil .8, 1220+kfrq, 1 ;add to frequency
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 8192 10 1 ;sine wave
f 2 0 1024 7 0 512 1 0 -1 512 0 ;sawtooth up and down
f 3 0 1024 7 1 512 1 0 -1 512 -1 ;square
f 4 0 1024 7 1 1024 -1 ;saw down

i 1 0 2 2
i 1 + 2 3
i 1 + 1 4
e
</CsScore>
</CsoundSynthesizer>

```

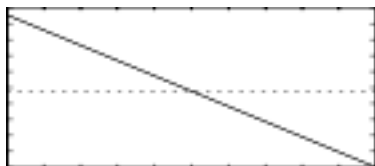
Voici les diagrammes des formes d'onde des routines GEN07 utilisées dans l'exemple :



f 2 0 1024 7 0 512 1 0 -1 512 0 - dent de scie ascendante commençant et finissant à 0



f 3 0 1024 7 1 512 1 0 -1 512 -1 - onde carrée positive puis négative



f 4 0 1024 7 1 1024 -1 - dent de scie descendante, positive puis négative

## Voir aussi

GEN05, GEN06 et GEN08



# GEN08

GEN08 — Génère une courbe spline cubique par morceaux.

## Description

Ce sous-programme génèrera une courbe spline cubique par morceaux, la plus lisse possible le long de tous les points spécifiés.

## Syntaxe

`f # date taille 8 a n1 b n2 c n3 d ...`

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir *instruction f*).

*a, b, c, etc.* -- valeurs d'ordonnée de la fonction.

*n1, n2, n3 ...* -- longueur de chaque segment mesurée en valeurs mémorisées. Ne peuvent pas être nulles, mais peuvent être fractionnaires. Un segment particulier peut stocker ou non des valeurs ; les valeurs stockées seront générées à des points entiers à partir de début de la fonction. La somme  $n1 + n2 + \dots$  sera normalement égale à *taille* pour les fonctions complètement spécifiées.



### Note

- *GEN08* construit une table stockée à partir de morceaux d'une fonction polynomiale cubique. Chaque segment s'étend entre deux points spécifiés mais dépend aussi de leurs voisins de chaque côté. Les segments voisins coïncideront en valeur et en pente à leur point commun. (La pente commune est celle d'une parabole passant par ce point et ses deux voisins). La pente aux deux extrémités de la fonction est forcée à zéro (plate).
- *Conseil* : pour créer une discontinuité de pente ou de valeur dans la fonction stockée, disposer une série de points dans l'intervalle entre deux valeurs stockées ; faire de même pour une pente non nulle à l'une des extrémités.

## Exemples

Voici un exemple de la routine GEN08. Il utilise le fichier *gen08.csd* [examples/gen08.csd].

### Exemple 1204. Un exemple de la routine GEN08.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen08.wav -W ;; for file output any platform
</CsOptions>
```

```

<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifn = p4 ;choose between tables
kcps init 1/p3 ;create index over duration of note.
kndx phasor kcps
ixmode = 1
kval table kndx, 2, ixmode ;normalize index data

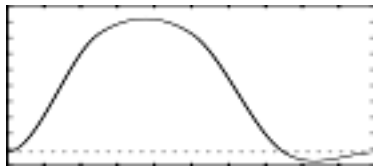
ibasefreq = 440
kfreq = kval * 100 ;scale
asig poscil .7, ibasefreq + kfreq, 1 ;and add to frequency
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave.
f 2 0 65 8 0 16 1 16 1 16 0 17 0
f 3 0 65 8 -1 32 1 2 0 14 0 17 0

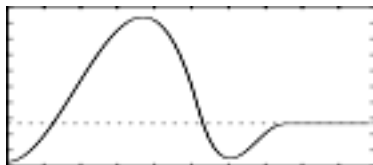
i 1 0 2 1
i 1 3 2 2
e
</CsScore>
</CsSoundSynthesizer>

```

Voici les diagrammes des formes d'onde des routines GEN08 utilisées dans l'exemple :



f 2 0 65 8 0 16 1 16 1 16 0 17 0 - une courbe avec une bosse régulière au milieu, brièvement négative sur les bords et plate aux extrémités



f 3 0 65 8 -1 32 1 2 0 14 0 17 0 - à partir d'une valeur négative, une courbe avec une bosse régulière, puis négative créant une petite bosse et enfin plate

## Voir aussi

GEN05, GEN06 et GEN07

# GEN09

GEN09 — Génère des formes d'ondes complexes obtenues par une somme pondérée de sinus.

## Description

Ce sous-programme génère des formes d'ondes complexes obtenues par une somme pondérée de sinus. La spécification de chaque partiel nécessite 3 p-champs avec *GEN09*.

## Syntaxe

```
f # date taille 9 pna ampa phsa pnb ampb phsb ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*).

*pna*, *pnb*, etc. -- numéro de partiel (par rapport à un fondamental qui occuperait *taille* positions par période) des sinus a, sinus b, etc. Doit être positif, mais pas nécessairement un nombre entier, c'est-à-dire que des partiels non harmoniques sont autorisés. Les partiels peuvent être dans n'importe quel ordre.

*ampa*, *ampb*, etc. -- amplitude des partiels *pna*, *pnb*, etc. Ce sont des amplitudes relatives, car la forme d'onde complexe peut être reproporionnée à posteriori. On peut utiliser des valeurs négatives pour signifier une opposition de phase (180 degrés).

*phsa*, *phsb*, etc. -- phase initiale des partiels *pna*, *pnb*, etc., exprimée en degrés (0-360).



### Note

- Ces sous-programmes génèrent des fonctions stockées qui sont la somme de sinus de différentes fréquences. Les deux restrictions majeures de *GEN10* qui sont des partiels harmoniques et en phase ne s'appliquent pas à *GEN09* ou à *GEN19*.
- Dans chaque cas, l'onde complexe, une fois calculée, est reproporionnée à l'unité si p4 est positif. Un p4 négatif empêchera cette opération.

## Exemples

Voici un exemple de la routine GEN09. Il utilise le fichier *gen09.csd* [examples/gen09.csd].

### Exemple 1205. Exemple de la routine GEN09.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
```

```

; For Non-realtime ouput leave only the line below:
; -o gen09.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gil ftgen 1,0,2^10,9, 1,3,0, 3,1,0, 9,0.333,180 ;an approximation of a square wave
gi2 ftgen 2,0,2^10,9, 1,3,180, 3,1,0, 9,0.333,0 ;same values as gil, except some phase values
gi3 ftgen 3,0,2^10,9, 1, .4, 0, 2.2, .5, 0, 3.8, 1, 0 ;inharmonic, but does not sound well --> wav
gi4 ftgen 4,0,2^10,9, 10, .4, 0, 22, .5, 0, 38, 1, 0 ;the same proportions, but value of partial n
;because the sudden "jump" like the one in gi3 will pop up only once in 10 repetitions

instr 1

kamp = .6
kcps = 220
ifn = p4

asig poscil kamp, kcps*p5, ifn
outs asig,asig

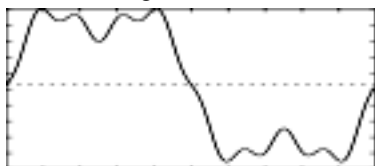
endin
</CsInstruments>
<CsScore>

i 1 0 2 1 1 ;subtle difference between table 1 and 2
i 1 3 2 2 1
i 1 7 2 3 1 ;big difference between table 3 and 4
i 1 10 2 4 .1 ;p5 has to compensate for the 10 repetitions of gi4 as opposed to gi3 to get the same pit

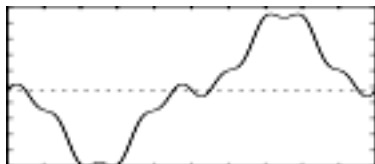
e
</CsScore>
</CsoundSynthesizer>

```

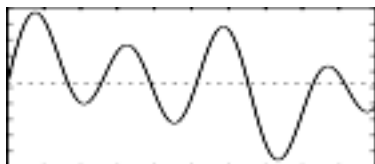
Voici les diagrammes des formes d'onde des routines GEN09 utilisées dans l'exemple :



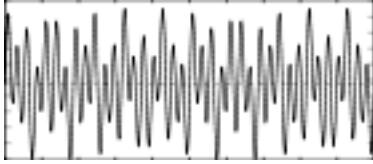
gil ftgen 1,0,2^10,9, 1,3,0, 3,1,0, 9,0.333,180 - approximation d'une onde carrée



gi2 ftgen 2,0,2^10,9, 1,3,180, 3,1,0, 9,0.333,0 - mêmes valeurs que gi1, sauf pour la phase



gi3 ftgen 3,0,2^10,9, 1,2,0, 3,2,0, 9,0.333,180 - partiels inharmoniques, avec une distorsion due au saut abrupt au début et à la fin de l'onde



gi4 ftgen 4,0,2^10,9, 1,2,180, 3,2,0, 9,0.333,0 - même rapport que gi3, avec moins d'artefacts

## Voir aussi

*GEN10, GEN19*

# GEN10

GEN10 — Génère des formes d'ondes complexes obtenues par une somme pondérée de sinus.

## Description

Ce sous-programme génère des formes d'ondes complexes obtenues par une somme pondérée de sinus. La spécification de chaque partiel nécessite 1 p-champ avec *GEN10*.

## Syntaxe

```
f # date taille 10 amp1 amp2 amp3 amp4 ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*).

*amp1*, *amp2*, *amp3*, etc. -- amplitudes relatives des partiels harmoniques fixes de numéro 1, 2, 3, etc., commençant en p5. Les partiels non désirés recevront une amplitude nulle.



### Note

- Ces sous-programmes génèrent des fonctions stockées qui sont la somme de sinus de différentes fréquences. Les deux restrictions majeures de *GEN10* qui sont des partiels harmoniques et en phase ne s'appliquent pas à *GEN09* ou à *GEN19*.
- Dans chaque cas, l'onde complexe, une fois calculée, est reproportionnée à l'unité si p4 est positif. Un p4 négatif empêchera cette opération.

## Exemples

Voici un exemple de la routine GEN10. Il utilise le fichier *gen10.csd* [examples/gen10.csd].

### Exemple 1206. Un exemple de la routine GEN10.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen10.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kamp = .6
```

```

kcps = 440
ifn = p4

asig oscil kamp, kcps, ifn
outs asig,asig

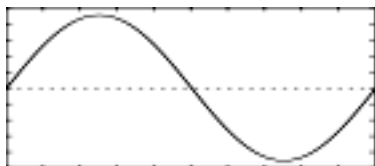
endin
</CsInstruments>
<CsScore>
f1 0 16384 10 1 ; Sine
f2 0 16384 10 1 0.5 0.3 0.25 0.2 0.167 0.14 0.125 .111 ; Sawtooth
f3 0 16384 10 1 0 0.3 0 0.2 0 0.14 0 .111 ; Square
f4 0 16384 10 1 1 1 1 0.7 0.5 0.3 0.1 ; Pulse

i 1 0 2 1
i 1 3 2 2
i 1 6 2 3
i 1 9 2 4

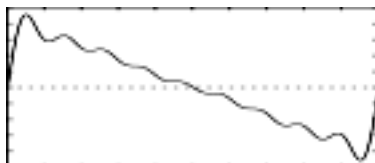
e
</CsScore>
</CsoundSynthesizer>

```

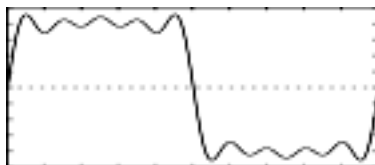
Voici les diagrammes des formes d'onde des routines GEN10 utilisées dans l'exemple :



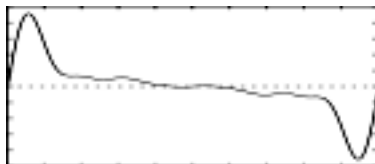
f 1 0 16384 10 1 - onde sinus (seulement la fréquence fondamentale)



f 2 0 16384 10 1 0.5 0.3 0.25 0.2 0.167 0.14 0.125 .111 - dent de scie, avec la fondamentale et 8 harmoniques



f 3 0 16384 10 1 0 0.3 0 0.2 0 0.14 0 .111 - onde carrée, avec la fondamentale et 8 harmoniques mais les 4 de rang paire sont nulles



f 4 0 16384 10 1 1 1 1 0.7 0.5 0.3 0.1 - pulsation, avec la fondamentale et 8 harmoniques

### Exemple 1207. Un exemple de la routine GEN10.

Voir les sections *Audio en Temps-Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen10.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

kamp = .6
kcps = 440
ifn = p4

asig oscil kamp, kcps, ifn
outs asig,asig

endin
</CsInstruments>
<CsScore>
f1 0 16384 10 1 ; Sine
f2 0 16384 10 1 0.5 0.3 0.25 0.2 0.167 0.14 0.125 .111 ; Sawtooth
f3 0 16384 10 1 0 0.3 0 0.2 0 0.14 0 .111 ; Square
f4 0 16384 10 1 1 1 1 0.7 0.5 0.3 0.1 ; Pulse

i 1 0 2 1
i 1 3 2 2
i 1 6 2 3
i 1 9 2 4

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

GEN09, GEN11 et GEN19.



# GEN11

GEN11 — Génère un ensemble additif de partiels cosinus.

## Description

Ce sous-programme génère un ensemble additif de partiels cosinus, à la manière des générateurs de Csound *buzz* et *gbuzz*.

## Syntaxe

```
f # date taille ll nh [lh] [r]
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*).

*nh* -- nombre d'harmoniques demandés. Doit être positif.

*lh*(optional) -- harmonique présent le plus bas. Peut être positif, nul ou négatif. L'ensemble d'harmoniques peut démarrer à n'importe quel numéro d'harmonique et progresse vers le haut ; si *lh* est négatif, tous les harmoniques en dessous de zéro se réfléchiront autour de zéro pour produire des harmoniques positifs sans changement de phase (car le cosinus est une fonction paire), et s'ajouteront de façon constructive aux harmoniques positifs de l'ensemble. La valeur par défaut est 1.

*r*(facultatif) -- multiplicateur dans une série de coefficients d'amplitude. C'est une séries de puissances : si le *lh* ème harmonique a un coefficient d'amplitude de *A* le (*lh* + *n*)ème harmonique aura un coefficient de  $A * r^n$ , c'est-à-dire que les valeurs d'amplitudes suivent une courbe exponentielle. *r* peut être positif, nul ou négatif, et n'est pas restreint à des entiers. La valeur par défaut est 1.



### Note

- Ce sous-programme est une version invariante dans le temps des générateurs de Csound *buzz* et *gbuzz*, et il est similairement utile comme source sonore complexe pour la synthèse soustractive. Si *lh* et *r* sont utilisés, il agit comme *gbuzz* ; si les deux sont absents ou égaux à 1, il se réduit au générateur plus simple *buzz* (c'est-à-dire *nh* harmoniques d'amplitude égale commençant avec le fondamental).
- Lire la forme d'onde stockée avec un oscillateur est plus efficace que d'utiliser les unités dynamiques *buzz*. Cependant, le contenu spectral est invariant et il faut faire attention à ce que les harmoniques les plus hauts ne dépassent pas la fréquence de Nyquist pour éviter les repliements.

## Exemples

Voici un exemple de la routine GEN11. Il utilise le fichier *gen11.csd* [examples/gen11.csd].

### Exemple 1208. Un exemple de la routine GEN11.

```
<CsoundSynthesizer>  
<CsOptions>
```

```

; Select audio/midi flags here according to platform
-o dac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen11.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

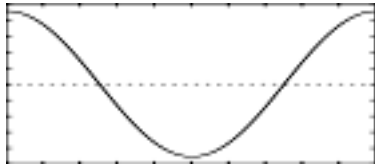
ifn = p4
asig oscil .8, 220, ifn
outs asig,asig

endin
</CsInstruments>
<CsScore>
f 1 0 16384 11 1 1 ;number of harmonics = 1
f 2 0 16384 11 10 1 .7 ;number of harmonics = 10
f 3 0 16384 11 10 5 2 ;number of harmonics = 10, 5th harmonic is amplified 2 times

i 1 0 2 1
i 1 + 2 2
i 1 + 2 3
e
</CsScore>
</CsoundSynthesizer>

```

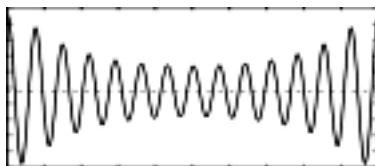
Voici les diagrammes des formes d'onde des routines GEN11 utilisées dans l'exemple :



f 1 0 16384 11 1 1



f 2 0 16384 11 10 1 .7



f 3 0 16384 11 10 5 2

## Voir aussi

*GEN10*

# GEN12

GEN12 — Génère le logarithme d'une fonction de Bessel de seconde espèce modifiée.

## Description

Génère le logarithme d'une fonction de Bessel de seconde espèce modifiée, d'ordre 0, adaptée pour la MF modulée en amplitude.

## Syntaxe

**f** # date taille 12 intx

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*).

*intx* -- spécifie l'intervalle des  $x$  [0 à +*intx*] sur lequel la fonction est définie.



### Note

- Ce sous-programme calcule le logarithme naturel d'une fonction de Bessel de seconde espèce modifiée, d'ordre 0 (habituellement écrite comme  $I_0$ ), sur l'intervalle des  $x$  demandé. Cet appel devrait désactiver la normalisation.
- Cette fonction est utile comme facteur d'échelle d'amplitude dans la MF à période synchrone modulée en amplitude. (Voir Palamin & Palamin, *J. Audio Eng. Soc.*, 36/9, Sept. 1988, pp.671-684.) L'algorithme est intéressant car il permet de rendre le spectre de MF, habituellement symétrique, asymétrique autour d'une fréquence autre que la porteuse, et il est ainsi utile pour placer des formants. En utilisant un index de lecture dans la table de  $I(r - 1/r)$ , où  $I$  est l'index de modulation et  $r$  est un paramètre exponentiel affectant l'importance des partiels, l'algorithme Palamin se montre relativement efficace, ne demandant que des oscil, des lecture de table, et un appel d'*exp*.

## Exemples

Voici un exemple de la routine GEN12. Il utilise le fichier *gen12.csd* [examples/gen12.csd].

### Exemple 1209. Un exemple de la routine GEN12.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
```

```

; For Non-realtime ouput leave only the line below:
; -o gen12.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
;example from the Csound Book, page 87
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

idur = p3
iamp = p4
icarfrq = p5
imodfrq = p6
aenv expseg .01, idur*.1, iamp, idur*.8, iamp*.75, idur*.1, .01
i1 = p7*imodfrq ;p7=modulation index start
i2 = p8*imodfrq ;p8=modulation index end
adev line i1, idur, i2 ;modulation frequency
aindex line p7, idur, p8 ;modulation index

ar linseg 1, .1, p9, p3-.2, p10, .1, 1 ; r value envelope: p9-p10 =exp. partial strength parameters
amp1 = (aindex*(ar+(1/ar)))/2
afmod oscili amp1, imodfrq, 1 ;FM modulator (sine)
atab = (aindex*(ar-(1/ar)))/2 ;index to table
alook tablei atab, 37 ;table lookup to GEN12
aamod oscili atab, adev, 2 ;am modulator (cosine)
aamod = (exp(alook+aamod))*aenv
acar oscili aamod, afmod+icarfrq, 1 ;AFM (carrier)
asig balance acar, aenv
outs asig, asig

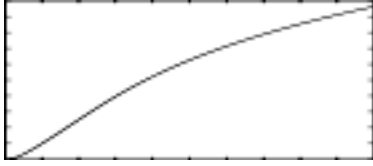
endin

</CsInstruments>
<CsScore>
f 1 0 8192 10 1
f 2 0 8192 9 1 1 90
f37 0 1024 -12 40 ;Bessel function-defined from 0 to 40

i 1 0 2 .2 800 800 1 6 .1 2
i 1 + . . 1900 147 8 1 4 .2
i 1 . . . 1100 380 2 9 .5 2
i 1 . 10 . 100 100 11 3 .2 5
s
i 1 0 1 .1 200 100 1 6 .1 2
i 1 + . < < < < < <
i 1 + . . < < < < < <
i 1 + . . < < < < < <
i 1 + . . < < < < < <
i 1 + . . < < < < < <
i 1 + . . < < < < < <
i 1 + 10 .2 800 800 9 1 .9 6
s
i 1 0 11 .25 50 51 1 6 .1 2
i 1 1 9 .05 700 401 1 6 .1 2
i 1 2 8 . 900 147 8 1 4 .2
i 1 3 7 . 1100 381 2 9 .5 2
i 1 4 6 . 200 102 11 3 .2 5
i 1 5 6 . 800 803 9 1 .9 6
e
</CsScore>
</CsoundSynthesizer>

```

Voici le diagramme de la forme d'onde de la routine GEN12 utilisée dans l'exemple :



f 37 0 1024 -12 40 - fonction de Bessel définie de 0 à 40

## Crédits

L'exemple provient du Csound Book (p. 87) avec de légères modifications.

# GEN13

GEN13 — Mémorise un polynôme dont les coefficients sont dérivés des polynômes de Tchebychev de première espèce.

## Description

Utilise les coefficients de Tchebychev pour générer des fonctions polynomiales stockées qui, dans le waveshaping, peuvent être utilisées pour séparer une sinus en harmoniques selon un spectre prédéfini.

## Syntaxe

```
f # date taille 13 xint xamp h0 h1 h2 ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*). La valeur normale est une puissance-de-2 plus 1.

*xint* -- fournit les valeurs gauches et droites  $[-xint, +xint]$  de l'intervalle des *x* sur lequel le polynôme doit être évalué. *GEN13* et *GEN14* appellent *GEN03* pour évaluer leurs fonctions ; la valeur en *p5* est ainsi étendue en une paire négative-positive *p5*, *p6* avant l'appel de *GEN03*. La valeur normale est 1.

*xamp* -- facteur de pondération de l'amplitude de l'entrée sinusoïdale qui est attendue pour produire le spectre suivant.

*h0*, *h1*, *h2*, etc. -- importance relative des harmoniques 0 (CC), 1 (fondamental), 2 ... qui résulteront quand une sinus d'amplitude

$xamp * \text{int}(\text{taille}/2)/xint$

est traitée en waveshaping avec cette table de fonction. Ces valeurs décrivent ainsi un spectre de fréquences associé à un facteur particulier *xamp* du signal d'entrée.



### Note

*GEN13* est le générateur de fonction normalement employé dans le waveshaping standard. Il stocke un polynôme dont les coefficients dérivent des polynômes de Tchebychev de première espèce, de sorte qu'une sinus d'amplitude *xamp* pilotant le dispositif produise le spectre spécifié en sortie. Noter que l'évolution de ce spectre ne varie généralement pas linéairement en fonction de *xamp*. Cependant, il est à bande limitée (les seuls harmoniques qui apparaissent seront ceux qui auront été spécifiés au moment de la génération) ; et les harmoniques auront tendance à apparaître et à se développer en ordre ascendant (les harmoniques inférieurs dominant pour de faibles *xamp*, et la richesse spectrale augmentant pour des valeurs plus grandes de *xamp*). Une valeur *hn* négative implique une opposition de phase de cet harmonique ; le spectre d'amplitude complet demandé ne sera pas affecté par ce déphasage, bien que l'évolution de plusieurs de ses harmoniques puisse l'être. Le schéma +, +, -, -, +, +, ... pour *h0*, *h1*, *h2*, ... minimisera le problème de la normalisation pour de faibles valeurs de *xamp* (voir ci-dessus), mais ne fournira pas nécessairement le schéma d'évolution le plus lisse.

## Exemples

Voici un exemple de la routine GEN13. Il utilise le fichier *gen13.csd* [examples/gen13.csd].

## Exemple 1210. Exemple de la routine GEN13.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen13.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
;example by Russell Pinkston - Univ. of Texas (but slightly modified)

gisine   ftgen 0, 0, 16384, 10, 1           ;sine wave

instr    1

ihertz = cpspch(p4)
ipkamp = p5
iwsfn = p6           ;waveshaping function
inmfns = p7          ;normalization function
agate   linen 1, .01, p3, .1                ;overall amp envelope
kctrl   linen .99, 2, p3, 2                ;waveshaping index control
aindex  poscil kctrl/2, ihertz, gisine      ;sine wave to be distorted
asignal tablei .5*aindex, iwsfn, 1          ;waveshaping
knormal tablei kctrl, inmfns, 1            ;amplitude normalization
asig    =      asignal*knormal*ipkamp*agate
        outs  asig, asig

endin
</CsInstruments>
<CsScore>
; This proves the statement in Dodge (p. 147) that Chebyshev polynomials
; of order K have "only the kth harmonic." This is only true when the
; waveshaping index is at the maximum - i.e., when the entire transfer
; function is being accessed. RP.
;-----
; quasi sawtooth transfer function:
;   h0  h1  h2  h3  h4  h5  h6   h7  h8  h9  h10  h11  h12  h13  h14  h15  h16  h17
f1 0 513 13 1 1 0 100 -50 -33 25 20 -16.7 -14.2 12.5 11.1 -10 -9.09 8.333 7.69 -7.14 -6.67
f2 0 257 4 1 1 1 ; normalizing function with midpoint bipolar offset

; st dur pch amp wsfn nmfn
i1 0 4 6.00 .7 1 2
i1 4 . 7.00 .
i1 8 . 8.00 .
;-----
; quasi square wave transfer function:
;   h0  h1  h2  h3  h4  h5  h6   h7  h8  h9  h10  h11  h12  h13  h14  h15  h16  h17
f3 0 513 13 1 1 0 100 0 -33 0 20 0 -14.2 0 11.1 0 -9.09 0 7.69 0 -6.67
f4 0 257 4 3 1 1 ; normalizing function with midpoint bipolar offset

; st dur pch amp wsfn nmfn
i1 16 4 6.00 .7 3 4
i1 20 . 7.00 .
i1 24 . 8.00 .
;-----
; quasi triangle wave transfer function:
```

```

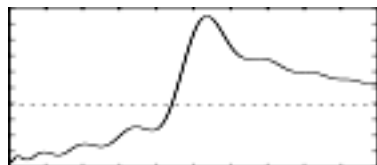
; h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17
f5 0 513 13 1 1 0 100 0 -11.11 0 4 0 -2.04 0 1.23 0 -.826 0 .59 0 -.444
f6 0 257 4 5 1 ; normalizing function with midpoint bipolar offset
; st dur pch amp wsfn nmfn
i1 32 4 6.00 .7 5 6
i1 36 . 7.00 .
i1 40 . 8.00 .
;-----
; transfer function1: h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
f7 0 513 13 1 1 0 1 -.8 0 .6 0 0 0 .4 0 0 0 0 .1 -.2 -.3 .5
f8 0 257 4 7 1 ; normalizing function with midpoint bipolar offset
; st dur pch amp wsfn nmfn
i1 48 4 5.00 .7 7 8
i1 52 . 6.00 .
i1 56 . 7.00 .
;-----
;=====
; This demonstrates the use of high partials, sometimes without a ;
; fundamental, to get quasi-inharmonic spectra from waveshaping. ;
;=====
; transfer function2: h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
f9 0 513 13 1 1 0 0 0 -.1 0 .3 0 -.5 0 .7 0 -.9 0 1 0 -1 0
f10 0 257 4 9 1 ; normalizing function with midpoint bipolar offset
; st dur pch amp wsfn nmfn
i1 64 4 5.00 .7 9 10
i1 68 . 6.00 .
i1 72 . 7.00 .
;-----
; transfer function3: h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14
f11 0 513 13 1 1 0 0 0 0 0 0 0 0 -1 0 1 0 0 0 -.1 0 .1
f12 0 257 4 11 1 ; normalizing function with midpoint bipolar offset

; st dur pch amp wsfn nmfn
i1 80 4 5.00 .7 11 12
i1 84 . 5.06 .
i1 88 . 6.00 .
;-----
;=====
; split a sinusoid into 3 odd-harmonic partials of relative strength 5:3:1
;=====
;-----
; transfer function4: h0 h1 h2 h3 h4 h5
f13 0 513 13 1 1 0 5 0 3 0 1
f14 0 257 4 13 1 ; normalizing function with midpoint bipolar offset

; st dur pch amp wsfn nmfn
i1 96 4 5.00 .7 13 14
i1 100 . 5.06 .
i1 104 . 6.00 .
e
</CsScore>
</CsoundSynthesizer>

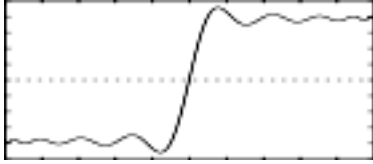
```

Voici les diagrammes des formes d'onde des routines GEN13 utilisées dans l'exemple :

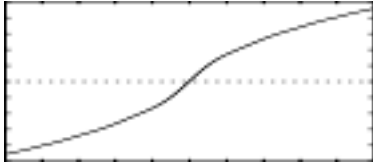


f1 0 513 13 1 1 0 100 -50 -33 25 20 -16.7 -14.2 12.5 11.1 -10 -9.09 8.333 7.69 -7.14 -6.67 6.25 5.88 -5.55  
-5.26 5 - fonction de transfert quasi dent de scie

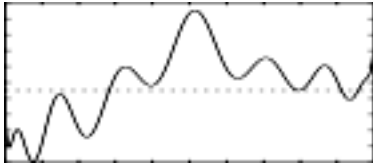




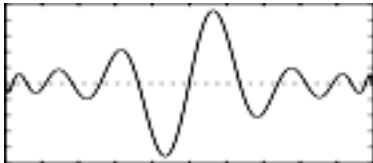
f3 0 513 13 1 1 0 100 0 -33 0 20 0 -14.2 0 11.1 0 -9.09 0 7.69 0 -6.67 0 5.88 0 -5.26 - fonction de transfert quasi carrée



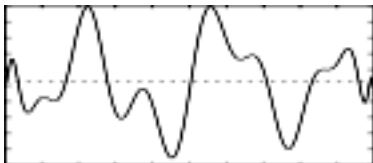
f5 0 513 13 1 1 0 100 0 -11.11 0 4 0 -2.04 0 1.23 0 -.826 0 .59 0 -.444 0 .346 0 -.277 - fonction de transfert quasi triangulaire



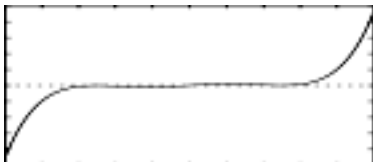
f7 0 513 13 1 1 0 1 -.8 0 .6 0 0 0 .4 0 0 0 0 .1 -.2 -.3 .5 - fonction de transfert 1



f9 0 513 13 1 1 0 0 0 -.1 0 .3 0 -.5 0 .7 0 -.9 0 1 0 -1 0 - fonction de transfert 2



f11 0 513 13 1 1 0 0 0 0 0 0 0 -1 0 1 0 0 -.1 0 .1 0 -.2 .3 0 -.7 0 .2 0 -.1 - fonction de transfert 3



f13 0 513 13 1 1 0 5 0 3 0 1 - divise une sinusoïde en trois partiels harmoniques impaires d'importance relative 5:3:1

## Voir aussi

*GEN03*, *GEN14* et *GEN15*.

Information au sujet des polynômes de Tchebychev sur Wikipedia : [http://fr.wikipedia.org/wiki/Polynôme\\_de\\_Tchebychev](http://fr.wikipedia.org/wiki/Polynôme_de_Tchebychev) [[http://fr.wikipedia.org/wiki/Polyn%C3%B4me\\_de\\_Tchebychev](http://fr.wikipedia.org/wiki/Polyn%C3%B4me_de_Tchebychev)]

# GEN14

GEN14 — Mémorise un polynôme dont les coefficients sont dérivés des polynômes de Tchebychev de seconde espèce.

## Description

Utilise les coefficients de Tchebychev pour générer des fonctions polynomiales stockées qui, dans le waveshaping, peuvent être utilisées pour séparer une sinus en harmoniques selon un spectre prédéfini.

## Syntaxe

```
f # date taille 14 xint xamp h0 h1 h2 ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*). La valeur normale est une puissance-de-2 plus 1.

*xint* -- fournit les valeurs gauches et droites  $[-xint, +xint]$  de l'intervalle des *x* sur lequel le polynôme doit être évalué. *GEN13* et *GEN14* appellent *GEN03* pour évaluer leurs fonctions ; la valeur en *p5* est ainsi étendue en une paire négative-positive *p5*, *p6* avant l'appel de *GEN03*. La valeur normale est 1.

*xamp* -- facteur de pondération de l'amplitude de l'entrée sinusoïdale qui est attendue pour produire le spectre suivant.

*h0*, *h1*, *h2*, etc. -- importance relative des harmoniques 0 (CC), 1 (fondamental), 2 ... qui résulteront quand une sinus d'amplitude

$xamp * \text{int}(\text{taille}/2)/xint$

est traitée en waveshaping avec cette table de fonction. Ces valeurs décrivent ainsi un spectre de fréquences associé à un facteur particulier *xamp* du signal d'entrée.



### Note

- *GEN13* est le générateur de fonction normalement employé dans le waveshaping standard. Il stocke un polynôme dont les coefficients dérivent des polynômes de Tchebychev de première espèce, de sorte qu'une sinus d'amplitude *xamp* pilotant le dispositif produise le spectre spécifié en sortie. Noter que l'évolution de ce spectre ne varie généralement pas linéairement en fonction de *xamp*. Cependant, il est à bande limitée (les seuls harmoniques qui apparaissent seront ceux qui auront été spécifiés au moment de la génération) ; et les harmoniques auront tendance à apparaître et à se développer en ordre ascendant (les harmoniques inférieurs dominant pour de faibles *xamp*, et la richesse spectrale augmentant pour des valeurs plus grandes de *xamp*). Une valeur *hn* négative implique une opposition de phase de cet harmonique ; le spectre d'amplitude complet demandé ne sera pas affecté par ce déphasage, bien que l'évolution de plusieurs de ses harmoniques puisse l'être. Le schéma +, +, -, -, +, +, ... pour *h0*, *h1*, *h2*, ... minimisera le problème de la normalisation pour de faibles valeurs de *xamp* (voir ci-dessus), mais ne fournira pas nécessairement le schéma d'évolution le plus lisse.

- *GEN14* stocke un polynôme dont les coefficients dérivent de polynômes de Tchebychev de seconde espèce.

## Exemples

Voici un exemple de la routine GEN14. Il utilise le fichier *gen14.csd* [examples/gen14.csd].

### Exemple 1211. Exemple de la routine GEN14.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac ;;realtime audio out
;-iadc ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen14.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
;after the example from The Csound Book, page 83

instr 1 ;compare results from GEN13 & GEN14

iwshpfn = p6
inrmfn = p7
aswp linseg 0.01, p3*.5, .49, p3*.5, 0.01 ;index sweep function
aindex poscil aswp, p5, 2 ;sound to waveshape
atable tablei aindex, iwshpfn, 1, .5 ;waveshape index
anrm tablei aswp*2, inrmfn, 1 ;normalization
aenv linen p4, .01, p3, .02 ;amplitude envelope
asig = (atable*anrm)*aenv ;normalize and impose envelope
asig dcblock2 asig ;get rid of DC
outs asig, asig

endin
</CsInstruments>
<CsScore>

f 2 0 8192 10 1 ;sine wave

f 28 0 4097 13 1 1 1 0 .8 0 .5 0 .2 ;waveshaping function: GEN13 - odd harmonics
f 280 0 2049 4 28 1 ;normalization function for f28
f 29 0 4097 14 1 1 1 0 .8 0 .5 0 .2 ;waveshaping function: GEN14 - same harmonics
f 290 0 2049 4 29 1 ;normalization function for f29

f 30 0 4097 13 1 1 0 1 0 .6 0 .4 0 .1 ;waveshaping function: GEN13 - even harmonics
f 301 0 2049 4 30 1 ;normalization function for f30
f 31 0 4097 14 1 1 0 1 0 .6 0 .4 0 .1 ;waveshaping function: GEN13 - even harmonics
f 310 0 2049 4 31 1 ;normalization function for f31
s
i1 0 3 .7 440 28 280
i1 4 . .7 . 29 290
i1 8 . .7 . 30 301
i1 12 3 .7 . 31 310
```

```
e  
</CsScore>  
</CsoundSynthesizer>
```

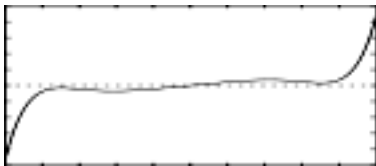
Voici les diagrammes des formes d'onde des routines GEN14 utilisées dans l'exemple :



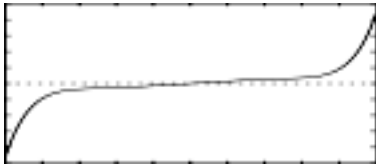
f28 0 4097 13 1 1 1 0 .8 0 .5 0 .2 - fonction de distorsion non linéaire : GEN13, harmoniques impairs



f29 0 4097 14 1 1 1 1 0 .8 0 .5 0 .2 - fonction de distorsion non linéaire : GEN14, les mêmes harmoniques impairs



f30 0 4097 13 1 1 1 0 1 0 .6 0 .4 0 .1 - fonction de distorsion non linéaire : GEN13, harmoniques pairs



f31 0 4097 14 1 1 1 0 1 0 .6 0 .4 0 .1 - fonction de distorsion non linéaire : GEN14, les mêmes harmoniques pairs

## Voir aussi

*GEN03*, *GEN13* et *GEN15*.

Information au sujet des polynômes de Tchebychev sur Wikipedia : [http://fr.wikipedia.org/wiki/Polynôme\\_de\\_Tchebychev](http://fr.wikipedia.org/wiki/Polynôme_de_Tchebychev) [[http://fr.wikipedia.org/wiki/Polyn%C3%B4me\\_de\\_Tchebychev](http://fr.wikipedia.org/wiki/Polyn%C3%B4me_de_Tchebychev)]

# GEN15

GEN15 — Crée deux tables de fonctions polynomiales mémorisées.

## Description

Ce sous-programme crée deux tables de fonctions polynomiales mémorisées, appropriées pour une utilisation en quadrature de phase.

## Syntaxe

```
f # date taille 15 xint xamp h0 phs0 h1 phs1 h2 phs2 ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*). La valeur normale est une puissance-de-2 plus 1.

*xint* -- fournit les valeurs gauches et droites  $[-xint, +xint]$  de l'intervalle des  $x$  sur lequel le polynôme doit être évalué. Ce sous-programme appellera éventuellement *GEN03* pour évaluer les deux fonctions ; la valeur en  $p5$  est alors étendue en une paire négative-positive  $p5, p6$  avant l'appel de *GEN03*. La valeur normale est 1.

*xamp* -- facteur de pondération de l'amplitude de l'entrée sinusoïdale qui est attendue pour produire le spectre suivant.

*h0, h1, h2, ..., hn* -- importance relative des harmoniques 0 (CC), 1 (fondamental), 2 ... qui résulteront quand une sinus d'amplitude

$xamp * \text{int}(\text{taille}/2)/xint$

est traitée en waveshaping avec cette table de fonction. Ces valeurs décrivent ainsi un spectre de fréquences associé à un facteur particulier *xamp* du signal d'entrée.

*phs0, phs1, ...* -- phase en degrés des harmoniques désirés *h0, h1, ...* lorsque les deux fonctions de *GEN15* sont utilisées en quadrature de phase.



## Notes

*GEN15* crée deux tables de même taille, étiquetées  $f\#$  et  $f\# + 1$ . La table  $\#$  contiendra une fonction de Tchebychev de première espèce, évaluée par *GEN13* avec des harmoniques d'amplitude  $h0\cos(phs0), h1\cos(phs1), \dots$ . Table  $\# + 1$  contiendra une fonction de Tchebychev de deuxième espèce, évaluée par *GEN14* avec les harmoniques  $h1\sin(phs1), h2\sin(phs2), \dots$  (noter le déplacement harmonique). Les deux tables peuvent être utilisées en conjonction dans un réseau de waveshaping qui exploite la quadrature de phase.

Avant la version 5.16 il y avait un bogue (signalé par Menno Knevel et corrigé par François Pinot) sur le nombre de p-champs transmis à *GEN13* et à *GEN14* par *GEN15*. En conséquence, tous les fichiers csd, ou orc et sco qui utilisaient *GEN15* avant la correction du bogue, donneront probablement un résultat différent maintenant.

## Exemples

Voici un exemple de la routine *GEN15*. Il utilise le fichier *gen15.csd* [examples/gen15.csd].

## Exemple 1212. Exemple de la routine GEN15.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen15.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
;example from the Csound Book, page 85
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
```

```
instr 1

idur = p3
iamp = p4
ifrq = cpspch(p5)      ;pitch
iswp1 = p6
iswp2 = p7
kswp line iswp1, p3, iswp2      ;amplitude sweep values
acosi oscili kswp*.5, ifrq, 2    ;f2=cosine wave
asine oscili kswp, ifrq, 1      ;f1=sine wave
atab1 tablei acosi, 33, 1, .5    ;tables a1 to GEN13
atab2 tablei acosi, 34, 1, .5    ;tables a1 to GEN14
knrm1 tablei kswp, 35, 1        ;normalizing f35
knrm2 tablei kswp, 36, 1        ;normalizing f36
anrm1 = atab1*knrm1            ;normalize GEN13 signal
anrm2 = atab2*knrm2*asine      ;normalize GEN14 signal
amix = anrm1+anrm2            ;mix GEN13 and GEN14
kenv expseg .001, idur*.1, iamp, idur*.1, iamp*.8, idur*.8, .001
asig = amix*kenv
outs asig, asig
```

```
endin
```

```
</CsInstruments>
<CsScore>
f 1 0 8193 10 1      ;sine wave
f 2 0 8193 9 1 1 90  ;cosine wave
```

```
; Note that all the f33 tables in the following sections are defined with p4=-15,
; which means that tables 33 and 34 will not be normalized. Thus if we display
; tables when running this example, we'll get correct diagrams even if one table
; has very small values instead of 0 values, due to cpu approximations in processing
; sin(180), as in sections 2, 4, and 5. This has no consequence on the audio result,
; because of the use of amp normalization (tables 35 and 36).
```

```
f 33 0 8193 -15 1 1 1 0 1 180 .8 45 .6 270 .5 90 .4 225 .2 135 .1 315 ;makes function tables 33 and 34
f 35 0 4097 4 33 1      ;amp normalization for f33
f 36 0 4097 4 34 1      ;amp normalization for f34
i 1 0 5 .6 8.00 0 1
i 1 + . .6 8.00 1 0
s
;even harmonics with no phase shift, odd harmonics with phase shift
f 33 0 8193 -15 1 1 1 0 1 0 1 180 1 180 1 0 1 0 1 180 1 180 1 0 1 0 1 180 1 180
f 35 0 4097 4 33 1      ;amp normalization for f33
f 36 0 4097 4 34 1      ;amp normalization for f34
i 1 0 5 .6 8.00 0 1
i 1 + . .6 8.00 1 0
s
```

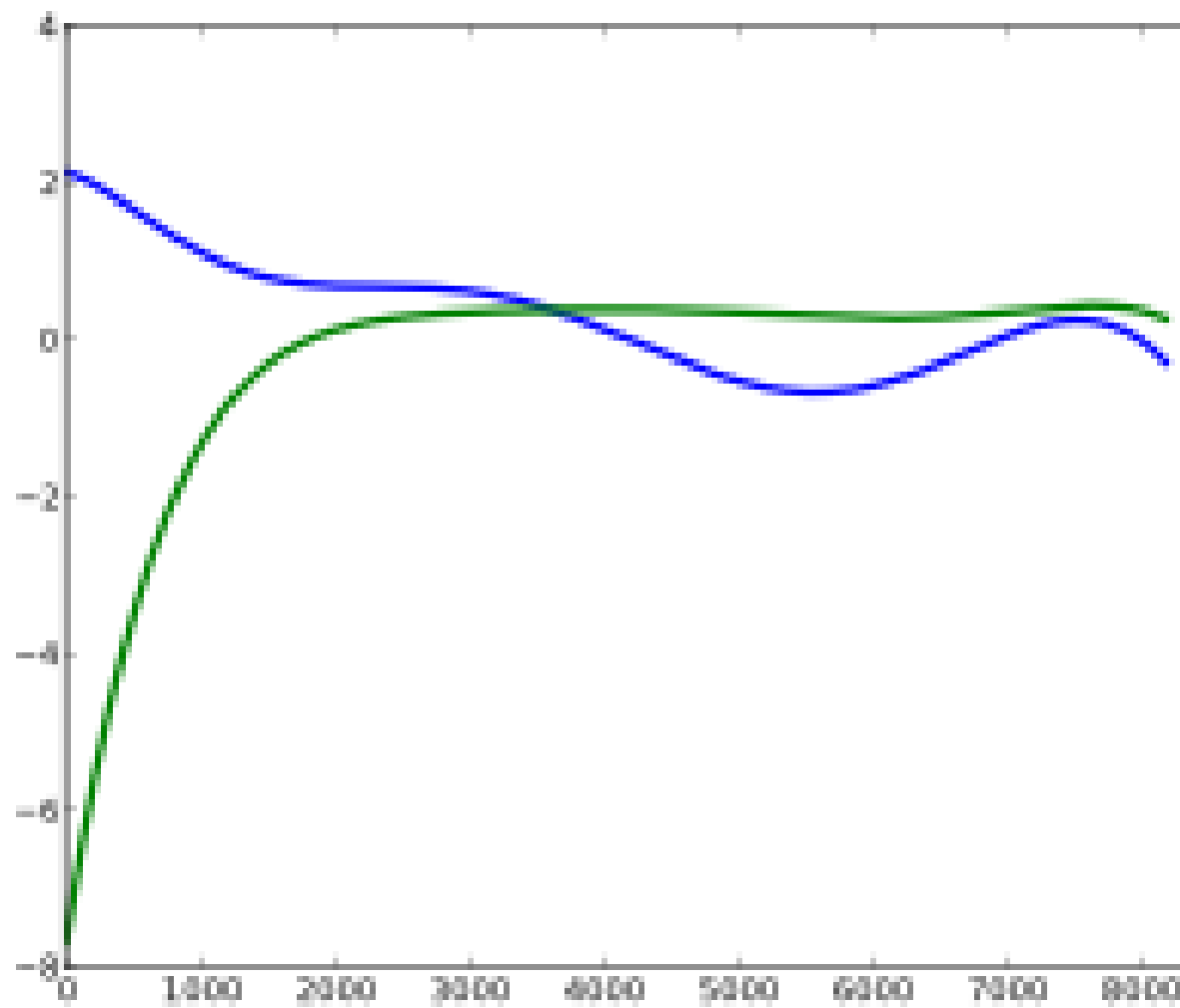
```

;different harmonic strenghts and phases
f 33 0 8193 -15 1 1 1 0 1 0 .9 180 .5 270 .75 90 .4 45 .2 225 .1 0
f 35 0 4097 4 33 1 ;amp normalization for f33
f 36 0 4097 4 34 1 ;amp normalization for f34
i 1 0 5 .6 8.00 0 1
i 1 + . .6 8.00 1 0
s
;lower harmonics no phase shift, upper harmonics with phase shift
f 33 0 8193 -15 1 1 1 0 1 0 .5 0 .9 0 .3 0 .75 0 .2 180 .6 180 .15 180 .5 180 .1 180
f 35 0 4097 4 33 1 ;amp normalization for f33
f 36 0 4097 4 34 1 ;amp normalization for f34
i 1 0 5 .6 8.00 0 1
i 1 + . .6 8.00 1 0

s
;lower harmonics with phase shift, upper harmonics no phase shift
f 33 0 8193 -15 1 1 1 180 1 180 .5 180 .9 180 .3 180 .75 180 .2 0 .6 0 .15 0 .5 0 .1 0
f 35 0 4097 4 33 1 ;amp normalization for f33
f 36 0 4097 4 34 1 ;amp normalization for f34
i 1 0 5 .6 8.00 0 1
i 1 + . .6 8.00 1 0
e
</CsScore>
</CsoundSynthesizer>

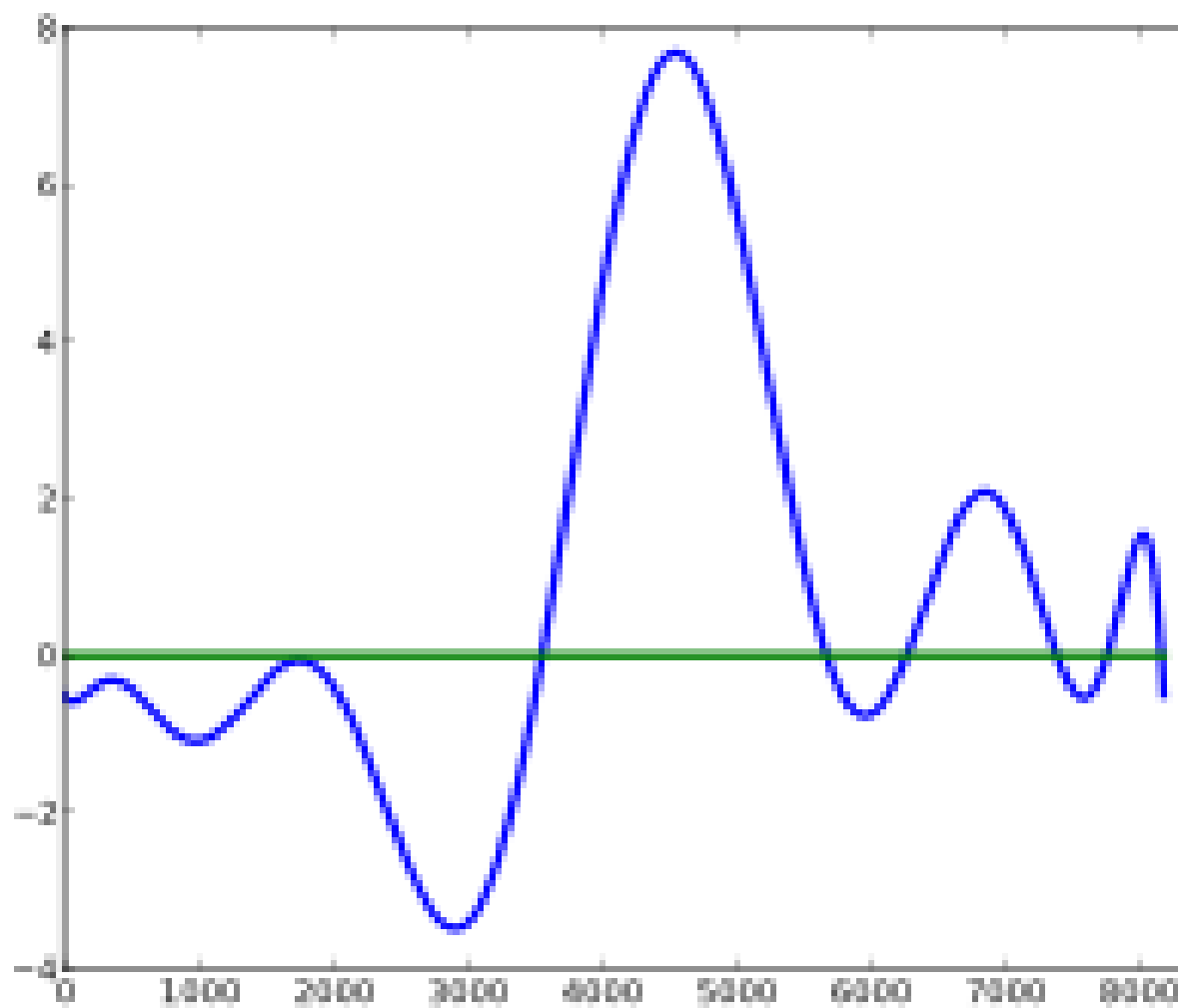
```

Voici les diagrammes des formes d'onde de la routine GEN15 utilisées dans l'exemple (dans chaque diagramme, la courbe en bleu représente la table 33 et la courbe en vert représente la table 34) :

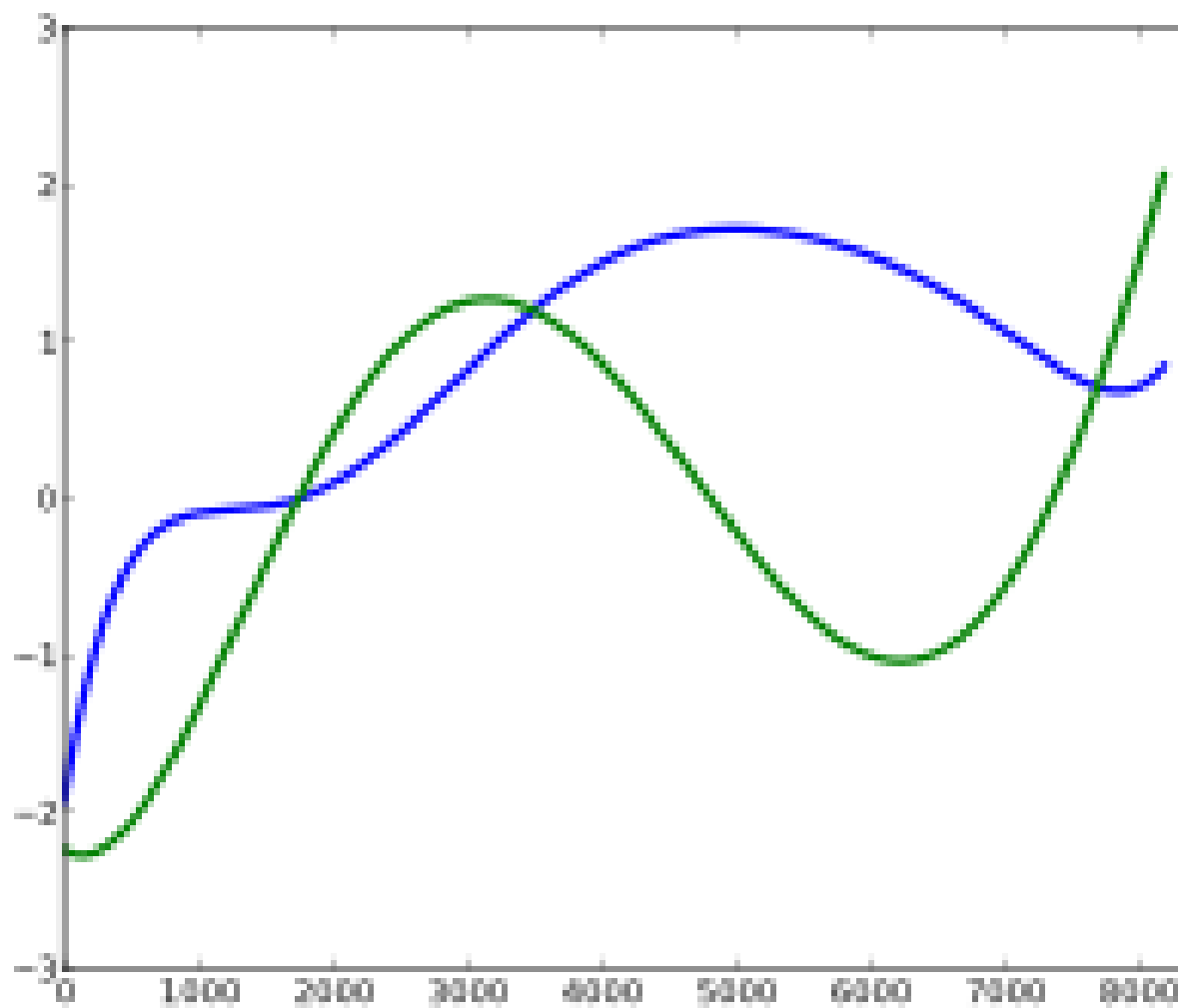


f 33 0 8193 -15 1 1 1 0 1 180 .8 45 .6 270 .5 90 .4 225 .2 135 .1 315

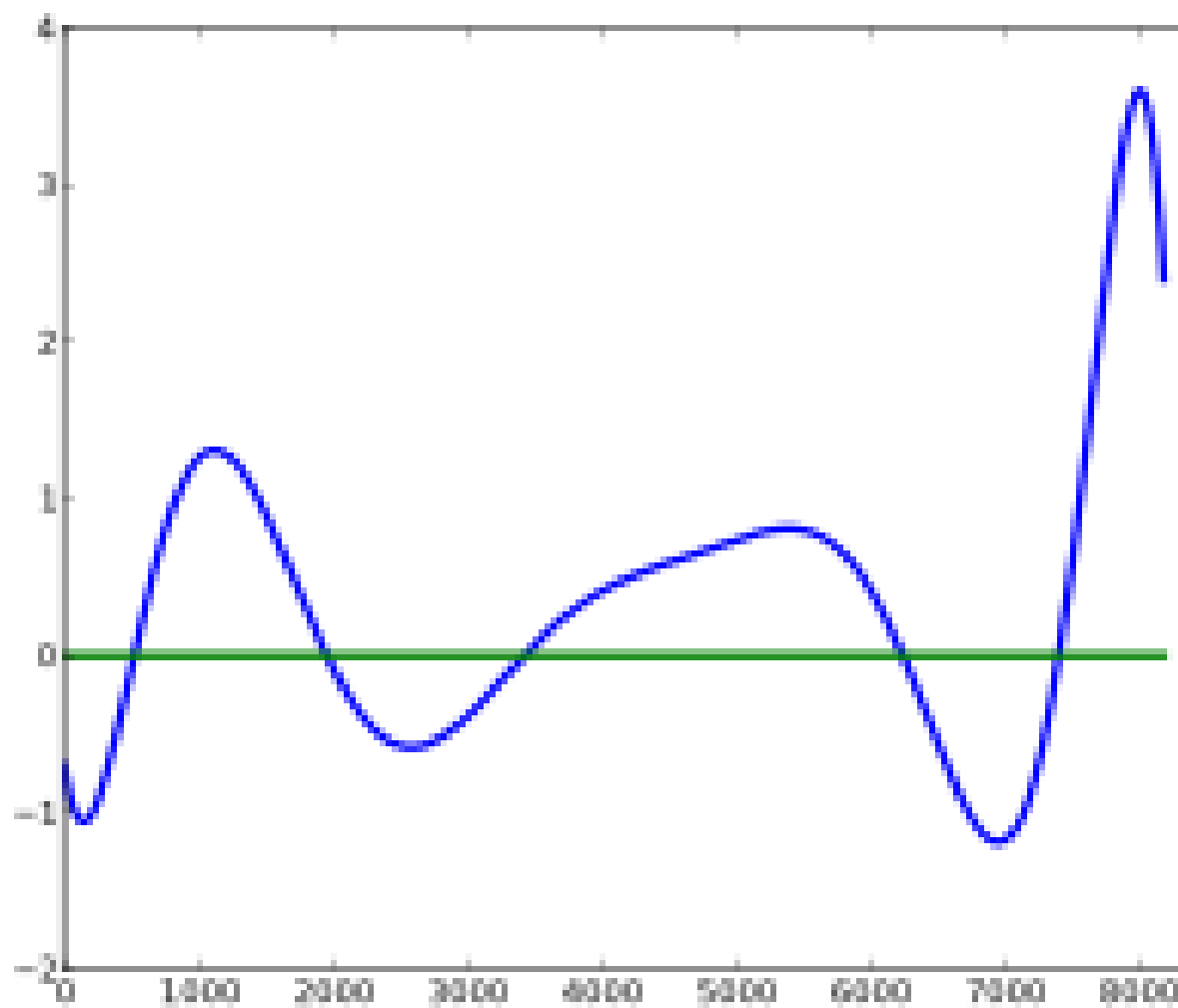




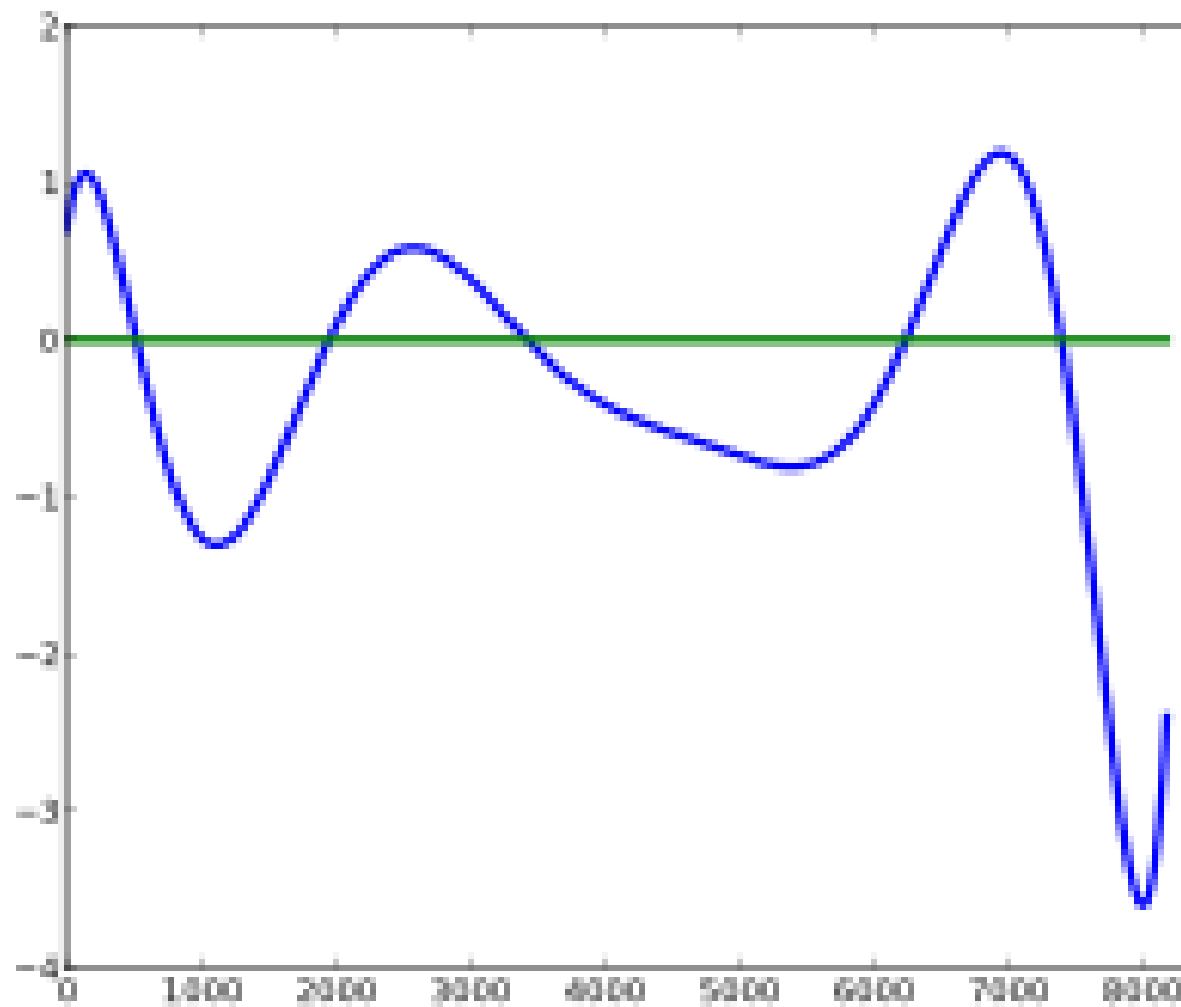
f 33 0 8193 -15 1 1 1 0 1 0 1 180 1 180 1 0 1 0 1 180 1 180 1 0 1 0 1 180 1 180



f 33 0 8193 -15 1 1 1 0 1 0 .9 180 .5 270 .75 90 .4 45 .2 225 .1 0



f 33 0 8193 -15 1 1 1 0 1 0.5 0.9 0.3 0.75 0.2 180 .6 180 .15 180 .5 180 .1 180



f 33 0 8193 -15 1 1 1 180 1 180 .5 180 .9 180 .3 180 .75 180 .2 0 .6 0 .15 0 .5 0 .1 0

## Voir aussi

*GEN13* et *GEN14*.

# GEN16

GEN16 — Crée une table depuis une valeur initiale jusqu'à une valeur terminale.

## Description

Crée une table depuis la valeur *deb* jusqu'à la valeur *fin* en *dur* pas.

## Syntaxe

```
f # date taille 16 val1 dur1 type1 val2 [dur2 type2 val3 ... typeX valN]
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*). La valeur normale est une puissance-de-2 plus 1.



### Note

La dernière valeur n'est atteinte que lorsque la longueur de la table est une puissance-de-2 plus 1. (Cette longueur est cruciale dans l'utilisation de l'opcode *tab*).

*deb* -- valeur de départ

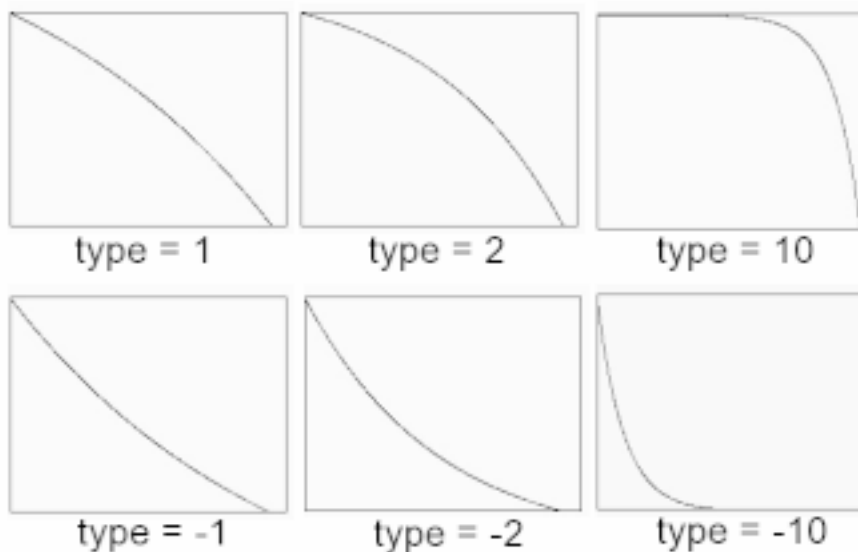
*dur* -- nombre de segments

*type* -- si 0, une ligne droite est produite. Si différent de zéro, alors *GEN16* crée la courbe suivante sur *dur* pas :

$$deb + (fin - deb) * (1 - \exp(i * type / (dur - 1))) / (1 - \exp(type))$$

*fin* -- valeur après *dur* segments

Voici quelques exemples de courbes générées pour différentes valeurs de *type* :



Tables générées par GEN16 pour différentes valeurs de *type*.



## Note

Si *type* > 0, on a une courbe montant lentement (concave) ou décroissant lentement (convexe), tandis que si *type* < 0, la courbe monte rapidement (convexe) ou décroît rapidement (concave). Voir aussi *transeg*.

## Exemple 1213. Un exemple simple de la routine GEN16.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out   Audio in
-odac          -iadc      ;;RT audio I/O
; For Non-realtime output leave only the line below:
; -o gen16.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 1

instr 1
  kcps init 1/p3
  kndx phasor kcps

  ifn = p4
  ixmode = 1
  kval table kndx, ifn, ixmode

  ibasefreq = 440
  kfreq = kval * ibasefreq
  a1 oscil 20000, ibasefreq + kfreq, 1
  out a1
endin

</CsInstruments>
<CsScore>

f 1 0 16384 10 1

f 2 0 1024 16 1 1024 1 0
f 3 0 1024 16 1 1024 2 0
f 4 0 1024 16 1 1024 10 0
f 5 0 1024 16 1 1024 -1 0
f 6 0 1024 16 1 1024 -2 0
f 7 0 1024 16 1 1024 -10 0

i 1 0 2 2
i 1 + . 3
i 1 + . 4
i 1 + . 5
i 1 + . 6
i 1 + . 7

e
```

```
</CsScore>  
</CsoundSynthesizer>
```

## Crédits

Auteur : John ffitch  
University of Bath, Codemist. Ltd.  
Bath, UK  
Octobre 2000

Nouveau dans la version 4.09 de Csound

# GEN17

GEN17 — Crée une fonction en escalier à partir des paires x-y données.

## Description

Ce sous-programme crée une fonction en escalier à partir des paires x-y données.

## Syntaxe

```
f # date taille 17 x1 a x2 b x3 c ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*). La valeur normale est une puissance-de-2 plus 1.

*x1*, *x2*, *x3*, etc. -- valeurs d'abscisse x, en ordre ascendant, commençant par 0.

*a*, *b*, *c*, etc. -- valeurs y à ces valeurs d'abscisse x, maintenues jusqu'à la valeur d'abscisse x suivante.



### Note

Ce sous-programme crée une fonction en escalier de paires x-y dont les valeurs y sont maintenues vers la droite. La valeur de y la plus à droite est ensuite maintenue jusqu'à la fin de la table. Cette fonction est utile pour mettre en correspondance un ensemble de données avec un autre, tel que des numéros de notes MIDI avec des numéros de ftables de sons échantillonnés. (voir *loscil*).

## Exemples

```
f 1 0 128 -17 0 1 12 2 24 3 36 4 48 5 60 6 72 7 84 8
```

Ceci décrit une fonction en escalier avec huit niveaux croissants successifs, chacun occupant 12 positions sauf pour le dernier qui étend sa valeur jusqu'à la fin de la table. La normalisation est empêchée. En indexant cette table avec un numéro de note MIDI, on retrouvera une valeur différente pour chaque octave jusqu'à la huitième, au-delà de laquelle la valeur retournée restera la même.

Voici un exemple complet de la routine GEN17. Il utilise le fichier *gen17.csd* [examples/gen17.csd].

### Exemple 1214. Un exemple de la routine GEN17.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -M0 -+rtmidi=virtual    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen17.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```



```

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

inote cpsmidi
iveloc ampmidi .5
ictl midictrl 5 ;move slider of controller 5 to change ftable
itab table ictl, 2
aout poscil iveloc, inote, itab
outs aout, aout

endin
</CsInstruments>
<CsScore>
f 1 0 8193 10 1
f 2 0 128 -17 0 10 32 20 64 30 96 40 ;inhibit rescaling

f 10 0 16384 10 1 ; Sine
f 20 0 16384 10 1 0.5 0.3 0.25 0.2 0.167 0.14 0.125 .111; Sawtooth
f 30 0 16384 10 1 0 0.3 0 0.2 0 0.14 0 .111; Square
f 40 0 16384 10 1 1 1 0.7 0.5 0.3 0.1 ; Pulse

f 0 30 ;run for 30 seconds
e
</CsScore>
</CsoundSynthesizer>

```

Voici le diagramme de la forme d'onde de la routine GEN17 utilisée dans l'exemple :



f 2 0 128 -17 0 10 32 20 64 30 96 40 - une fonction en escalier avec quatre niveaux égaux, chacun ayant une largeur de 32 positions, sauf le dernier qui va jusqu'à la fin de la table

## Voir aussi

GEN02

# GEN18

GEN18 — Ecrit des formes d'onde complexes construites à partir de formes d'ondes déjà existantes.

## Description

Ecrit des formes d'onde complexes construites à partir de formes d'ondes déjà existantes. Chaque forme d'onde utilisée nécessite 4 p-champs et peut se chevaucher avec les autres formes d'onde.

## Syntaxe

```
f # date taille 18 fna ampa debuta fina fnb ampb debutb finb ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de deux (voir l'instruction f).

*fna, fnb, etc.* -- numéros des tables pré-existantes à écrire dans la table.

*ampa, ampb, etc.* -- amplitude des formes d'onde. Ces amplitudes sont relatives, car la forme d'onde composée pourra être post-normalisée. Des valeurs négatives sont autorisées et impliquent une opposition de phase.

*debuta, debutb, etc.* -- où commencer à écrire fn dans la table.

*fina, finb, etc.* -- où terminer l'écriture de fn dans la table. La dernière position valable est la puissance de deux moins un.

## Exemples

```
f 1 0 4096 10 1
f 2 0 1024 18 1 1 0 511 1 1 512 1023
```

f2 est constitué de deux copies de f1 positionnées en 0-511 et 512-1023.

Voici un exemple de la routine GEN18. Il utilise le fichier *gen18.csd* [examples/gen18.csd].

### Exemple 1215. Un exemple de la routine GEN18.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen18.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
```

```
instr 1

ifn = p4
ilen = ftlen(ifn)
aphase phasor 220
asig tablei aphase*ilen, ifn
outs asig, asig

endin
</CsInstruments>
<CsScore>

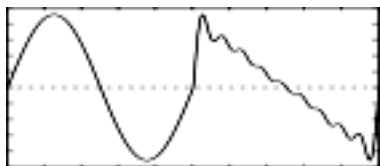
f 1 0 4096 10 1 ;sine
f 2 0 4096 10 1 0.5 0.3 0.25 0.2 0.167 0.14 0.125 .111 ;sawtooth
f 3 0 4096 11 10 5 2 ;cosine partials

f 11 0 8192 18 1 1 0 4095 2 1 4096 8191 ;sine+sawtooth
f 12 0 8192 18 1 1 0 4095 3 1 4096 8191 ;sine+cosine partials
f 13 0 1024 18 1 0.7 0 767 3 0.7 512 1023 ;sine+cosine partials overlapped, shorter table

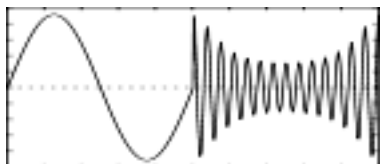
i 1 0 2 2 ;play sawtooth
i 1 + 2 3 ;then cosine partials
i 1 5 2 11 ;now sine+sawtooth
i 1 + 2 12 ;and sine+cosine partials
i 1 + 2 13 ;and sine+cosine partials overlapped, shorter table

e
</CsScore>
</CsoundSynthesizer>
```

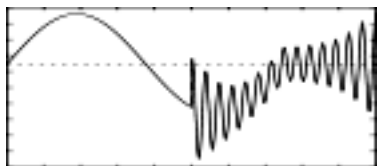
Voici les diagrammes des formes d'onde générées par GEN18, utilisées dans l'exemple :



f 11 0 8193 18 1 1 0 4096 2 1 4097 8192 - forme d'onde composite faite d'une sinus et d'une dent de scie



f 12 0 8192 18 1 1 0 4096 3 1 4097 8192 - forme d'onde composite faite d'une sinus et d'une onde de partiels cosinus



f 13 0 1024 18 1 0.7 0 767 3 0.7 512 1023 - partiels sinus+cosinus se chevauchant, avec une table plus courte que f12

## Noms anciennement utilisés

GEN18 était appelé GEN22 dans la version 4.18. Le nom fut changé à cause d'un conflit avec DirectCsound.

## Crédits

Auteur : William « Pete » Moss  
University of Texas at Austin  
Austin, Texas USA  
Janvier 2002

Nouveau dans la version 4.18, changé dans la version 4.19

# GEN19

GEN19 — Génère des formes d'ondes complexes obtenues par une somme pondérée de sinus.

## Description

Ce sous-programme génère des formes d'ondes complexes obtenues par une somme pondérée de sinus. La spécification de chaque partiel nécessite 4 p-champs dans *GEN19*.

## Syntaxe

```
f # date taille 19 pna ampa phsa dcoa pnb ampb phsb dcob ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction f).

*pna*, *pnb*, etc. -- numéro de partiel (relativement à un fondamental qui occuperait *taille* positions par période) de sinus a, sinus b, etc. Doit être positif, mais pas nécessairement un nombre entier, c'est-à-dire que des partiels non harmoniques sont autorisés. Les partiels peuvent être dans n'importe quel ordre.

*ampa*, *ampb*, etc. -- amplitude des partiels *pna*, *pnb*, etc. Ces amplitudes sont relatives, car la forme d'onde composée peut être normalisée plus tard. Des valeurs négatives sont autorisées et impliquent une opposition de phase.

*phsa*, *phsb*, etc. -- phase initiale des partiels *pna*, *pnb*, etc., exprimée en degrés.

*dcoa*, *dcob*, etc. -- Décalage CC (Composante Continue) des partiels *pna*, *pnb*, etc. Il est appliqué *après* l'amplitude, c'est-à-dire qu'une valeur de 2 montera une sinus d'amplitude 2 de l'intervalle [-2,2] à l'intervalle [0,4] (avant la normalisation finale).



### Note

- Ces sous-programmes génèrent des fonctions stockées comme sommes de sinus de différentes fréquences. Les deux restrictions majeures de *GEN10* qui sont des partiels harmoniques et en phase ne s'appliquent pas à *GEN09* ou à *GEN19*.
- Dans chaque cas l'onde composée, une fois évaluée, est ensuite normalisée à l'unité si p4 est positif. Un p4 négatif empêchera cette opération.

## Exemples

Voici un exemple de la routine GEN19. Il utilise le fichier *gen19.csd* [exemples/gen19.csd].

### Exemple 1216. Un exemple de la routine GEN19.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;;realtime audio out
```

```

;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen19.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

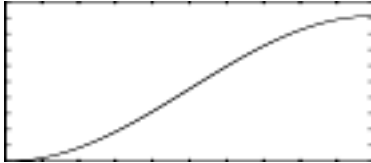
;p4 = transposition factor
;p5 = speed factor
;p6 = function table for grain data
i1 = sr/ftlen(1) ;scaling to reflect sample rate and table length
a1 phasor i1*p5 ;index for speed
asigl fog .5, 15, p4, a1, 1, 0, .01, .5, .01, 30, 1, 2, p3 ;left channel
asigr fog .4, 25, p4+.2, a1, 1, 0, .01, .5, .01, 30, 1, 2, p3, .5 ;right channel
outs asigl, asigr
endin

</CsInstruments>
<CsScore>
f 1 0 131072 1 "fox.wav" 0 0 0
f 2 0 1024 19 .5 .5 270 .5

i 1 0 10 .7 .1
i 1 + 4 1.2 2
e
</CsScore>
</CsoundSynthesizer>

```

Voici le diagramme de la forme d'onde générée par GEN19, utilisé dans l'exemple :



f 2 0 1024 19 .5 .5 270 .5 - une sigmoïde montante

## Voir aussi

GEN09 et GEN10

# GEN20

GEN20 — Génère les fonctions de différentes fenêtres.

## Description

Ce sous-programme génère les fonctions de différentes fenêtres. Ces fenêtres sont utilisées habituellement pour l'analyse spectrale ou pour des enveloppes de grain.

## Syntaxe

```
f # date taille 20 fenêtre max [opt]
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ( + 1).

*fenêtre* -- Type de la fenêtre à générer :

- 1 = Hamming
- 2 = Hanning
- 3 = Bartlett (triangle)
- 4 = Blackman (3-termes)
- 5 = Blackman - Harris (4-termes)
- 6 = Gaussienne
- 7 = Kaiser
- 8 = Rectangle
- 9 = Sync

*max* -- Pour p4 négatif ce sera la valeur absolue au pic de la fenêtre. Si p4 est positif ou si p4 est négatif et p6 est absent la table sera post-normalisée à une valeur maximale de 1.

*opt* -- Argument facultatif nécessaire pour la fenêtre gaussienne et pour la fenêtre de Kaiser.

## Exemples

```
f      1      0      1024      20      5
```

Crée une fonction qui contient une fenêtre de Blackman - Harris à 4 termes avec une valeur maximale de 1.

```
f      1      0      1024     -20      2      456
```

Crée une fonction qui contient une fenêtre de Hanning avec une valeur maximale de 456.

```
f      1      0      1024      -20      1
```

Crée une fonction qui contient une fenêtre de Hamming avec une valeur maximale de 1.

```
f      1      0      1024      20      7      1      2
```

Crée une fonction qui contient une fenêtre de Kaiser avec une valeur maximale de 1. L'argument supplémentaire spécifie comment la fenêtre est "ouverte", par exemple une valeur de 0 donne une fenêtre rectangulaire et une valeur de 10 donne une fenêtre semblable à une fenêtre de Hamming.

```
f      1      0      1024      20      6      1      2
```

Crée une fonction qui contient une fenêtre gaussienne avec une valeur maximale de 1. L'argument supplémentaire spécifie la largeur de la fenêtre, comme l'écart type de la courbe ; dans cette exemple l'écart type vaut 2. La valeur par défaut est 1.

Pour tous les diagrammes, voir les *Fonctions fenêtre*

Voici un exemple de la routine GEN20. Il utilise le fichier *gen20.csd* [examples/gen20.csd].

### Exemple 1217. Exemple de la routine GEN20.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen20.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

insnd = 10      ;"fox.wav"
ibasfrq = 44100 / ftlen(insnd) ;use original sample rate of insnd file

kamp expseg .001, p3/2, .7, p3/2, .8 ;envelope
kpitch line ibasfrq, p3, ibasfrq * .8
kdens line 600, p3, 10
kaoff line 0, p3, .1
kpoff line 0, p3, ibasfrq * .5
kgdur line .04, p3, .001 ;shorten duration of grain during note
imaxgdur = .5
igfn = p4      ;different windows
asigL grain kamp, kpitch, kdens, kaoff, kpoff, kgdur, insnd, igfn, imaxgdur, 0.0
asigR grain kamp, kpitch, kdens, kaoff, kpoff, kgdur, insnd, igfn, imaxgdur, 0.0
outs asigL, asigR

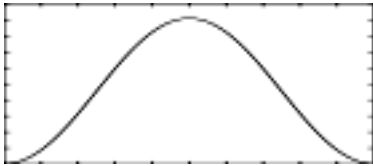
endin
</CsInstruments>
<CsScore>
```



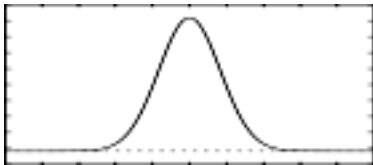
```
f1 0 512 20 2 ;Hanning window
f2 0 512 20 6 1 ;Gaussian window
f10 0 16384 1 "fox.wav" 0 0 0

i1 0 5 1 ;use Hanning window
i1 + 5 2 ;use Gaussian window
e
</CsScore>
</CsoundSynthesizer>
```

Voici les diagrammes des formes d'onde des routines GEN20 utilisées dans l'exemple :



f 1 0 512 20 2 - fenêtre de Hanning



f 2 0 512 20 6 1 - fenêtre gaussienne

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.2 de Csound

L'argument facultatif de la gaussienne a été ajouté dans la version 5.10

# GEN21

GEN21 — Génère les tables de différentes distributions aléatoires.

## Description

Génère les tables de différentes distributions aléatoires. (Voir aussi *betarand*, *bexprnd*, *cauchy*, *exprand*, *gauss*, *linrand*, *pcauchy*, *poisson*, *trirand*, *unirand* et *weibull*)

## Syntaxe

```
f # date taille 21 type niveau [arg1 [arg2]]
```

## Initialisation

*date* et *taille* sont les arguments habituels des fonctions GEN. *niveau* définit l'amplitude. Noter que GEN21 n'effectue pas d'auto-normalisation comme le font la plupart des autres fonctions GEN. *type* définit la distribution à utiliser :

- 1 = Uniforme (seulement des nombres positifs)
- 2 = Linéaire (seulement des nombres positifs)
- 3 = Triangulaire (nombres positifs et négatifs)
- 4 = Exponentielle (seulement des nombres positifs)
- 5 = Biexponentielle (nombres positifs et négatifs)
- 6 = Gaussienne (nombres positifs et négatifs)
- 7 = Cauchy (nombres positifs et négatifs)
- 8 = Cauchy Positive (seulement des nombres positifs)
- 9 = Beta (seulement des nombres positifs)
- 10 = Weibull (seulement des nombres positifs)
- 11 = Poisson (seulement des nombres positifs)

De tous ces cas seulement le 9 (Beta) et le 10 (Weibull) ont besoin d'arguments supplémentaires. Beta nécessite deux arguments et Weibull un.

Si *type* = 6, les nombres aléatoires dans la ftable suivent une distribution normale centrée sur 0 ( $\mu = 0.0$ ) avec une variance ( $\sigma$ ) de  $level / 3.83$ . Ainsi plus de 99.99% des valeurs aléatoires générées sont comprises entre  $-level$  et  $+level$ . La valeur par défaut de *level* est 1 ( $\sigma = 0.261$ ). Si l'on veut une valeur moyenne différente de 0.0, il faut ajouter cette valeur moyenne aux nombres générés.

## Exemples

```
f1 0 1024 21 1 ; Uniforme (bruit blanc)
f1 0 1024 21 6 ; Gaussienne (mu=0.0, sigma=1/3.83=0.261)
```

```
f1 0 1024 21 6 5.745 ; Gaussienne (mu=0.0, sigma=5.745/3.83=1.5)
f1 0 1024 21 9 1 1 2 ; Beta (noter que le niveau précède les arguments)
f1 0 1024 21 10 1 2 ; Weibull
```

Toutes les additions ci-dessus furent conçus par l'auteur entre mai et décembre 1994, sous la supervision du Dr Richard Boulanger.

Voici un exemple complet de la routine GEN21. Il utilise le fichier *gen21.csd* [exemples/gen21.csd].

### Exemple 1218. Exemple de la routine GEN21.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac ;;;realtime audio out
;-iadc ;;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen21.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifn = p4
isize = ftlen(ifn)
prints "TABLE NUMBER: %d\n", ifn
prints "Index\tValue\n"

iindex = 0 ;start loop
begin_loop:
  ivalue tab_i iindex, ifn
  prints "%d:\t%f\n", iindex, ivalue
  iindex = iindex + 1
  if (iindex < isize) igoto begin_loop

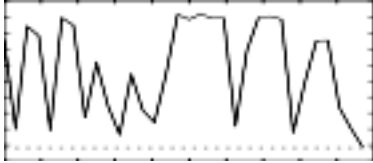
;produce sound - and repeat it 10 times so you can hear the patterns:
aphase phasor 10/10 ;play all 32 values 10x over 10 seconds
aphase = aphase*isize ;step through table
afrq table aphase, p4 ;read table number
asig poscil .5, (afrq*500)+1000,10 ;scale values of table 500 times, add 1000 Hz
outs asig , asig ;so we can distinguish the different tables
endin

</CsInstruments>
<CsScore>
f1 0 32 21 1 ;Uniform (white noise)
f2 0 32 21 6 ;Gaussian (mu=0.0, sigma=1/3.83=0.261)
f3 0 32 21 6 5.745 ;Gaussian (mu=0.0, sigma=5.745/3.83=1.5)
f4 0 32 21 9 1 1 2 ;Beta (note that level precedes arguments)
f5 0 32 21 10 1 2 ;Weibull
f10 0 8192 10 1 ;Sine wave

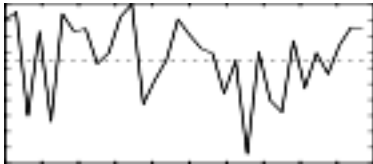
i 1 0 10 1
i 1 11 10 2
i 1 22 10 3
i 1 33 10 4
```

```
i 1 44 10 5  
e  
</CsScore>  
</CsoundSynthesizer>
```

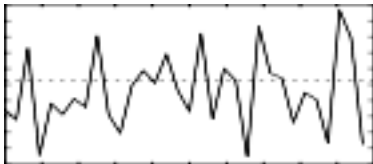
Voici les diagrammes des formes d'onde des routines GEN21 utilisées dans l'exemple :



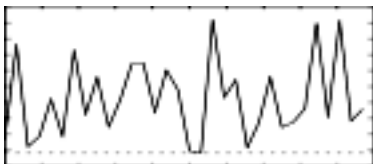
f 1 0 32 21 1 - seulement des nombres positifs



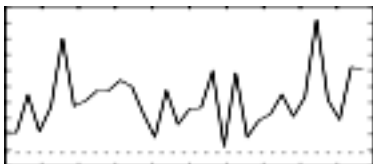
f 2 0 32 21 6



f 3 0 32 21 6 5.745



f 4 0 32 21 9 1 1 2 - seulement des nombres positifs



f 5 0 32 21 10 1 2 - seulement des nombres positifs

## Crédits

Auteur : Paris Smaragdis  
MIT, Cambridge  
1995

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK

Précisions sur mu et sigma ajoutées par François Pinot après une discussion avec Joachim Heintz sur la liste de Csound, Décembre 2010.

Nouveau dans la version 3.2 de Csound

# GEN23

GEN23 — Lit des valeurs numériques à partir d'un fichier texte.

## Description

Ce sous-programme lit des valeurs numériques à partir d'un fichier ASCII.

## Syntaxe

```
f # date taille -23 "nomfichier.txt"
```

## Initialisation

"*nomfichier.txt*" -- les valeurs numériques contenues dans "*nomfichier.txt*" (qui indique le nom de chemin complet du fichier de caractères à lire) peuvent être séparées par des espaces, des tabulations, des caractères de passage à la ligne ou des virgules.

*taille* -- nombre de points dans la table. Doit être une puissance de 2, une puissance de 2 + 1, ou zéro. Si *taille* = 0, la taille de la table est déterminée par le nombre de valeurs numériques dans *nomfichier.txt*. (Nouveau dans la version 3.57 de Csound)



### Note

Tous les caractères suivant un ';', un '#' (commentaire) ou un '<' (balise XML depuis la version 6.04) sont ignorés jusqu'à la ligne suivante (les nombres aussi).

## Exemples

Voici un exemple de la routine GEN23. Il utilise les fichiers *gen23.csd* [examples/gen23.csd] et *spectrum.txt* [examples/spectrum.txt].

### Exemple 1219. Exemple de la routine GEN23.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen23.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
;"spectrum.txt" is created by the spectrum plotter of Audacity (set at size 128), using "fox.wav".
```

```
instr 1 ;performs additive synthesis based on spectrum.txt

indx =0      ;start reading at first value
loop:
ifreq tab_i indx, 2      ;take odd values of list (= frequency)
iamp tab_i indx+1, 2      ;take even values of list (= amplitude)
event_i "i", 10, 0, p3, iamp, ifreq      ;use "event_i" to trigger instr. 10
    loop_lt indx, 2, 126, loop      ;use all 126 frequency and amplitude values

endin

instr 10 ;generate sound

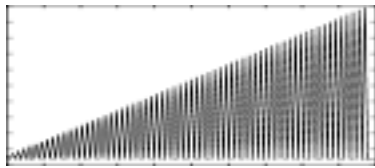
iamp = p4
ifreq = p5
asig poscil ampdb(iamp), ifreq, 1
asig linen asig, .01, p3, p2
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1      ;sine wave
f 2 0 128 -23 "spectrum.txt" ;"spectrum.txt" can be found in /manual/examples

i1 0 2

e
</CsScore>
</CsoundSynthesizer>
```

Voici le diagramme de la forme d'onde de la routine GEN23 utilisée dans l'exemple :



f 2 0 128 -23 "spectrum.txt" - non normalisé

## Crédits

Auteur : Gabriel Maldonado  
Italie  
Février 1998

Nouveau dans la version 3.47 de Csound. Les commentaires commençant par un '#' sont ignorés depuis la version 5.12 de csound.

# GEN24

GEN24 — Lit les valeurs numériques d'une table de fonction déjà allouée en les reproportionnant.

## Description

Ce sous-programme lit les valeurs numériques d'une table de fonction déjà allouée et les reproportionne selon les valeurs *min* et *max* données par l'utilisateur.

## Syntaxe

```
f # date taille -24 ftable min max
```

## Initialisation

*#*, *date*, *taille* -- les paramètres GEN habituels. Voir l'instruction *f*.

*ftable* -- *ftable* doit être une table déjà allouée avec la même taille que cette fonction.

*min*, *max* -- l'intervalle de recadrage.



### Note

Ce GEN est utile, par exemple, pour éliminer le décalage du début dans les morceaux d'exponentielle permettant d'avoir une vraie origine à zéro.

## Exemples

Voici un exemple de la routine GEN24. Il utilise le fichier *gen24.csd* [exemples/gen24.csd].

### Exemple 1220. Exemple de la routine GEN24.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen24.wav -W   ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ifn = p4      ;choose between tables
kcps init 1/p3 ;create index over duration of note.
```



```
kndx phasor kcps
ixmode = 1 ;normalize to 0-1
kval table kndx, ifn, ixmode
asig poscil .7, 440 + kval, 1 ;add to frequency
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave
f 10 0 16384 -24 1 0 400;scale sine wave from table 1 from 0 to 400
f 11 0 16384 -24 1 0 50 ;and from 0 to 50

i 1 0 3 10
i 1 4 3 11
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Gabriel Maldonado

Nouveau dans la version 4.16 de Csound

# GEN25

GEN25 — Construit des fonctions à partir de morceaux de courbes exponentielles avec des points charnière (breakpoints).

## Description

Ces sous-programmes sont utilisés pour construire des fonctions à partir de morceaux de courbes exponentielles avec des points charnière (breakpoints).

## Syntaxe

```
f # date taille 25 x1 y1 x2 y2 x3 ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*).

*x1*, *x2*, *x3*, etc. -- positions dans la table auxquelles la valeur *y* suivante devra être atteinte. Doivent être en ordre croissant. Si la dernière valeur est inférieure à la taille, les positions restantes seront mises à zéro. Ne doivent pas être négatives mais peuvent être nulles.

*y1*, *y2*, *y3*, etc. -- Valeurs charnière atteintes à la position spécifiée par la valeur *x* précédente. Elles doivent être non nulles et toutes du même signe.



### Note

Si *p4* est positif, les fonctions sont post-normalisées (reproportionnées à une valeur absolue maximale de 1 après génération). Un *p4* négatif empêchera cette opération.

## Exemples

Voici un exemple de la routine GEN25. Il utilise le fichier *gen25.csd* [examples/gen25.csd].

### Exemple 1221. Exemple de la routine GEN25.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen25.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
```

```
Odbfs = 1

gisin ftgen 1, 0, 32768, 10, 1
gienv ftgen 2, 0, 1025, 25, 0, 0.01, 200, 1, 400, 1, 513, 0.01 ; y value must be >= 0

instr 1

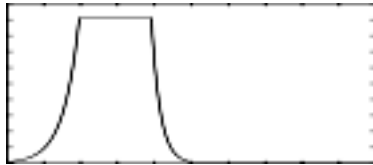
kcps init 3/p3 ;play 3x over duration of note
kndx phasor kcps
ixmode = 1 ;normalize to 0-1
kval table kndx, gienv, ixmode
kval =kval*100 ;scale up to 0-100
asig poscil 1, 220+kval, gisin ;use table for amplitude
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 4

e
</CsScore>
</CsoundSynthesizer>
```

Voici le diagramme de la forme d'onde de la routine GEN25 utilisée dans l'exemple :



f 2 0 1025 25 0 0.01 200 1 400 1 513 0.01 - une fonction qui commence à 0.01, monte jusqu'à 1 à la position 200 de la table, trace un segment de droite jusqu'à la position 400, et revient à 0.01 à la fin de la table

## Voir aussi

*Instruction f, GEN27*

## Crédits

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.49 de Csound

# GEN27

GEN27 — Construit des fonctions à partir de morceaux de lignes droites avec des points charnière.

## Description

Construit des fonctions à partir de morceaux de lignes droites avec des points charnière.

## Syntaxe

```
f # date taille 27 x1 y1 x2 y2 x3 ...
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1. (voir l'instruction *f*).

*x1*, *x2*, *x3*, etc. -- positions dans la table auxquelles la valeur *y* suivante devra être atteinte. Doivent être en ordre croissant. Si la dernière valeur est inférieure à la taille, les positions restantes seront mises à zéro. Ne doivent pas être négatives mais peuvent être nulles.

*y1*, *y2*, *y3*, etc. -- Valeurs charnière atteintes à la position spécifiée par la valeur *x* précédente.



### Note

Si *p4* est positif, les fonctions sont post-normalisées (reproportionnées à une valeur absolue maximale de 1 après génération). Un *p4* négatif empêchera cette opération.

## Exemples

Voici un exemple de la routine GEN27. Il utilise le fichier *gen27.csd* [examples/gen27.csd].

### Exemple 1222. Exemple de la routine GEN27.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen27.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

gisin ftgen 1, 0, 32768, 10, 1
gienv ftgen 2, 0, 1025, 27, 0, 0,200, 1, 400, -1, 513, 0
```

```
instr 1

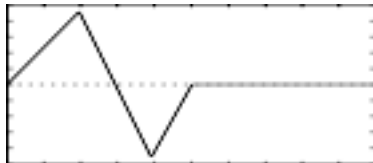
kcps init 3/p3 ;play 3x over duration of note
kndx phasor kcps
ixmode = 1 ;normalize to 0-1
kval table kndx, gienv, ixmode
kval = kval*100 ;scale 0-100
asig poscil 1, 220+kval, 1 ;add to 220 Hz
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 4

e
</CsScore>
</CsoundSynthesizer>
```

Voici le diagramme de la forme d'onde de la routine GEN27 utilisée dans l'exemple :



f 2 0 1025 27 0 0 200 1 400 -1 513 0 - une fonction qui commence à 0, monte jusqu'à 1 à la position 200 de la table, descend jusqu'à -1 à la position 400, et revient à 0 à la fin de la table. L'interpolation est linéaire

## Voir aussi

*Instruction f, GEN25*

## Crédits

Auteur : John ffitch  
University of Bath/Codemist Ltd.  
Bath, UK

Nouveau dans la version 3.49 de Csound

# GEN28

GEN28 — Lit un fichier texte qui contient une trajectoire paramétrée par le temps.

## Description

Ce générateur de fonction lit un fichier texte qui contient des ensembles de trois valeurs représentant des coordonnées xy et un paramètre temporel indiquant quand placer le signal à cette position, permettant à l'utilisateur de définir une trajectoire paramétrée par le temps. Le format du fichier est de la forme :

```
temps1  X1  Y1
temps2  X2  Y2
temps3  X3  Y3
```

La configuration des coordonnées xy dans l'espace place le signal de la manière suivante :

- a1 est -1, 1
- a2 est 1, 1
- a3 est -1, -1
- a4 est 1, -1

Cela suppose des haut-parleurs disposés avec a1 en avant gauche, a2 en avant droite, a3 en arrière gauche, a4 en arrière droite. Les valeurs supérieures à 1 provoqueront une atténuation des sons comme s'ils étaient distants. *GEN28* crée les valeurs avec une résolution de 10 millisecondes.

## Syntaxe

```
f # date taille 28 codfic
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être 0. *GEN28* prend une taille de 0 et alloue la mémoire automatiquement.

*codfic* -- chaîne de caractères dénotant le nom du fichier source. Une chaîne de caractères (entre apostrophes doubles, espaces autorisés) donne le nom du fichier lui-même, optionnellement un nom de chemin complet. Si le chemin n'est pas complet, le fichier nommé est cherché dans le répertoire courant.

## Exemples

```
f1 0 0 28 "move"
```

Le fichier "move" ressemblera à ceci :

```
0   -1   1
1    1   1
```

2	4	4
2.1	-4	-4
3	10	-10
5	-40	0

Puisque *GEN28* crée les valeurs avec une résolution de 10 millisecondes, il y aura 500 valeurs créées en interpolant entre X1 et X2, X2 et X3, etc., et entre Y1 et Y2, Y2 et Y3, etc., sur le nombre approprié de valeurs qui sont stockées dans la table de fonction. Le son démarrera à l'avant gauche, il bougera pendant 1 seconde vers l'avant droite, durant la seconde suivante il s'éloignera mais toujours à l'avant droite, ensuite il bougera vers l'arrière gauche en seulement 1/10 de seconde, un peu éloigné. Enfin, pendant les 0,9 secondes restantes le son bougera vers l'arrière droite, modérément éloigné, et il viendra s'arrêter entre les deux canaux gauche (plein ouest !), assez éloigné.

Voici un exemple de la routine GEN28. Il utilise le fichier *gen28.csd* [examples/gen28.csd].

### Exemple 1223. Exemple de la routine GEN28.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen28.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 4

ga1 init 0
ga2 init 0
ga3 init 0
ga4 init 0

instr 1 ;uses GEN28 file "move", as found in /manual/examples

kx    init 0
ky    init 0
ktime line 0, 5, 5    ;same time as in table 1 ("move")
asig   diskin2 "beats.wav", 1, 0, 1 ;sound source is looped
a1, a2, a3, a4 space asig, 1, ktime, .1, kx, ky ;use table 1 = GEN28
ar1, ar2, ar3, ar4 spsend    ;send to reverb

ga1 = ga1+ar1
ga2 = ga2+ar2
ga3 = ga3+ar3
ga4 = ga4+ar4
outq a1, a2, a3, a4

endin

instr 99 ; reverb instrument

a1 reverb2 ga1, 2.5, .5
a2 reverb2 ga2, 2.5, .5
```

```
a3 reverb2 ga3, 2.5, .5
a4 reverb2 ga4, 2.5, .5
  outq a1, a2, a3, a4

ga1=0
ga2=0
ga3=0
ga4=0

endin
</CsInstruments>
<CsScore>
f1 0 0 28 "move"

i1 0 5 ;same time as ktime
i 99 0 10 ;keep reverb active
e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Richard Karpen  
Seattle, Wash  
1998

Nouveau dans la version 3.48 de Csound



# GEN30

GEN30 — Génère des partiels harmoniques en analysant une table existante.

## Description

Extrait un sous-ensemble de la série harmonique d'une forme d'onde existante.

## Syntaxe

```
f # date taille 30 src minh maxh [ref_sr] [interp]
```

## Exécution

*src* -- ftable source

*minh* -- numéro de l'harmonique le plus bas

*maxh* -- numéro de l'harmonique le plus haut

*ref\_sr* (facultatif) -- *maxh* est pondéré par (*sr* / *ref\_sr*). La valeur par défaut de *ref\_sr* est *sr*. Si *ref\_sr* est nul ou négatif, il est ignoré.

*interp* (facultatif) -- si différent de zéro, permet de changer l'amplitude des harmoniques le plus bas et le plus haut en fonction de la partie fractionnaire de *minh* et *maxh*. Par exemple, si *maxh* vaut 11.3 alors le 12ème harmonique est ajouté avec une amplitude de 0.3. Ce paramètre vaut zéro par défaut.

GEN30 ne supporte pas les tables avec un point de garde (c'est-à-dire une taille de table = puissance-de-deux + 1). Bien que de telles tables fonctionnent aussi bien en entrée qu'en sortie, lors de la lecture d'une table source, le point de garde est ignoré, et lors de l'écriture de la table en sortie, le point de garde est simplement copié du premier échantillon (index de table = 0).

La raison de cette limitation est que GEN30 utilise la TFR, qui nécessite que la taille de table soit une puissance de deux. GEN32 permet l'utilisation de l'interpolation linéaire pour le rééchantillonnage et le déphasage, ce qui rend possible l'utilisation de n'importe quelle taille de table (cependant, pour les partiels calculés par TFR, la limitation de la puissance de deux existe toujours).

## Exemples

Voici un exemple de la routine GEN30. Il utilise le fichier *gen30.csd* [examples/gen30.csd].

### Exemple 1224. Exemple de la routine GEN30.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      ;;realtime audio out
; For Non-realtime ouput leave only the line below:
; -o gen30.wav -W ;; for file output any platform
```

```

</CsOptions>
<CsInstruments>

; a simplified example of Istvan Varga
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

isaw ftgen 1, 0, 16384, 7, 1, 16384, -1 ;sawtooth wave
iFM ftgen 3, 0, 4096, 7, 0, 512, 0.25, 512, 1, 512, 0.25, 512, \
    0, 512, -0.25, 512, -1, 512, -0.25, 512, 0 ;FM waveform
iAM ftgen 4, 0, 4096, 5, 1, 4096, 0.01 ;AM waveform
iEQ ftgen 5, 0, 1024, 5, 1, 512, 32, 512, 1 ;FM to EQ
isine ftgen 6, 0, 1024, 10, 1 ;sine wave

/* generate bandlimited sawtooth waves */
i0 = 0
loop1:
imaxh = sr / (2 * 440.0 * exp(log(2.0) * (i0 - 69) / 12))
i_ ftgen i0 + 10, 0, 4096, -30, 1, 1, imaxh ;use gen 30
i0 = i0 + 1
if (i0 < 127.5) igoto loop1

instr 1

kcps = 440.0 * exp(log(2.0) * (p4 - 69) / 12) ;note frequency
klpmaxf limit p5 * kcps, 1000.0, 12000.0 ;lowpass max. frequency

kfmd1 = 0.03 * kcps ;FM depth in Hz
kamfr = kcps * 0.02 ;AM frequency
kamfr2 = kcps * 0.1

kfnum = (10 + 69 + 0.5 + 12 * log(kcps / 440.0) / log(2.0)) ;table number
aenv linseg 0, p3*0.25, 1, p3*0.75, 0 ;amp. envelope

asig oscbnk kcps, 0.0, kfmd1, 0.0, 40, 200, 0.1, 0.2, 0, 0, 144, \
    0.0, klpmaxf, 0.0, 0.0, 1.5, 1.5, 2, kfnum, 3, 0, 5, 5, 5
asig = asig * aenv*.03
outs asig, asig

endin

</CsInstruments>
<CsScore>
s
i 1 0 6 41 10
i 1 0 6 60
i 1 0 6 65
i 1 0 6 69
s
i 1 0 6 41 64
i 1 0 6 60
i 1 0 6 65
i 1 0 6 69
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.16

# GEN31

GEN31 — Mélange n'importe quelle forme d'onde définie dans une table existante.

## Description

Cette routine est semblable à GEN09, mais permet le mélange de n'importe quelle forme d'onde définie dans une table existante.

## Syntaxe

```
f # date taille 31 src pna ampa phsa pnb ampb phsb ...
```

## Exécution

*src* -- numéro de la table source

*pna, pnb, ...* -- numéro de partiel, doit être un entier positif

*ampa, ampb, ...* -- échelle d'amplitude

*phsa, phsb, ...* -- phase initiale (0 à 1)

*GEN31* ne supporte pas les tables avec un point de garde (c'est-à-dire une taille de table = puissance-de-deux + 1). Bien que de telles tables fonctionnent aussi bien en entrée qu'en sortie, lors de la lecture d'une table source, le point de garde est ignoré, et lors de l'écriture de la table en sortie, le point de garde est simplement copié du premier échantillon (index de table = 0).

La raison de cette limitation est que *GEN31* utilise la TFR, qui nécessite que la taille de table soit une puissance de deux. *GEN32* permet l'utilisation de l'interpolation linéaire pour le rééchantillonnage et le déphasage, ce qui rend possible l'utilisation de n'importe quelle taille de table (cependant, pour les partiels calculés par TFR, la limitation de la puissance de deux existe toujours).

## Exemples

Voici un exemple de la routine GEN31. Il utilise le fichier *gen31.csd* [examples/gen31.csd].

### Exemple 1225. Exemple de la routine GEN31.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>

<CsInstruments>
0dbfs = 1

gisine ftgen 0,0,4096,10,1
gi31   ftgen 0,0,4096,31,gisine, 1,1,0, 2,1,0, 3,1,0, 4,1,0, 5,1,0, 6,1,0

instr 1
  aa oscil3 0.6, 440, gi31
  out aa
```

```
endin
</CsInstruments>

<CsScore>
i1 0 5
e
</CsScore>

</CsoundSynthesizer>
```

## Crédits

Auteur : Istvan Varga

Nouveau dans la version 4.15

# GEN32

GEN32 — Mélange n'importe quelle forme d'onde, rééchantillonnée soit par TFR soit par interpolation linéaire.

## Description

Cette routine est semblable à *GEN31*, mais elle permet la spécification d'une table source pour chaque partiel. Les tables peuvent être rééchantillonnées soit par TFR soit par interpolation linéaire.

## Syntaxe

```
f # date taille 32 srca pna ampa phsa srcb pnb ampb phsb ...
```

## Exécution

*srca, srcb* -- numéro de table source. Une valeur négative peut être utilisée pour lire une table avec interpolation linéaire (par défaut, la forme d'onde source est transposée et déphasée par TFR) ; c'est moins précis, mais plus rapide, et cela permet des numéros de partiels non entiers et négatifs.

*pnna, pnb*, ... -- numéro de partiel, doit être un entier positif si le numéro de la table source est positif (c'est-à-dire rééchantillonnage par TFR).

*ampa, ampb*, ... -- échelle d'amplitude

*phsa, phsb*, ... -- phase initiale (0 à 1)

## Exemples

Voici un exemple de la routine GEN32. Il utilise le fichier *gen32.csd* [examples/gen32.csd].

### Exemple 1226. Exemple de la routine GEN32.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen32.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

itmp      ftgen 1, 0, 16384, 7, 1, 16384, -1 ; sawtooth
itmp      ftgen 2, 0, 8192, 10, 1 ; sine
itmp      ftgen 5, 0, 4096, -32, -2, 1.5, 1.0, 0.25, 1, 2, 0.5, 0, 1, 3, -0.25, 0.5 ; mix tables
```

```

itmp    ftgen 6, 0, 16384, 20, 3, 1    ; window
; generate band-limited waveforms
inote   = 0
loop0:
icps    = 440 * exp(log(2) * (inote - 69) / 12)    ; one table for
inumh   = sr / (2 * icps)                        ; each MIDI note number
ift     = int(inote + 256.5)
itmp    ftgen ift, 0, 4096, -30, 5, 1, inumh
inote   = inote + 1
        if (inote < 127.5) igoto loop0

instr 1

kcps    expon 20, p3, 16000
kft     = int(256.5 + 69 + 12 * log(kcps / 440) / log(2))
kft     = (kft > 383 ? 383 : kft)
a1      phasor kcps
a1      tableikt a1, kft, 1, 0, 1
outs    a1*.5, a1*.5

endin

instr 2

kcps    expon 20, p3, 16000
kft     = int(256.5 + 69 + 12 * log(kcps / 440) / log(2))
kft     = (kft > 383 ? 383 : kft)
kgdur   limit 10 / kcps, 0.1, 1
a1      grain2 kcps, 0.02, kgdur, 30, kft, 6, -0.5
outs    a1*.08, a1*.08

endin
</CsInstruments>
<CsScore>
t 0 60

i 1 0 10
i 2 12 10
e
</CsScore>
</CsoundSynthesizer>

```

## Crédits

Auteur : Rasmus Ekman

Programmeur : Istvan Varga

Nouveau dans la version 4.17

# GEN33

GEN33 — Génère des formes d'onde complexes en mélangeant des sinus.

## Description

Ces routines génèrent des formes d'onde complexes en mélangeant des sinus, comme *GEN09*, mais les paramètres des partiels sont spécifiés dans une table déjà existante, ce qui permet de calculer n'importe quel nombre de partiels dans l'orchestre.

La différence entre *GEN33* et *GEN34* est que *GEN33* utilise la TFR inverse pour générer la sortie, alors que *GEN34* est basé sur l'algorithme utilisé dans les opcode oscils. *GEN33* ne permet que des partiels entiers, et ne supporte pas les tailles de table égales à une puissance-de-deux plus 1, mais peut être significativement plus rapide avec un grand nombre de partiels. D'un autre côté, avec *GEN34*, il est possible d'utiliser des numéros de partiel non entiers et un point de garde, et cette routine peut être plus rapide s'il n'y a qu'un petit nombre de partiels (noter que *GEN34* est aussi plusieurs fois plus rapide que *GEN09*, bien que ce dernier soit plus précis).

## Syntaxe

```
f # date taille 33 src nh ech [fmode]
```

## Initialisation

*taille* -- nombre de points dans la table. Doit être une puissance de deux et au moins 4.

*src* -- numéro de la table source. Cette table contient les paramètres de chaque partiel dans le format suivant :

*ampa*, *pna*, *phsa*, *ampb*, *pnb*, *phsb*, ...

les paramètres sont :

- *ampa*, *ampb*, etc. : amplitude relative des partiels. L'amplitude actuelle dépend de la valeur de *ech*, ou de la normalisation (si celle-ci est active).
- *pna*, *pnb*, etc. : numéro de partiel, ou fréquence, en fonction de *fmode* (voir ci-dessous) ; zéro et des valeurs négatives sont autorisés, cependant, si la valeur absolue du numéro de partiel dépasse (*taille* / 2), le partiel ne sera pas rendu. Avec *GEN33*, le numéro de partiel est arrondi à l'entier le plus proche.
- *phsa*, *phsb*, etc. : phase initiale, dans l'intervalle de 0 à 1.

La longueur de la table (sans compter le point de garde) devrait être d'au moins  $3 * nh$ . Si la table est trop courte, le nombre de partiels (*nh*) est réduit à (longueur de la table) / 3, arrondi vers zéro.

*nh* -- nombre de partiels. Zéro ou des valeurs négatives sont autorisés, et donnent une table vide (silence). Le nombre effectif peut être diminué si la table source (*src*) est trop courte, ou si certains partiels ont une fréquence trop haute.

*ech* -- échelle d'amplitude.

*fmode* (facultatif, défaut = 0) -- une valeur non nulle indique que les fréquences sont en Hz au lieu de numéros de partiel dans la table source. Le taux d'échantillonnage est supposé être *fmode* si celui-ci est positif, ou  $-(sr * fmode)$  si une valeur négative est spécifiée.

## Exemples

Voici un exemple de la routine GEN33. Il utilise le fichier *gen33.csd* [examples/gen33.csd].

### Exemple 1227. Exemple de la routine GEN33.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen33.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; partials 1, 4, 7, 10, 13, 16, etc. with base frequency of 400 Hz

ibsfrq = 400
inumh = int(1.5 + sr * 0.5 / (3 * ibsfrq)) ; estimate number of partials
isrcln = int(0.5 + exp(log(2) * int(1.01 + log(inumh * 3) / log(2)))) ; source table length
itmp ftgen 1, 0, isrcln, -2, 0 ; create empty source table
ifpos = 0
ifrq = ibsfrq
inumh = 0
11:
    tableiw ibsfrq / ifrq, ifpos, 1 ; amplitude
    tableiw ifrq, ifpos + 1, 1 ; frequency
    tableiw 0, ifpos + 2, 1 ; phase
ifpos = ifpos + 3
ifrq = ifrq + ibsfrq * 3
inumh = inumh + 1
if (ifrq < (sr * 0.5)) igoto 11

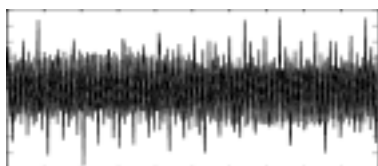
itemp ftgen 2, 0, 262144, -33, 1, inumh, 1, -1 ; store output in ftable 2 (size = 262144)
asig poscil .5, ibsfrq, itemp
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 2

e
</CsScore>
</CsoundSynthesizer>
```

Voici le diagramme de la forme d'onde générée par GEN33, utilisé dans l'exemple :





f 2 0 262144 -33 1 inumh 1 -1

## Voir aussi

*GEN09, GEN34*

## Crédits

Programmeur : Istvan Varga  
Mars 2002

Nouveau dans la version 4.19

# GEN34

GEN34 — Génère des formes d'onde complexes en mélangeant des sinus.

## Description

Ces routines génèrent des formes d'onde complexes en mélangeant des sinus, comme *GEN09*, mais les paramètres des partiels sont spécifiés dans une table déjà existante, ce qui permet de calculer n'importe quel nombre de partiels dans l'orchestre.

La différence entre *GEN33* et *GEN34* est que *GEN33* utilise la TFR inverse pour générer la sortie, alors que *GEN34* est basé sur l'algorithme utilisé dans les opcode oscils. *GEN33* ne permet que des partiels entiers, et ne supporte pas les tailles de table égales à une puissance-de-deux plus 1, mais peut être significativement plus rapide avec un grand nombre de partiels. D'un autre côté, avec *GEN34*, il est possible d'utiliser des numéros de partiel non entiers et un point de garde, et cette routine peut être plus rapide s'il n'y a qu'un petit nombre de partiels (noter que *GEN34* est aussi plusieurs fois plus rapide que *GEN09*, bien que ce dernier soit plus précis).

## Syntaxe

```
f # date taille 34 src nh ech [fmode]
```

## Initialisation

*size* -- nombre de points dans la table. Doit être une puissance de deux ou une puissance-de-deux plus 1.

*src* -- numéro de la table source. Cette table contient les paramètres de chaque partiel dans le format suivant :

*ampa*, *pna*, *phsa*, *ampb*, *pnb*, *phsb*, ...

les paramètres sont :

- *ampa*, *ampb*, etc. : amplitude relative des partiels. L'amplitude actuelle dépend de la valeur de *ech*, ou de la normalisation (si celle-ci est active).
- *pna*, *pnb*, etc. : numéro de partiel, ou fréquence, en fonction de *fmode* (voir ci-dessous) ; zéro et des valeurs négatives sont autorisés, cependant, si la valeur absolue du numéro de partiel dépasse (*taille* / 2), le partiel ne sera pas rendu.
- *phsa*, *phsb*, etc. : phase initiale, dans l'intervalle de 0 à 1.

La longueur de la table (sans compter le point de garde) devrait être d'au moins  $3 * nh$ . Si la table est trop courte, le nombre de partiels (*nh*) est réduit à (longueur de la table) / 3, arrondi vers zéro.

*nh* -- nombre de partiels. Zéro ou des valeurs négatives sont autorisés, et donnent une table vide (silence). Le nombre effectif peut être diminué si la table source (*src*) est trop courte, ou si certains partiels ont une fréquence trop haute.

*ech* -- échelle d'amplitude.

*fmode* (facultatif, défaut = 0) -- une valeur non nulle indique que les fréquences sont en Hz au lieu de numéros de partiel dans la table source. Le taux d'échantillonnage est supposé être *fmode* si celui-ci est positif, ou  $-(sr * fmode)$  si une valeur négative est spécifiée.

## Exemples

Voici un exemple de la routine GEN34. Il utilise le fichier *gen34.csd* [examples/gen34.csd].

### Exemple 1228. Exemple de la routine GEN34.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen34.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; partials 1, 4, 7, 10, 13, 16, etc. with base frequency of 400 Hz

ibsfrq = 400
inumh = int(1.5 + sr * 0.5 / (3 * ibsfrq)) ; estimate number of partials
isrcln = int(0.5 + exp(log(2) * int(1.01 + log(inumh * 3) / log(2)))) ; source table length
itmp ftgen 1, 0, isrcln, -2, 0 ; create empty source table
ifpos = 0
ifrq = ibsfrq
inumh = 0
11:
    tableiw ibsfrq / ifrq, ifpos, 1 ; amplitude
    tableiw ifrq, ifpos + 1, 1 ; frequency
    tableiw 0, ifpos + 2, 1 ; phase
ifpos = ifpos + 3
ifrq = ifrq + ibsfrq * 3
inumh = inumh + 1
if (ifrq < (sr * 0.5)) igoto 11

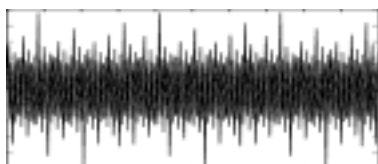
itmp ftgen 2, 0, 262144, -34, 1, inumh, 1, -1 ; store output in ftable 2 (size = 262144)
asig poscil .5, ibsfrq, itmp
outs asig, asig

endin
</CsInstruments>
<CsScore>

i 1 0 2

e
</CsScore>
</CsoundSynthesizer>
```

Voici le diagramme de la forme d'onde générée par GEN34, utilisé dans l'exemple :



f 2 0 262144 -34 1 inumh 1 -1

## Voir aussi

*GEN09, GEN33*

## Crédits

Programmeur : Istvan Varga  
Mars 2002

Nouveau dans la version 4.19

# GEN40

GEN40 — Génère une distribution aléatoire à partir d'un histogramme.

## Description

Génère une distribution aléatoire continue en partant de la forme d'un histogramme défini par l'utilisateur.

## Syntaxe

```
f # date taille 40 tblforme
```

## Exécution

La forme de l'histogramme doit être stockée dans une table préalablement définie, en fait, *tblforme* doit contenir le numéro de cette table.

La forme de l'histogramme peut être générée avec n'importe quelle GEN routine. Comme il n'y a pas d'interpolation lorsque GEN40 opère la traduction, il est suggéré de donner à la table contenant la forme de l'histogramme une taille raisonnablement grande, afin d'obtenir une meilleure précision (cependant, cette dernière table peut être détruite après le traitement pour récupérer de la mémoire).

Ce sous-programme est prévu pour être utilisé avec l'opcode *cuserrnd* (voir *cuserrnd* pour plus d'information).

## Exemples

Voici un exemple de l'opcode GEN40. Il utilise le fichier *gen40.csd* [examples/gen40.csd].

### Exemple 1229. Exemple de l'opcode GEN40.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o GEN40.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; every run time same values

kuser cuserrnd 0, 100, 1
printk .2, kuser
asig poscil .5, 220+kuser, 3
```

```

        outs asig, asig
    endin

    instr 2 ; every run time different values

        seed 0
    kuser cuserrrnd 0, 100, 1
        printk .2, kuser
    asig poscil .5, 220+kuser, 3
        outs asig, asig
    endin
</CsInstruments>
<CsScore>
f 1 0 16 -7 1 4 0 8 0 4 1 ;distrubution using GEN07
f 2 0 16384 40 1 ;GEN40 is to be used with cuserrrnd
f 3 0 8192 10 1 ;sine

i 1 0 2
i 2 3 2
e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

i 1 1 time 0.00067: 53.14918
i 1 1 time 0.20067: 0.00000
i 1 1 time 0.40067: 0.00000
i 1 1 time 0.60067: 96.80406
i 1 1 time 0.80067: 94.20729
i 1 1 time 1.00000: 0.00000
i 1 1 time 1.20067: 86.13032
i 1 1 time 1.40067: 31.37096
i 1 1 time 1.60067: 70.35889
i 1 1 time 1.80000: 0.00000
i 1 1 time 2.00000: 49.18914

```

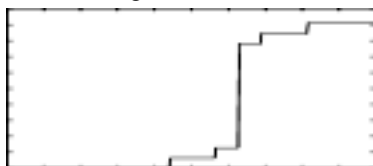
WARNING: Seeding from current time 2006647442

```

i 2 2 time 3.00067: 21.45002
i 2 2 time 3.20067: 44.32333
i 2 2 time 3.40067: 46.05420
i 2 2 time 3.60000: 0.00000
i 2 2 time 3.80067: 41.32175
i 2 2 time 4.00000: 0.00000
i 2 2 time 4.20000: 63.72019
i 2 2 time 4.40067: 0.00000
i 2 2 time 4.60067: 0.00000
i 2 2 time 4.80067: 0.00000
i 2 2 time 5.00000: 74.49330

```

Voici le diagramme de la forme d'onde de la routine GEN40 utilisée dans l'exemple :



f 2 0 16384 40 1

## Crédits

Auteur : Gabriel Maldonado

# GEN41

GEN41 — Génère une liste aléatoire de paires numériques.

## Description

Génère une fonction de distribution aléatoire discrète en donnant une liste de paires numériques.

## Syntaxe

```
f # date taille -41 valeur1 prob1 valeur2 prob2 valeur3 prob3 ... valeurN probN
```

## Exécution

Le premier nombre de chaque paire est une valeur, et le second est la probabilité que cette valeur soit choisie par un algorithme aléatoire. Même si n'importe quel nombre peut être assigné à l'élément probabilité de chaque paire, il vaut mieux lui donner une valeur en pourcentage, afin de rendre les choses plus claires pour l'utilisateur.

Ce sous-programme est prévu pour être utilisé avec les opcodes *dusernd* et *urd* (voir *dusernd* pour plus d'information).

## Exemples

Voici un exemple de l'opcode GEN41. Il utilise le fichier *gen41.csd* [examples/gen41.csd].

### Exemple 1230. Exemple de l'opcode GEN41.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc if RT audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o GEN41.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

k1    dusernd 1
      printk 0, k1
asig  poscil .5, 220*k1, 2
      outs asig, asig

endin
</CsInstruments>
```

```
<CsScore>
f1 0 -20 -41 2 .1 8 .9 ;choose 2 at 10% probability, 8 at 90%

f2 0 8192 10 1

i1 0 2
e
</CsScore>
</CsoundSynthesizer>
```

Sa sortie contiendra des lignes comme celles-ci :

```
i 1 time 0.00067: 8.00000
i 1 time 0.00133: 8.00000
i 1 time 0.00200: 8.00000
i 1 time 0.00267: 8.00000
i 1 time 0.00333: 2.00000
i 1 time 0.00400: 8.00000
i 1 time 0.00533: 8.00000
i 1 time 0.00600: 8.00000
.....
```

Voici le diagramme de la forme d'onde de la routine GEN41 utilisée dans l'exemple :



f 1 0 -20 -41 2 .1 8 .9

## Crédits

Auteur : Gabriel Maldonado



# GEN42

GEN42 — Génère une distribution aléatoire d'intervalles discrets de valeurs.

## Description

Génère une fonction de distribution aléatoire d'intervalles discrets de valeurs en donnant une liste de groupes de trois nombres.

## Syntaxe

```
f # date taille -42 min1 max1 prob1 min2 max2 prob2 min3 max3 prob3 ... minN maxN probN
```

## Exécution

Le premier nombre de chaque groupe est la valeur minimum de l'intervalle, le second est la valeur maximum et le troisième est la probabilité qu'un élément appartenant à cet intervalle de valeurs soit choisi par un algorithme aléatoire. La probabilité pour un intervalle doit être une fraction de 1, et la somme des probabilités pour tous les intervalles doit être égale à 1.

Ce sous-programme est prévu pour être utilisé avec les opcodes *dusernd* et *urd* (voir *dusernd* pour plus d'information). Comme ni *dusernd* ni *urd* n'utilisent l'interpolation, il est suggéré de donner une taille raisonnablement grande.

## Exemples

Voici un exemple de l'opcode GEN42. Il utilise le fichier *gen42.csd* [examples/gen42.csd].

### Exemple 1231. Exemple de l'opcode GEN42.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o GEN42.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

ktab = 1    ;ftable 1
kurd = urd(ktab)
ktrig metro 5    ;triggers 5 times per second
kres samphold kurd, ktrig ;sample and hold value of kurd
```

```

        printk2 kres ;print it
asig poscil .5, 220+kres, 2
    outs asig, asig
endin

instr 2

seed 0 ;every run new values

ktab = 1 ;ftable 1
kurd = urd(ktab)
ktrig metro 5 ;triggers 5 times per second
kres samphold kurd, ktrig ;sample and hold value of kurd
    printk2 kres ;print it
asig poscil .5, 220+kres, 2
    outs asig, asig

endin
</CsInstruments>
<CsScore>
f1 0 -20 -42 10 20 .3 100 200 .7 ;30% choose between 10 and 20 and 70% between 100 and 200
f2 0 8192 10 1 ;sine wave

i 1 0 5
i 2 6 5
e
</CsScore>
</CsoundSynthesizer>

```

Sa sortie contiendra des lignes comme celles-ci :

```

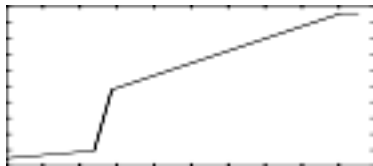
i1 184.61538
i1 130.76923
i1 169.23077
i1 12.00000
.....

WARNING: Seeding from current time 3751086165

i2 138.46154
i2 12.00000
i2 123.07692
i2 161.53846
i2 123.07692
i2 153.84615
.....

```

Voici le diagramme de la forme d'onde de la routine GEN42 utilisée dans l'exemple :



f 1 0 -20 -42 10 20 .3 100 200 .7

## Crédits

Auteur : Gabriel Maldonado

# GEN43

GEN43 — Charge un fichier PVOCEX contenant une analyse VP.

## Description

Ce sous-programme charge un fichier PVOCEX contenant l'analyse VP (amp-fréq) d'un fichier son et calcule les magnitudes moyennes de toutes les trames d'analyse d'un ou de tous les canaux audio. Il crée ensuite une table avec ces magnitudes pour chaque bin VP.

## Syntaxe

```
f # date taille 43 codfic canal
```

## Initialisation

*taille* -- nombre de points dans la table, puissance de deux ou puissance-de-deux plus 1. *GEN43* ne fait aucune distinction entre ces deux tailles, mais la table doit avoir pour taille au moins la moitié de celle de la tfr. Les bins VP couvrent le spectre positif de 0 Hz (index 0 de la table) à la fréquence de Nyquist (index  $taille/2+1$  de la table) par incréments réguliers (de taille  $sr/taille/2$ ).

*codfic* -- un fichier pvocex (qui peut être généré par pvanal).

*canal* -- numéro du canal audio duquel les magnitudes seront extraites ; un 0 donnera la moyenne des magnitudes de tous les canaux.

La lecture s'arrête à la fin du fichier.



### Note

Si p4 est positif, la table sera post-normalisée. Un p4 négatif empêchera la post-normalisation.

## Exemples

```
f1 0 512 43 "viola.pvx" 1
f1 0 -1024 -43 "noiseprint.pvx" 0
```

On peut utiliser cette table comme table de masquage pour *pvstencil* et *pvmask*. Le premier exemple utilise un fichier d'analyse de vocodeur de phase par TFR à 1024 points duquel on utilise le premier canal. Le second utilise tous les canaux d'un fichier de 2048 points, sans post-normalisation. Pour les applications à la réduction de bruit avec *pvstencil*, il est mieux de ne pas normaliser la table (code GEN négatif).

## Crédits

Auteur : Victor Lazzarini

# GEN49

GEN49 — Transfère les données d'un fichier son MP3 dans une table de fonction.

## Description

Ce sous-programme transfère les données d'un fichier son MP3 dans une table de fonction.

## Syntaxe

```
f# time size 49 filcod skiptime format
```

## Exécution

*size* -- nombre de points dans la table. Ordinairement une puissance de 2 ou une puissance-de-2 plus 1 (voir l'*instruction f*) ; la taille de table maximale est de 16777216 ( $2^{24}$ ) points. L'allocation de mémoire pour la table peut être *différée* en mettant ce paramètre à 0 ; la taille allouée est alors le nombre de points dans le fichier (probablement pas une puissance de 2), et la table n'est pas utilisable par les oscillateurs normaux, mais par l'unité *loscil*. Le fichier son peut être mono ou stéréo.

*filcod* -- entier ou chaîne de caractères dénotant le nom du fichier son source. Un entier dénote le fichier *soundin.filcod* ; une chaîne de caractères (entre apostrophes doubles, espaces autorisés) donne le nom du fichier lui-même, optionnellement un nom de chemin complet. Si le chemin n'est pas complet, le fichier est d'abord cherché dans le répertoire courant, ensuite dans celui qui est donné par la variable d'environnement *SSDIR* (si elle est définie) enfin par *SFDIR*. Voir aussi *soundin*.

*skiptime* -- commence à lire à *skiptime* secondes dans le fichier.

*format* -- spécifie le format de fichier audio requis :

1 - Fichier mono	3 - Premier canal (gauche)
2 - Fichier stéréo	4 - Second canal (droite)

Si *format* = 0 le format d'échantillon est pris dans l'en-tête du fichier son.



### Note

- La lecture s'arrête à la fin du fichier ou lorsque la table est pleine. Les cellules de la table non remplies contiendront des zéros.
- Si *p4* est positif, la table sera post-normalisée (reproportionnée avec une valeur absolue maximale de 1 après génération). Une valeur de *p4* négative empêche cette opération.

## Exemples

Voici un exemple de la routine GEN49. Il utilise le fichier *gen49.csd* [exemples/gen49.csd].

**Exemple 1232. Un exemple de la routine GEN49.**

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen49.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

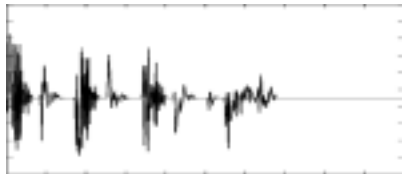
instr 1

kcps = sr/ftlen(1)
asig oscil .8, kcps, 1
outs asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 131072 49 "beats.mp3" 0 1 ;read an audio file (using GEN49).

i 1 0 2
e
</CsScore>
</CsoundSynthesizer>
```

Voici le diagramme de la forme d'onde de la routine GEN49 utilisée dans l'exemple :



f 1 0 131072 49 "beats.mp3" 0 1

## Crédits

Ecrit par John ffitch

Février 2009.

# GEN51

GEN51 — Ce sous-programme remplit une table avec une échelle microtonale personnalisée, à la manière des opcodes de Csound *cpstun*, *cpstuni* et *cpstmid*.

## Description

Ce sous-programme remplit une table avec une échelle microtonale personnalisée, à la manière des opcodes de Csound *cpstun*, *cpstuni* et *cpstmid*.

## Syntaxe

**f** # date taille -51 nbrdegres intervalle freqbase touchebase rapport1 rapport2 .... rapportN

## Exécution

Les quatre premiers paramètres (c'est-à-dire p5, p6, p7 et p8) définissent les directives de génération suivantes :

*p5 (nbrdegres)* -- le nombre de degrés de l'échelle microtonale

*p6 (intervalle)* -- l'intervalle de fréquences couvert avant de répéter les rapports des degrés, par exemple 2 pour une octave, 1,5 pour une quinte, etc.

*p7 (freqbase)* -- la fréquence de base de l'échelle en cps

*p8 (touchebase)* -- L'indice entier dans la table auquel assigner la fréquence de base inchangée

Les autres paramètres définissent les rapports de l'échelle :

*p9 ... pN (rapport1 ... etc.)* -- les rapports des degrés de l'échelle

Par exemple, pour une échelle standard de 12 degrés avec une fréquence de base de 261 cps assignée à la touche numéro 60, l'instruction f de la partition pour générer la table serait :

```
;          nbrdegres      freqbase      rapports (tempérament égal) .....
;          intervalle      touchebase
f1 0 64 -51      12      2      261      60      1 1.059463 1.12246 1.18920 ..etc...
```

Après le calcul du gen, la table f1 est remplie avec 64 valeurs de fréquences différentes. Le 60ème élément est rempli avec la valeur de fréquence 261, et tous les autres éléments de la table (précédents et suivants) sont remplis selon les rapports des degrés.

Un autre exemple avec une échelle de 24 degrés, une fréquence de base de 440 cps assignée à la touche numéro 48, et un intervalle de répétition de 1,5 :

```
;          nbrdegres      freqbase      rapports .....
;          intervalle      touchebase
f1 0 64 -51      24      1.5      440      48      1 1.01 1.02 1.03 ..etc...
```

## Exemples

Voici un exemple de la routine GEN51. Il utilise le fichier *gen51.csd* [exemples/gen51.csd].

**Exemple 1233. Un exemple de la routine GEN51.**

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac -M0 ;;realtime audio out and midi input
;-iadc ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gen51.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
;example by Iain McCurdy
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

giEqTmp12 ftgen 1,0,128,-51,12,2,cpsoct(8),60,1,2^(1/12),2^(2/12),2^(3/12),2^(4/12),2^(5/12),2^(6/12),2^(7/12),2^(8/12),2^(9/12),2^(10/12),2^(11/12)
giEqTmp10 ftgen 2,0,128,-51,10,2,cpsoct(8),60,1,2^(1/10),2^(2/10),2^(3/10),2^(4/10),2^(5/10),2^(6/10),2^(7/10),2^(8/10),2^(9/10),2^(10/10),2^(11/10),2^(12/10)
giEqTmp24 ftgen 3,0,128,-51,24,2,cpsoct(8),60,1,2^(1/24),2^(2/24),2^(3/24),2^(4/24),2^(5/24),2^(6/24),2^(7/24),2^(8/24),2^(9/24),2^(10/24),2^(11/24),2^(12/24),2^(13/24),2^(14/24),2^(15/24),2^(16/24),2^(17/24),2^(18/24),2^(19/24),2^(20/24),2^(21/24),2^(22/24),2^(23/24)

instr 1 ;midi input instrument
/*USE PITCH BEND TO MODULATE NOTE NUMBER UP OR DOWN ONE STEP - ACTUAL INTERVAL IT WILL MODULATE BY WILL BE 1 SEMITONE
;kbend pchbend 0,2

/*ALTERNATIVELY IF USING VIRTUAL MIDI DEVICE OR A KEYBOARD WITH NO PITCH BEND WHEEL, USE CONTROLLERS 17 AND 20
kup ctrl7 1, 1, 0, 1
kdown ctrl7 1, 2, 0, -1
kbend = kup+kdown

inum notnum
kcps tablei inum+kbend, giEqTmp24 ;read cps values from GEN51, scale table using a combination of note number and pitch bend
a1 vco2 0.2, kcps, 4, 0.5
outs a1, a1
endin

instr 2 ;score input instrument

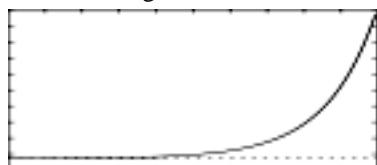
knum line p4, p3, p5 ;gliss using a straight line bewteen p4 and p5 for the entire note duration
kcps tablei knum, giEqTmp24 ;read cps values from GEN51 scale table
a1 vco2 0.2, kcps, 4, 0.5
outs a1, a1
endin

</CsInstruments>
<CsScore>
f 0 3600

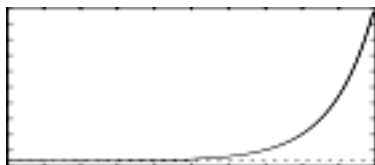
;instr 2. Score input. Gliss from step number p4 to step number p5
;p4 - starting note number
;p5 - ending note number
i 2 0 2 60 61
i 2 + 2 70 58
i 2 + 2 66 66.5
i 2 + 2 71.25 71
e
</CsScore>
</CsoundSynthesizer>

```

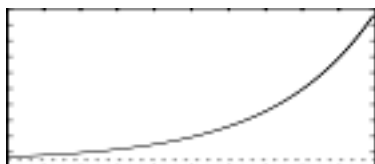
Voici les diagrammes des formes d'onde des routines GEN51 utilisées dans l'exemple :



f 1 0 128 -51 12 2 cpsoct(8) 60 1  $2^{(1/12)}$   $2^{(2/12)}$   $2^{(3/12)}$   $2^{(4/12)}$   $2^{(5/12)}$   $2^{(6/12)}$   $2^{(7/12)}$   $2^{(8/12)}$   
 $2^{(9/12)}$   $2^{(10/12)}$   $2^{(11/12)}$   $2^{(12/12)}$



f 2 0 128 -51 10 2 cpsoct(8) 60 1  $2^{(1/10)}$   $2^{(2/10)}$   $2^{(3/10)}$   $2^{(4/10)}$   $2^{(5/10)}$   $2^{(6/10)}$   $2^{(7/10)}$   $2^{(8/10)}$   
 $2^{(9/10)}$   $2^{(10/10)}$



f 3 0 128 -51 24 2 cpsoct(8) 60 1  $2^{(1/24)}$   $2^{(2/24)}$   $2^{(3/24)}$   $2^{(4/24)}$   $2^{(5/24)}$   $2^{(6/24)}$   $2^{(7/24)}$   $2^{(8/24)}$   
 $2^{(9/24)}$   $2^{(10/24)}$   $2^{(11/24)}$   $2^{(12/24)}$   $2^{(13/24)}$   $2^{(14/24)}$   $2^{(15/24)}$   $2^{(16/24)}$   $2^{(17/24)}$   $2^{(18/24)}$   
 $2^{(19/24)}$   $2^{(20/24)}$   $2^{(21/24)}$   $2^{(22/24)}$   $2^{(23/24)}$   $2^{(24/24)}$

## Crédits

Auteur : Gabriel Maldonado



# GEN52

GEN52 — Crée une table à plusieurs canaux entrelacés à partir des tables source spécifiées, dans le format attendu par l'opcode *ftconv*.

## Description

*GEN52* crée une table à plusieurs canaux entrelacés à partir des tables source spécifiées, dans le format attendu par l'opcode *ftconv*. Il peut aussi être utilisé pour extraire un canal d'une table multicanaux et le stocker dans une table mono normale, copier des tables en omettant certains échantillons, ajouter un délai, ou stocker en ordre inverse, etc.

Il faut donner trois paramètres pour chaque canal à traiter. *fsrc* déclare le numéro de la ftable source. Le paramètre *offset* spécifie un décalage pour le fichier source. S'il est différent de 0, le fichier source n'est pas lu depuis le début, un nombre *offset* de valeurs étant ignorées. L'*offset* est utilisé pour déterminer le numéro de canal à lire depuis les ftables entrelacées, par exemple pour le canal 2, *offset* doit valoir 1. Il peut aussi être utilisé pour fixer un décalage de lecture sur la table source. Ce paramètre donne des valeurs absolues, si bien que si l'on désire un décalage de 20 unités d'échantillonnage pour une ftable à deux canaux, *offset* doit valoir 40. Le paramètre *srcchnls* est utilisé pour fixer le nombre de canaux dans la ftable source. Ce paramètre fixe la taille du pas de progression lors de la lecture de la ftable source.

Quand il y a plus d'un canal (*nchannels* > 1), les ftables source sont entrelacées dans la table nouvellement créée.

Si la ftable source est finie avant que la ftable destination ne soit remplie, les valeurs restantes sont fixées à 0.

## Syntaxe

```
f # date taille 52 ncanaux fsrc1 offset1 srcchnls1 [fsrc2 offset2 srcchnls2 ... fsrcN offsetN srcchnlsN]
```

## Exemples

```
; tables sources
f 1 0 16384 10 1
f 2 0 16384 10 0 1
; crée une table avec 2 canaux entrelacés
f 3 0 32768 -52 2 1 0 1 2 0 1
; extrait le premier canal de la table 3
f 4 0 16384 -52 1 3 0 2
; extrait le second canal de la table 3
f 5 0 16384 -52 1 3 1 2
```

Voici un exemple complet de la routine GEN52. Il utilise le fichier *gen52.csd* [examples/gen52.csd].

### Exemple 1234. Exemple de la routine GEN52.

Voir les sections *Audio en Temps Réel* et *Options de Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac ;;realtime audio out
;-iadc ;;uncomment -iadc if realtime audio input is needed too
```

```

; For Non-realtime ouput leave only the line below:
; -o gen52.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

garvb init 0
gaW init 0
gaX init 0
gaY init 0

itmp ftgen 1, 0, 64, -2, 2, 40, -1, -1, -1, 123, \
      1, 13.000, 0.05, 0.85, 20000.0, 0.0, 0.50, 2, \
      1, 2.000, 0.05, 0.85, 20000.0, 0.0, 0.25, 2, \
      1, 16.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2, \
      1, 9.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2, \
      1, 12.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2, \
      1, 8.000, 0.05, 0.85, 20000.0, 0.0, 0.35, 2

itmp ftgen 2, 0, 262144, -2, 0
      spat3dt 2, -0.2, 1, 0, 1, 1, 2, 0.005

itmp ftgen 3, 0, 262144, -52, 3, 2, 0, 4, 2, 1, 4, 2, 2, 4

instr 1

a1 vco2 1, 440, 10
kfrq port 100, 0.008, 20000
a1 butterlp a1, kfrq
a2 linseg 0, 0.003, 1, 0.01, 0.7, 0.005, 0, 1, 0
a1 = a1 * a2 * 2
      denorm a1
      vincr garvb, a1
aw, ax, ay, az spat3di a1, p4, p5, p6, 1, 1, 2
      vincr gaW, aw
      vincr gaX, ax
      vincr gaY, ay

endin

instr 2

      denorm garvb
; skip as many samples as possible without truncating the IR
arW, arX, arY ftconv garvb, 3, 2048, 2048, (65536 - 2048)
aW = gaW + arW
aX = gaX + arX
aY = gaY + arY
garvb = 0
gaW = 0
gaX = 0
gaY = 0

aWre, aWim hilbert aW
aXre, aXim hilbert aX
aYre, aYim hilbert aY
aWXr = 0.0928*aXre + 0.4699*aWre
aWXiYr = 0.2550*aXim - 0.1710*aWim + 0.3277*aYre
aL = aWXr + aWXiYr
aR = aWXr - aWXiYr

outs aL, aR

```

```

endin

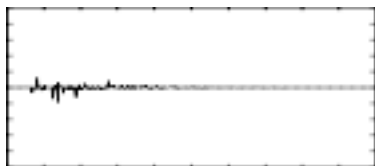
</CsInstruments>
<CsScore>

i 1 0 0.5 0.0 2.0 -0.8
i 1 1 0.5 1.4 1.4 -0.6
i 1 2 0.5 2.0 0.0 -0.4
i 1 3 0.5 1.4 -1.4 -0.2
i 1 4 0.5 0.0 -2.0 0.0
i 1 5 0.5 -1.4 -1.4 0.2
i 1 6 0.5 -2.0 0.0 0.4
i 1 7 0.5 -1.4 1.4 0.6
i 1 8 0.5 0.0 2.0 0.8
i 2 0 10
e

</CsScore>
</CsoundSynthesizer>

```

Voici le diagramme de la forme d'onde de la routine GEN52 utilisée dans l'exemple :



f 3 0 262144 -52 3 2 0 4 2 1 4 2 2 4

## Crédits

Auteur : Istvan Varga

# GEN53

GEN53 — Crée une table de réponse impulsionnelle à phase linéaire ou minimale à partir d'une table source contenant une réponse en fréquence ou une réponse impulsionnelle.

## Description

GEN53 crée une table de réponse impulsionnelle avec soit une phase linéaire soit une phase minimale. La source peut soit être une réponse fréquentielle soit une réponse impulsionnelle, stockée dans une table existante. Une autre table de fonction peut être utilisée facultativement comme fenêtre sur le signal en entrée et/ou en sortie.

## Syntaxe

```
f # time size 53 fsrc [mode fwin]
```

fsrc - table de fonction source. Si c'est une réponse impulsionnelle, sa taille doit correspondre à celle de la table de fonction créée. Si c'est une réponse fréquentielle, la taille de la fonction créée doit être deux fois plus grande que celle de la source. Les tailles doivent être des puissances de deux.

mode - facultatif, le mode opératoire, somme de (a) l'entrée, réponse en fréquence (0) ou réponse impulsionnelle (1), (b) la sortie, phase linéaire (0) ou phase minimale (2), (c) fenêtrage, aucun (0), entrée (4), et/ou sortie (8). Le mode par défaut (0) est réponse en fréquence en entrée, sortie à phase linéaire, pas de fenêtrage.

fwin - table de fonction de la fenêtre, dont la taille doit être une puissance de deux mais pas forcément égale à la taille de la table de fonction créée.

## Exemple

Voici un exemple complet du générateur GEN53. Il utilise le fichier *gen53.csd* [examples/gen53.csd].

### Exemple 1235. Exemple du générateur GEN53.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
-o dac
</CsOptions>
<CsInstruments>

instr 1

a1 diskin "fox.wav"
a2 ftconv a1, p5, 256
   out a2*p4

endin

</CsInstruments>
<CsScore>
```

```
; impulse response
f1 0 131072 1 "ir.wav" 0 0 1
; minimum-phase version
f2 0 131072 53 1 3

; Hann window
f3 0 1024 20 1 1
; low-pass frequency response
f4 0 1024 7 0 100 0 24 1 900 1
; low-pass linear-phase IR
f5 0 2048 53 4 4 3

;          scale  IR
i1  0  3 0.25  2
i1  +  3 1      5
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Istvan Varga

# GENtanh

"tanh" — Génère une table avec les valeurs de la fonction tanh.

## Description

Crée une table avec les valeurs de la fonction tanh.

## Syntaxe

```
f # time size "tanh" start end rescale
```

## Initialisation

*size* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*).

*start, end* -- la première et la dernière valeur à mémoriser ; le GEN dessine une courbe qui va de *start* à *end* : *tanh(start)* ... *tanh(end)*. Les points mis en mémoire sont répartis uniformément entre ces deux valeurs sur la longueur de la table.

*rescale* -- s'il est différent de zéro, la table n'est pas normalisée.

## Exemples

Voici un exemple simple de la routine GENtanh. Il utilise le fichier *gentanh.csd* [exemples/gentanh.csd].

### Exemple 1236. Exemple de la routine GENtanh.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gentanh.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

aswp linseg 0.01, p3*.5, .49, p3*.5, 0.01 ;index sweep function
aindex poscil aswp, 110, 1 ;sound to waveshape
atable table1 aindex, p4, 1, .5 ;waveshape index
aenv linen 0.8, .01, p3, .02 ;amplitude envelope
asig = (atable*aenv)*p5 ;impose envelope and scale
asig dcblock2 asig ;get rid of DC
```

```
outs    asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 8192 10 1 ;sine wave
f 2 0 8192 "tanh" -100 100 0 ;symmetrical transfer fuction
f 3 0 8192 "tanh" -10 10 0 ;symmetrical
f 4 0 8192 "tanh" 0 10 0 ;not symmetrical

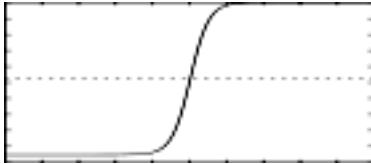
i1 0 3 2 1
i1 + 3 3 1
i1 + 3 4 2

e
</CsScore>
</CsoundSynthesizer>
```

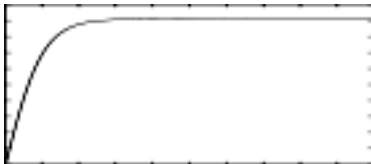
Voici les diagrammes des formes d'onde des routines GENtanh utilisées dans l'exemple :



f 2 0 8192 "tanh" -100 100 0 - beaucoup de distorsion



f 3 0 8192 "tanh" -10 10 0 - moins de distorsion que f2



f 4 0 8192 "tanh" -10 15 0

## Voir aussi

*GENexp* and *GENsone*.

Plus d'information sur cette routine : <http://www.csoundjournal.com/issue11/distortionSynthesis> [<http://www.csoundjournal.com/issue11/distortionSynthesis.html>], écrit par Victor Lazzarini

## Crédits

Ecrit par John ffitch

# GENexp

"exp" — Génère une table dont les valeurs proviennent de la fonction exp.

## Description

Crée une table avec des valeurs de la fonction exp.

## Syntaxe

```
f # time size "exp" start end rescale
```

## Initialisation

*size* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*).

*start, end* -- la première et la dernière valeur à mémoriser. Le GEN dessine une courbe allant de *start* à *end*: *exp(start)* ... *exp(end)*. Les points mis en mémoire sont répartis uniformément entre ces deux valeurs sur la longueur de la table.

*rescale* -- s'il est différent de zéro, la table n'est pas normalisée.

## Exemples

Voici un exemple simple de la routine GENexp. Il utilise le fichier *genexp.csd* [exemples/genexp.csd].

### Exemple 1237. Exemple de la routine GENexp.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o genexp.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

aswp linseg 0.01, p3*.5, .49, p3*.5, 0.01 ;index sweep function
aindex poscil aswp, 110, 1 ;sound to waveshape
atable tablei aindex, p4, 1, .5 ;waveshape index
aenv linen 0.8, .01, p3, .02 ;amplitude envelope
asig = (atable*aenv)*p5 ;impose envelope and scale
asig dcblock2 asig ;get rid of DC
```



```
outs    asig, asig

endin
</CsInstruments>
<CsScore>
f 1 0 8192 10 1 ;sine wave
f 2 0 8192 "exp" 0 15 0
f 3 0 8192 "exp" 0 3 0

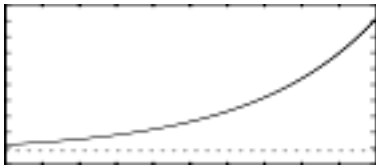
i1 0 3 2 2
i1 + 3 3 3

e
</CsScore>
</CsoundSynthesizer>
```

Voici les diagrammes des formes d'onde des routines GENexp utilisées dans l'exemple :



f 2 0 8192 "exp" 0 15 0



f 3 0 8192 "exp" 0 3 0

## Voir aussi

*GENexp* and *GENsone*.

Plus d'information sur cette routine : <http://www.csoundjournal.com/issue11/distortionSynthesis> [<http://www.csoundjournal.com/issue11/distortionSynthesis.html>], écrit par Victor Lazzarini

## Crédits

Ecrit par Victor Lazzarini

# GENsone

"sone" — Génère une table contenant des valeurs de sonie.

## Description

Crée une ftable avec des valeurs de sonie à puissance constante.

## Syntaxe

```
f # time size "sone" start end equalpoint rescale
```

## Initialisation

*size* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*).

*start, end* -- la première et la dernière valeur à mémoriser. Les points mis en mémoire sont répartis uniformément entre ces deux valeurs sur la longueur de la table.

*equalpoint* -- le point de la courbe où les valeurs d'entrée et de sortie sont égales.

*rescale* -- s'il est différent de zéro, la table n'est pas normalisée.

la table est remplie par la fonction  $x * \text{POWER}(x / \text{equalpoint}, \text{FL}(33.0) / \text{FL}(78.0))$  pour *x* compris entre les points *start* et *end*. C'est la courbe d'intensité en sone.

## Exemples

Voici un exemple simple de la routine *GENsone*. Il utilise le fichier *gensone.csd* [examples/gensone.csd].

### Exemple 1238. Un exemple simple de la routine GENsone.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o gensone.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ; simple oscillator with loudness correction.

kcps = cpspch(p4)
kenv linseg 0, p3*0.25, 1, p3*0.75, 0 ;amplitude envelope
kamp tablei 16384 *kenv, 2
asig oscil kamp, kcps, 1
```

```

outs asig, asig

endin

instr 2 ;neutral oscillator to compare with

kcps = cpspch(p4)
kenv linseg 0, p3*0.25, 1, p3*0.75, 0 ;amplitude envelope
asig oscil kenv, kcps, 1
outs asig, asig

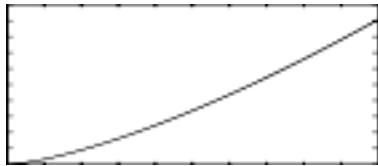
endin

</CsInstruments>
<CsScore>
f 1 0 16384 10 1 ;sine wave
f 2 0 16385 "sone" 0 32000 32000 0

s
f 0 1 ;1 second of silence before we start...
s
i 1 0 2 7.00
i 1 + . 7.01
i 1 + . 8.02
i 1 + . 8.03
s
i 2 0 2 7.00
i 2 + . 7.01
i 2 + . 8.02
i 2 + . 8.03
e
</CsScore>
</CsoundSynthesizer>

```

Voici le diagramme de la forme d'onde de la routine GENsone utilisée dans l'exemple :



f 2 0 16385 "sone" 0 32000 32000 0

## Voir aussi

Plus d'information sur le sone : <http://fr.wikipedia.org/wiki/Sone> [<http://fr.wikipedia.org/wiki/Sone>]

## Crédits

Ecrit par John ffitich

# GENquadbezier

"quadbezier" — Génère une table avec les valeurs d'une fonction de Bézier quadratique.

## Description

Opcode du greffon quadbezier.

Cette routine crée une ftable de segments construits par les chemins tracés par le fonction de Bézier quadratique.

## Syntaxe

```
f # time size "quadbezier" y1 cx1 cy1 x2 y2 [cx2 cy2 x3 y3 ...]
```

## Initialisation

*x2, x3, etc.* -- Positions dans la table auxquelles la valeur *y* suivante sera atteinte. Doivent être en ordre croissant. On suppose que *x1* vaut 0. Si la dernière valeur est inférieure à la taille, les positions restantes seront mises à zéro. Ne doivent pas être négatives.

*y1, y2, y3, etc.* -- Valeurs charnière atteintes à la position spécifiée par la valeur *x* précédente.

*cx1, cx2, cx3, etc.* -- Coordonnées *x* virtuelles pour les différents points de contrôle. *cx[n]* peut être égal ou supérieur à *x[n]* et égal ou inférieur à *x[n+1]*. Ne doivent pas être négatives mais peuvent être nulles.

*cy1, cy2, cy3, etc.* -- Coordonnées *y* virtuelles pour les différents points de contrôle.

## Exrmples

Voici un exemple de la routine GENquadbezier. Il utilise le fichier *genquadbezier.csd* [exemples/genquadbezier.csd].

### Exemple 1239. Exemple de la routine GENquadbezier.

Voir les sections *Audio en temps réel* et *Options de la ligne de commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac   ;;realtime audio out
;-iadc   ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o quadbezier.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
```

```

Odbfs = 1

instr 1
kndx phasor 1/p3
kenv tablei kndx, p4, 1
asig poscil kenv, 440, 1
    outs asig, asig

endin

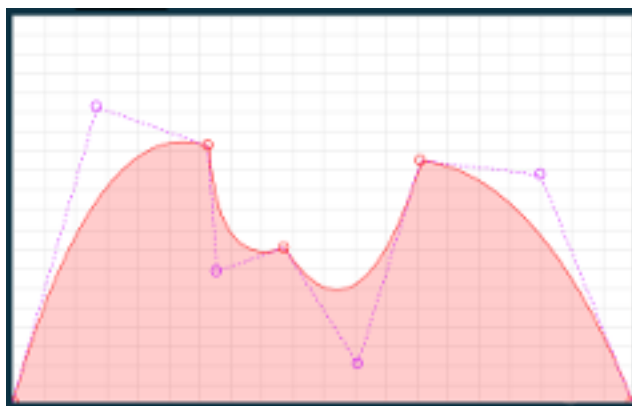
</CsInstruments>
<CsScore>
f 1 0 32768 10 1
f 2 0 1024 "quadbezier" 0 140 0.61 324 0.53 338 0.27 449 0.32 571 0.08 675 0.5 873 0.47 1024 0
f 3 0 1024 "quadbezier" 0 92 0.04 94 0.25 177 0.58 373 0.39 537 0.15 675 0.5 910 0.68 1024 0
f 4 0 1024 "quadbezier" 0 196 0.68 537 0.71 873 0.7 1024 0

i 1 0 4 2
i 1 4 4 3
i 1 8 4 4
</CsScore>

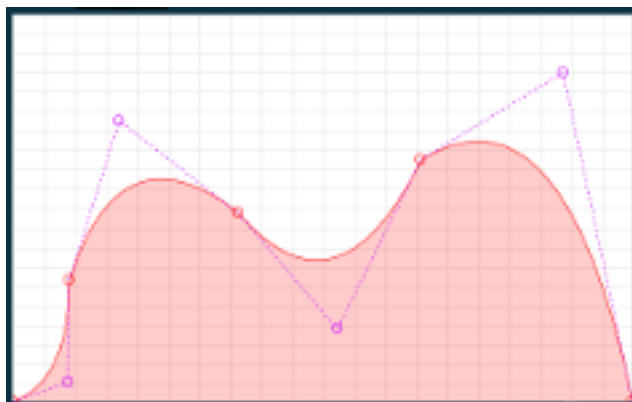
</CsoundSynthesizer>

```

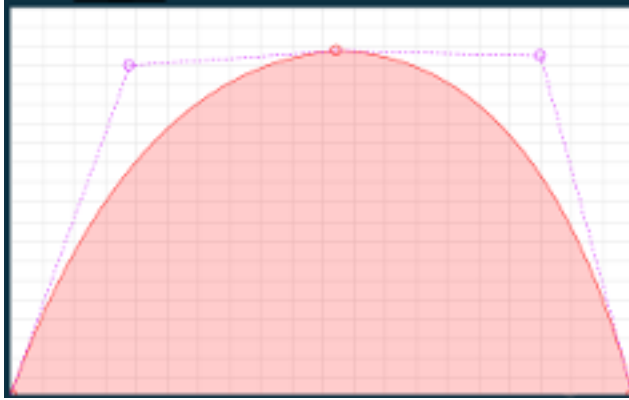
Voici les diagrammes des enveloppes dessinés par la routine GENQuadbezier dans l'exemple ci-dessus :



f 2 0 1024 "quadbezier" 0 140 0.61 324 0.53 338 0.27 449 0.32 571 0.08 675 0.5 873 0.47 1024 0



f 3 0 1024 "quadbezier" 0 92 0.04 94 0.25 177 0.58 373 0.39 537 0.15 675 0.5 910 0.68 1024 0



f 4 0 1024 "quadbezier" 0 196 0.68 537 0.71 873 0.7 1024 0

## Voir aussi

Pour les références, consulter les commentaires dans le *code source* [<https://github.com/csound/csound/blob/develop/Opcodes/quadbezier.c>].

## Crédits

Ecrit par Guillermo Senna

2016

# GENfarey

"farey" — Remplit une table avec la suite de Farey  $F_n$  d'ordre  $n$ .

## Description

Opcode du greffon fareygen.

Une suite de Farey  $F_n$  d'ordre  $n$  est une liste de fractions irréductibles comprises entre 0 et 1 et en ordre croissant. Leurs dénominateurs sont inférieurs ou égaux à  $n$ . Cela signifie qu'une fraction  $a/b$  appartient à  $F_n$  si  $0 \leq a \leq b \leq n$ . Le numérateur et le dénominateur de chaque fraction sont toujours premiers entre eux. 0 et 1 sont compris dans  $F_n$  sous la forme des fractions 0/1 et 1/1. Par exemple  $F_5 = \{0/1, 1/5, 1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 1/1\}$ . Quelques propriétés des suites de Farey :

- Si  $a/b$  et  $c/d$  sont deux termes successifs de  $F_n$ , alors  $bc - ad = 1$ .
- Si  $a/b, c/d, e/f$  sont trois termes successifs de  $F_n$ , alors :  $c/d = (a+e) / (b+f)$ . Dans ce cas, on dit que  $c/d$  est la fraction médiane entre  $a/b$  et  $e/f$ .
- Si  $n > 1$ , alors il n'existe pas de termes successifs de  $F_n$  ayant le même dénominateur.

La longueur de la suite de Farey  $F_n$  est déterminée par  $|F_n| = 1 + \text{SOMME sur } n (\phi(m))$  où  $\phi(m)$  est l'indicatrice d'Euler, qui donne le nombre d'entiers  $\leq m$  premiers avec  $m$ .

Quelques valeurs de la longueur de  $F_n$  en fonction de  $n$  :

<b>n</b>	<b><math>F_n</math></b>
1	2
2	3
3	5
4	7
5	11
6	13
7	19
8	23
9	29
10	33
11	43
12	47
13	59
14	65
15	73
16	81
17	97
18	103
19	121

n	$F_n$
20	129

## Syntaxe

`f # time size "farey" fareynum mode`

## Initialisation

*size* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance-de-2 plus 1 (voir l'instruction *f*).

*fareynum* -- l'entier n pour générer la suite de Farey  $F_n$ .

*mode* -- entier définissant le type de sortie à écrire dans la table :

- 0 -- nombres en virgule flottante représentant les éléments de  $F_n$ .
- 1 -- différences entre les éléments successifs de  $F_n$  ; utile pour générer des durées de note par exemple.
- 2 -- seulement les dénominateurs des rapports d'entiers ; utile comme indexation d'autres tables ou instruments par exemple.
- 3 -- comme le mode 2 mais normalisée. output.
- 4 -- comme le mode 0 mais avec 1 ajouté à chaque terme ; utile pour générer des tables pour les opcodes d'accordage, par exemple *cps2pch*.

## Exemples

```
f1 0 -23 "farey" 8 0
```

Génère la suite de Farey  $F_8$ . La table contient les 23 éléments de  $F_n$  en virgule flottante.

```
f1 0 -18 "farey" 7 1
```

Génère la suite de Farey  $F_7$ . La table contient les 18 différences entre les termes de  $F_7$ , c'est-à-dire les différences  $r_{i+1} - r_i$ , où r est le ième élément de  $F_n$ .

```
f1 0 -43 "farey" 11 2
```

Génère la suite de Farey  $F_{11}$ . La table contient les dénominateurs des 43 fractions de  $F_{11}$ .

```
f1 0 -43 "farey" 11 3
```

Génère la suite de Farey  $F_{11}$ . La table contient les dénominateurs des 43 fractions de  $F_{11}$ , chacun de ceux-ci étant divisé par 11, c'est-à-dire normalisé.

```
f1 0 -18 "farey" 7 4
```

Génère la suite de Farey  $F_7$ . La table contient les fractions de  $F_7$  comme dans le mode 0, mais la durée 'l' est ajouté à chaque élément de la table.

### Exemple 1240. Un exemple simple de la routine GENfarey.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.



```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr=44100
ksmps=10
nchnls=1

instr 4
  kndx init 0 ; read out elements of F_8 one by one and print to file
  if (kndx < 23) then
    kelem tab kndx, 1
    fprintks "farey8table.txt", "%2.6f\\n", kelem
    kndx = kndx+1
  endif
endin
</CsInstruments>
<CsScore>
; initialise integer for Farey Sequence F_8
f1 0 -23 "farey" 8 0
      ; if mode=0 then the table stores all elements of the Farey Sequence
      ; as fractions in the range [0,1]
i4 0 1
e
</CsScore>
</CsoundSynthesizer>

```

Voici un exemple complet de la routine GENfarey. Il utilise le fichier *genfarey-2.csd* [examples/genfarey-2.csd].

### Exemple 1241. Un autre exemple de la routine GENfarey.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```

<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc      ;;uncomment -iadc for RT audio input as well
; For Non-realtime ouput leave only the line below:
; -o genfarey.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

; GENfarey creates table gidelta.
; The table contains the delta values of Farey Sequence 7 (p5=7).
; They are used as Inter Onset Intervals (IOIs) or event durations.
; If p6 is set to 1 for IOI output then the length of the table (p3=-18) is -(|F_7| - 1)
; Remember that a negative sign is for non-power-of-2 table lengths.
; The negative sign in front of the GEN number prevents post-normalisation of its values.

gidelta ftgen 0,0,-18,"farey",7,1

; Use GENfarey with p6 set to 2 to generate the denominators of fractions of F_7
; this is used in this example as factors to create a series of pitches:
gimult ftgen 0,0,-18,"farey",7,2

;----- loop and trigger instrument 901 using a Farey Sequence polyrhythm

```

```

instr 1
kindx init 0
kindx2 init 0
ktrigger init 0
ktime_unit init p6
kstart init p4
kloop init p5
kinitndx init 0
kfn_times init gdelta
knote init 60
kbasenote init p8
ifundam init p7
ktrigger seqtime ktime_unit, kstart, kloop, kinitndx, kfn_times
  if (ktrigger > 0 ) then
    kpitch = cpspch(ifundam)
    kmult tab kindx2, gimult
    kpitch = kpitch * kmult
    knote = kbasenote + kmult
    event "i", 901, 0, .4, .10, kpitch, kpitch * .9, 0.4, 5, .75, .8, 1.0, .15, .0, .125, .12
    kindx = kindx + 1
    kindx = kindx % kloop
    kindx2 = kindx2 + 1
    kindx2 = kindx2 % kloop
  endif
endin

;----- basic 2 Operators FM algorithm -----
instr 901
inotedur = p3
imaxamp = p4 ;ampdb(p4)
icarrfreq = p5
imodfreq = p6
ilowndx = p7
indxdiff = p8-p7
knote = p27
aampenv linseg p9, p14*p3, p10, p15*p3, p11, p16*p3, p12, p17*p3, p13
adevenv linseg p18, p23*p3, p19, p24*p3, p20, p25*p3, p21, p26*p3, p22
amodosc oscili (ilowndx+indxdiff*adevenv)*imodfreq, imodfreq, 10
acarosc oscili imaxamp*aampenv, icarrfreq+amodosc, 10
outs acarosc, acarosc
endin
</CsInstruments>
<CsScore>
f10 0 4096 10 1 ;sine wave
; p4 kstart := index offset into the Farey Sequence
; p5 kloop := end index into Farey Seq.
; p6 timefac := time in seconds for one loop to complete
; p7 fundam := fundamental of the FM instrument
; p8 basenote:= root pitch of the midi voice output
; note that pitch structures of the midi file output are not equivalent to the
; ones used for the FM real-time synthesis.

; start dur kstart kloop timefac fundam. basenote
i1 0.0 44 0 18 2 6.05 60
i1 4 30 0 18 3 7.05 72
i1 34 12 9 18 3 7.05 72
i1 10 12 0 18 1.5 8 84
i1 22 12 0 9 1.5 8 84
i1 15 16 0 18 1 5 48
i1 22 20 5 17 1.7 4 36

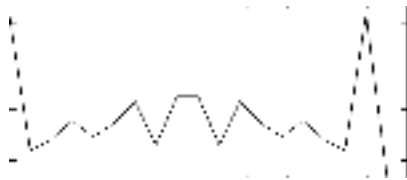
i1 46 20 3 11 2.5 7.04 71
i1 51 20 5 13 2.5 7.06 72

i1 73.5 1.5 11 18 1.5 5.05 48
i1 75 1 12 18 1 6.03 58
e

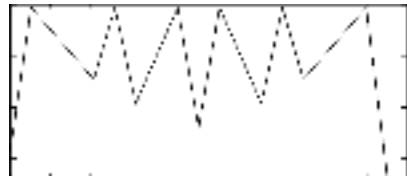
```

```
</CsScore>  
</CsoundSynthesizer>
```

Voici les diagrammes des formes d'onde de la routine GENfarey utilisées dans l'exemple :



gidelta ftgen 100, 0, -18, "farey", 7, 1 - valeurs delta de la suite de Farey 7



gimult ftgen 101, 0, -18, "farey", 7, 2 - génère les dénominateurs des fractions de F\_7

## Crédits

Auteur : Georg Boenn  
Université de Glamorgan  
2010

Nouveau dans la version 5.13 de Csound

# GENwave

"wave" — Génère une fonction d'ondelette à support compact.

## Description

Crée une ondelette à support compact, fonction de changement d'échelle ou paquet d'ondelette. La fonction de sortie est obtenue par déconvolution de la réponse impulsionnelle du filtre miroir correspondant. Cette procédure est appliquée de manière itérative.

Le banc de filtres utilisé dans la transformation en ondelettes discrète classique ne s'étend que vers les basses fréquences. Au contraire, la transformée en paquets d'ondelettes permet toutes les directions possibles d'expansion de l'arbre. La suite de filtres miroir utilisés dans la déconvolution est déterminée par la forme binaire de la valeur de *seq*. "0" correspond à un filtre passe-bas et "1" à un filtre passe-haut.

Le nombre de pas d'itération est déterminé par la longueur de filtre et par la taille de la table de fonction. Ainsi pour une longueur de filtre de 8 et une taille de table de 256, il y a  $\log_2(256/8) = 5$  itérations.

## Syntaxe

```
f # time size "wave" fnsf seq rescale
```

## Initialisation

*size* -- nombre de points dans la table. Doit être une puissance de 2 ou une puissance de 2 plus 1 (voir l'instruction *f*).

*fnsf* -- table pré-existante avec les coefficients de la fonction de changement d'échelle.

*seq* -- nombre entier non négatif qui correspond à la suite de filtres miroir passe-bas et passe-haut durant la procédure de déconvolution.

*rescale* -- s'il est différent de zéro, la table ne change pas d'échelle.

## Exemples

Voici un exemple de la routine GENwave. Il utilise le fichier *genwave.csd* [exemples/genwave.csd].

### Exemple 1242. Exemple de la routine GENwave.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac          -iadc     -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o 0dbfs.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
nchnls = 1
```

```

Odbfs = 1

zakinit 3,1

instr 1 ; wavelet synth instrument
iamp = p4; scaling factor of wavelets
ifreq = p5; frequency of wavelets
itab = p6; selected wavelet function
inum = p7; number of wavelets to be created
a1 osciln p4, p5, p6, p7
out a1
endin

instr 2 ; wavelet analysis intrument
a1 soundin "fox.wav"
; Decomposition Structure:
; 1 LEVEL 2 LEVEL
; HP->ah1
; a1->| HP(up2)->ah2
; LP->a11->|
; LP(up2)->a12
;
ain = a1*.5; attenuate input signal
; since wavelet coefficients
; could reach big values
ah1 dconv ain,ftlen(8),8
a11 dconv ain,ftlen(7),7
ah2 dconv a11,ftlen(10),10
a12 dconv a11,ftlen(9),9

zaw ah1,0
zaw a11,1
zaw ah2,2
zaw a12,3

aout zar p4
out aout
zacl 0,3
endin

</CsInstruments>
<CsScore>

; First of all, we need several FIR filters which are capable
; to produce wavelet families.
; One can input filter coefficients manually using GEN02
; or read them from text file.
; Most of compact-supported wavelet coefficients can be obtained from
; Wavelet Browser by PyWavelets wavelets.pybytes.com
; You can select family and order of filter
; then copy desired coefficients into txt file.
; Notice that for correct interpretation of results you should use
; coeffs of Decomposition low-pass filter.

; Daubechies 2
f 1 0 4 -2 -0.1294095226 0.2241438680 0.8365163037 0.4829629131
; Symlet 10
f 2 0 0 -23 "sym10.txt"

; Now we want to produce some wavelet granules.
; They can be used in wavelet synthesis etc.
; Tables of large sizes should produce smoother wavelets.
; We take array of filter coefficients from ftable 1
; and deconvolve it until output length of 16384.
; The order of filters through the deconvolution process
; is given by 14 which is 1110 in binary.
; So the first filter is LP ('0') and others are HP ('1').

```

```
f 3 0 16384 "wave" 1 14 0
f 4 0 16384 "wave" 2 1 0
f 5 0 16384 "wave" 2 7 0
f 6 0 16384 "wave" 2 6 0
; The main purpose of using wavelets is wavelet transform.
; It is not that easy to perform a classic DWT in Csound since downsampling
; of audio signal is needed at each step of wavelet decomposition.
; Anyway, using GENwave it is possible to create a number of upsampled
; wavelets and perform a so-called undecimated wavelet transform
; aka stationary wavelet transform (and it is even better).
; So we need some upsampled childs of mother wavelet.
f 7 0 16 "wave" 1 0 -1 ;db2 scaling function for 1st iteration
f 8 0 16 "wave" 1 1 -1 ;db2 wavelet function for 1st iteration
f 9 0 32 "wave" 1 0 -1 ;db2 scaling function for 2nd iteration
f 10 0 32 "wave" 1 1 -1 ;db2 wavelet function for 2nd iteration

; Let's hear how some wavelets could sound..
; amp frq wave times
i 1 0 1 0.6 15 3 8
i 1 0.5 . 0.9 20 4 5
i 1 0.9 . 0.7 8 5 .
i 1 1.1 . 0.4 30 6 9

; Now try to decompose input file using wavelets
i 2 2 4 1; approximation 1st level
i 2 5 . 2; details 2nd level
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

Ingrid Daubechies. Ten Lectures on Wavelets, SIAM 1992.

Pedro A.G. Batista. An Introduction to Sound Synthesis with Wavelet Packets, Csound Book CD Chapters, MIT 2000.

Victor M. Wickerhauser. Acoustic Signal Compression with Wavelet Packets, Yale 1992.

## Crédits

Ecrit par Gleb Rogozinsky

Université du film et de la télévision de Saint-Pétersbourg

Saint-Pétersbourg, Russie

Avril 2012

# GENpadsynth

"padsynth" — Génère une table d'échantillons au moyen de l'algorithme padsynth.

## Description

Opcode du greffon padsynth.

L'algorithme "padsynth" de Paul Octavian Nasca ajoute une largeur de bande à chaque partiel d'une forme d'onde périodique. Cette largeur de bande est entendue comme couleur, mouvement et enrichissement du son.

D'abord la forme d'onde est définie par l'utilisateur comme une série d'harmoniques. Puis une largeur de bande est ajoutée en étalant indépendamment chaque partiel de la forme d'onde originale depuis une fréquence vers ses voisines en suivant une fonction "profil" : une courbe gaussienne, une fonction carrée ou une exponentielle croissante puis décroissante.

On peut considérer les partiels de la forme d'onde originale comme des échantillons dans une transformée de Fourier discrète de la forme d'onde. Normalement il n'y a pas de correspondance exacte point à point entre les fréquences des échantillons (bins de fréquence) de la transformée de Fourier discrète et les fréquences des harmoniques de la forme d'onde originale, car toute fréquence dans l'inverse de la transformée de Fourier discrète pourrait être synthétisée par interférence entre n'importe quel nombre de bins. Cependant, l'algorithme padsynth utilise une petite astuce pour créer cette correspondance. La transformée de Fourier discrète est simplement rendue si grande que la fréquence de chaque harmonique de la forme d'onde originale sera très proche de la fréquence du bin correspondant dans la transformée de Fourier. Une fois cette correspondance créée, le profil de largeur de bande peut être appliqué en le centrant sur le bin de fréquence de l'harmonique original, en pondérant le profil par la largeur de bande et en multipliant simplement l'harmonique original par chaque échantillon du profil et en ajoutant le produit au bin correspondant de la transformée de Fourier.

Plus les fréquences des harmoniques augmentent, plus leur largeur de bande peut augmenter de manière facultative ou (moins souvent) diminuer.

Une fois les harmoniques étalés de cette manière, la transformée de Fourier discrète peut recevoir des phases aléatoires et est ensuite simplement inversée pour synthétiser la forme d'onde désirée, que l'on peut utiliser comme table d'onde d'un oscillateur numérique.

Nota bene : la taille de la table de fonction ne reflète pas nécessairement une période de la forme d'onde qu'elle contient. La fréquence fondamentale doit être utilisée pour générer la hauteur désirée par un oscillateur utilisant la table de fonction, par exemple

```
oscillator_hz = desired_hz * (sr / padsynth_size / fundamental_hz)
```

## Syntaxe

```
f # score_time table_size "padsynth" fundamental_frequency  
  partial_bandwidth partial_scale harmonic_stretch profile_shape profile_shape_parameter  
  partial1_amplitude [partial2_amplitude ...]
```

## Initialisation

*table\_size* -- Taille de la table de fonction. Doit être grande, par exemple  $2^{18} == 262144$ . Doit être une puissance de 2 ou une puissance de 2 plus 1 (voir l'*instruction f*).

*fundamental\_frequency* -- fréquence fondamentale pour la table générée.

*partial\_bandwidth* -- largeur de bande de chaque partiel en cents.

*partial\_scale* -- facteur de pondération pour la largeur de bande de chaque partiel (log de l'augmentation/diminution avec la fréquence d'harmonique, 0 signifiant pas d'étirement ni de compression).

*harmonic\_stretch* -- Etirement/compression harmonique pour tous les partiels (1 est harmonique).

*profile\_shape* -- Nombre indiquant la forme du profil de largeur de bande : 1 = gaussien, 2 = carré et 3 = exponentiel.

*profile\_shape\_parameter* -- Paramètre passé à la fonction générant la forme du profil, par exemple un exposant.

*partial1\_amplitude*, *partial2\_amplitude*, ... -- Amplitudes pour chaque harmonique (peut valoir zéro).

## Exemples

Voici un exemple de la routine GENpadsynth. Il utilise le fichier *padsynth\_gen.csd* [examples/pad-synth\_gen.csd].

### Exemple 1243. Un exemple de la routine GENpadsynth.

```
<CsoundSynthesizer>
<CsOptions>
-odac
</CsOptions>
<CsInstruments>
sr=44100
ksmps=1
nchnls=2
0dbfs=2000

gispec_len init 2^18

instr 1
  prints "Plain sine for frequency/amplitude/distortion comparison.\n"
  gi_padsynth_1 ftgenonce 0, 0, gispec_len, 10, 1
  iattack = 0.08
  idecay = 0.1
  isustain = 0.25
  irelease = 0.2
  aenv madsr iattack, idecay, isustain, irelease
  ifreq cpsmidinn p4
  iamp ampdb p5
  ibasefreq = 440 ; can be lower or higher frequency; close to played frequency is said to be best
  ibw_cents = 56.96943 ; width of the peaks, 100 is semitone
  asig poscil iamp, ifreq, gi_padsynth_1
  asig = aenv * asig
  aleft, aright pan2 asig, 0.5
  outs aleft, aright
endin

instr 2
  prints "PadSynth with sine tone.\n"
  ibasehz = 261.625565
  ;          p1 p2 p3          p4          p5          p6 p7          p8 p9 p10 p11
  gi_padsynth_1 ftgenonce 0, 0, gispec_len, "padsynth", ibasehz, p6, 0.0, 1, 1, 1.0, 1
  iattack = 0.08
  idecay = 0.1
  isustain = 0.25
  irelease = 0.2
  aenv madsr iattack, idecay, isustain, irelease
  ifreq cpsmidinn p4
```



```

    iamp ampdb p5
    asig poscil iamp, ifreq*(sr/gispec_len/ibasehz), gi_padsynth_1
    asig = aenv * asig
    aleft, aright pan2 asig, 0.5
    outs aleft, aright
    endin

instr 3
    prints "PadSynth with harmonics.\n"
    ibasehz = 261.625565
    ;
    gi_padsynth_1 ftgenonce 0, 0, gispec_len, "padsynth", ibasehz, p6, 1, 1, 1, 1, 0.7600046992, 0.619999
    iattack = 0.08
    idecay = 0.1
    isustain = 0.25
    irelease = 0.2
    aenv madsr iattack, idecay, isustain, irelease
    ifreq cpsmidinn p4
    iamp ampdb p5
    asig poscil iamp, ifreq*(sr/gispec_len/ibasehz), gi_padsynth_1
    asig = aenv * asig
    aleft, aright pan2 asig, 0.5
    outs aleft, aright
    endin

instr 4
    prints "PadSynth with inharmonic partials.\n"
    ibasehz = 261.625565
    ;
    gi_padsynth_1 ftgenonce 0, 0, gispec_len, "padsynth", ibasehz, p6, 1, 2, 3, 1, 0.7600046992, 0.619999
    iattack = 0.08
    idecay = 0.1
    isustain = 0.25
    irelease = 0.2
    aenv madsr iattack, idecay, isustain, irelease
    ifreq cpsmidinn p4
    iamp ampdb p5
    asig poscil iamp, ifreq*(sr/gispec_len/ibasehz), gi_padsynth_1
    asig = aenv * asig
    aleft, aright pan2 asig, 0.5
    outs aleft, aright
    endin

</CsInstruments>

<CsScore>

i1 0 2 60.00 60
i1 + 2 72.00 60
i1 + 2 84.00 60

i2 7 2 60.00 60 0.3
i2 + 2 72.00 60 0.3
i2 + 2 84.00 60 0.3
i2 + 2 60.00 60 25
i2 + 2 72.00 60 25
i2 + 2 84.00 60 25
i2 + 2 60.00 60 55
i2 + 2 72.00 60 55
i2 + 2 84.00 60 55

i3 26 2 60.00 60 0.3
i3 + 2 72.00 60 0.3
i3 + 2 84.00 60 0.3

```

```
i3 + 2 60.00 60 25
i3 + 2 72.00 60 25
i3 + 2 84.00 60 25
i3 + 2 60.00 60 55
i3 + 2 72.00 60 55
i3 + 2 84.00 60 55

i4 45 2 60.00 60 0.3
i4 + 2 72.00 60 0.3
i4 + 2 84.00 60 0.3
i4 + 2 60.00 60 25
i4 + 2 72.00 60 25
i4 + 2 84.00 60 25
i4 + 2 60.00 60 55
i4 + 2 72.00 60 55
i4 + 2 84.00 60 55

e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

Plus d'information sur padsynth : [www.paulnasca.com/algorithms-created-by-me](http://www.paulnasca.com/algorithms-created-by-me)

## Crédits

Ecrit par Michael Gogins

Nouveau dans la version 6.05

---

# Opcodes de l'orchestre expérimentaux et routines GEN

## Opcodes de l'orchestre expérimentaux

Les opcodes suivants nécessitent la présence d'un GPU de nVidia. Ils sont distribués dans l'espoir qu'ils seront utiles et pour encourager l'utilisation du parallélisme pour le traitement audio. Certains de ces opcodes produisent le même traitement que des opcodes conventionnels existants tandis que d'autres sont nouveaux.

# cudanal

**cudanal** — Génère un *fsig* à partir d'une source audio mono, en utilisant l'analyse par recouvrement-addition d'un vocodeur de phase et un GPU. Expérimental et seulement disponible dans le code source pour le moment.

## Description

Génère un *fsig* à partir d'une source audio mono, en utilisant l'analyse par recouvrement-addition d'un vocodeur de phase et un GPU.

## Syntaxe

```
fsig cudanal ain, ifftsize, ioverlap, iwinsize, iwintype [, iformat] [, iinit]
```

## Initialisation

*ifftsize* -- La taille de la TFR en échantillons. Ne doit pas forcément être une puissance de deux (bien que celles-ci sont particulièrement efficaces), mais doit être paire. Les nombres impairs sont arrondis en interne. *ifftsize* détermine le nombre de bins d'analyse dans *fsig*, soit  $ifftsize/2 + 1$ . Par exemple, si *ifftsize* = 1024, *fsig* contiendra 513 bins d'analyse, ordonnés linéairement de la fréquence fondamentale à la fréquence de Nyquist. La fréquence fondamentale de l'analyse (qui donne en principe la fréquence résoluble la plus basse) est déterminée par  $sr/ifftsize$ . Ainsi, pour l'exemple précédent en supposant que  $sr = 44100$ , la fréquence fondamentale de l'analyse vaut 43.07Hz. En pratique, comme le vocodeur de phase préserve la phase, la fréquence de chaque bin peut dévier de façon bilatérale, si bien que des composantes continues sont enregistrées. Avec un signal fortement tonal, les fréquences des bins adjacents peuvent s'aggréger très étroitement autour des partiels de la source, et les bins inférieurs peuvent même avoir des fréquences négatives.

En principe, la seule raison d'utiliser pour *ifftsize* une valeur qui n'est pas une puissance de deux est de s'adapter à la fréquence fondamentale connue d'une source fortement tonale. Les valeurs décomposables en plusieurs petits facteurs peuvent être presque aussi efficaces que les tailles en puissance de deux ; par exemple : 384, pour une source dont la hauteur est proche du la grave à 110 Hz.

*ioverlap* -- La distance en échantillons (« taille du saut ») entre les trames d'analyse se recouvrant. En principe, doit valoir au moins  $ifftsize/4$ , par exemple 256 dans l'exemple ci-dessus. *ioverlap* détermine le taux d'analyse sous-jacent, soit  $sr/ioverlap$ . Il n'est pas nécessaire que *ioverlap* soit un facteur simple de *ifftsize* ; par exemple, une valeur de 160 sera légale. Le choix de *ioverlap* peut être dicté par l'importance de la modification de hauteur appliquée au *fsig*, s'il y en a une. En règle générale, plus la transposition est importante et plus le taux d'analyse doit être élevé, ce qui implique une plus petite valeur de *ioverlap*. Un taux d'analyse plus élevé peut aussi être plus avantageux avec des sons à transitoires à large bande tels que des tambours (pour lesquels une petite fenêtre d'analyse diminue l'étalement mais augmente le nombre d'erreurs relatives à la fréquence).

Noter qu'il est possible, et raisonnable, d'avoir différents *fsigs* dans un orchestre (même dans le même instrument), évoluant à différents taux d'analyse. Les interactions entre de tels *fsigs* ne sont pas couramment supportées et l'opcode d'affectation de *fsig* ne permet pas la copie entre *fsigs* ayant des propriétés différentes, même si la seule différence est la valeur de *ioverlap*. Cependant, ceci ne conduit pas à une impasse, car il est théoriquement possible d'effectuer une conversion grossière du taux (en particulier par rapport aux fichiers d'analyse en mémoire) comme on le fait dans les techniques du domaine temporel.

*iwinsize* -- la taille en échantillons du filtre de la fenêtre d'analyse (fixé par *iwintype*). Doit valoir au moins *ifftsize*, et peut être utilement plus grande. Bien que d'autres proportions soit permises, il est recommandé que *iwinsize* soit toujours un multiple entier de *ifftsize*, par exemple 2048 dans l'exemple ci-dessus. En interne, la fenêtre d'analyse (Hamming, von Hann) est multipliée par une fonction sinc afin que les amplitudes soient nulles aux frontières de trame. La plus grande taille de fenêtre d'analyse s'est révélée particulièrement importante pour la resynthèse par banc d'oscillateurs (par exemple en utilisant *pvsadsyn*), car elle a pour effet d'augmenter la résolution en fréquence de l'analyse et ainsi, la précision de la resynthèse. Comme noté ci-dessus, *iwinsize* détermine la latence globale du système d'analyse/resynthèse. Dans bien des cas, et particulièrement en absence de transposition, on constate que l'égalité *iwinsize*=*ifftsize* fonctionne très bien et offre la latence la plus faible.

*iwintype* -- La forme de la fenêtre d'analyse. Actuellement, seulement deux choix sont implémentés :

- 0 = fenêtre de Hamming
- 1 = fenêtre de von Hann

Les deux sont aussi supportées par le format de fichier PVOC-EX. Le type de fenêtre est stocké comme attribut interne du fsig, avec les autres paramètres (voir *pvsinfo*). D'autres types pourront être implémentés dans le futur (par exemple la fenêtre de Kaiser, aussi supportée par PVOC-EX), bien qu'une alternative évidente soit de permettre la définition des fenêtres via une table de fonction. Le problème ici est la contrainte de taille en puissance de deux des tables de fonction, ce qui en fait une solution incomplète. La plupart des utilisateurs jugeront que la fenêtre de Hamming est suffisante pour les besoins courants et peut être le choix par défaut.

*iformat* -- (facultatif) Le format d'analyse. Pour le moment un seul format est implémenté par cet opcode :

- 0 = amplitude + fréquence

C'est le format classique du vocodeur de phase ; facile à traiter, et naturel pour la resynthèse par banc d'oscillateurs. Il serait très facile (on pourrait dire tentant) de ne pas traiter une trame de fsig purement comme une trame de vocodeur de phase mais comme une trame de synthèse additive générique. Il est en fait possible d'utiliser un fsig de cette manière, mais il est important de garder à l'esprit que les deux ne sont pas, à strictement parler, directement équivalents.

D'autres formats importants (supportés par PVOC-EX) sont :

- 1 = amplitude + phase
- 2 = complexe (réel + imaginaire)

*iformat* est là au cas où il pourrait être utile par la suite de supporter ces autres formats. Les formats 0 et 1 sont en relation étroite (car la phase est « cyclique » dans les deux cas - il est trivial de convertir de l'un à l'autre), alors que le format complexe pourrait garantir un second type explicite de signal (un « csig ») spécialement pour les traitements à base de convolution, et d'autres traitements dans lesquels le complément des opérateurs arithmétiques peut être utile.

*iinit* -- (facultatif) Ignore la réinitialisation. N'est actuellement implémenté dans aucun de ces opcodes, et il reste à décider s'il serait de quelque utilité.



## Avertissement

Il est dangereux d'utiliser la même variable-f à la fois comme entrée et comme sortie des opcodes cuda. Ceci peut produire un comportement indéfini de certains de ces opcodes. Utilisez une variable différente à gauche et à droite de l'opcode.

## Exemples

Voici un exemple de l'opcode `cudanal`. Il utilise le fichier `cudanal.csd` [examples/cudanal.csd].

### Exemple 1244. Exemple de l'opcode `cudanal`.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o cudanal.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1 ;cudanal has no influence when there is no transformation of original sound

ifftsize = p4
ioverlap = ifftsize / 4
iwinsize = ifftsize
iwinshape = 1 ;von-Hann window
Sfile = "fox.wav"
ain soundin Sfile
fftin cudanal ain, ifftsize, ioverlap, iwinsize, iwinshape ;fft-analysis of the audio-signal
fftblur pvscale fftin, p5 ;scale
aout cudasyntfftblur ;resynthesis
outs aout, aout

endin

</CsInstruments>
<CsScore>
s
i 1 0 3 512 1 ;original sound - ifftsize of pvsanal does not have any influence
i 1 3 3 1024 1 ;even with different
i 1 6 3 2048 1 ;settings

s
i 1 0 3 512 1.5 ;but transformation - here a fifth higher
i 1 3 3 1024 1.5 ;but with different settings
i 1 6 3 2048 1.5 ;for ifftsize of pvsanal

e
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
2013

Nouveau dans la version 6.02

# cudasynth

cudasynth — Synthèse par synthèse additive et TFR inverse. Expérimental et seulement disponible dans le code source pour le moment.

## Description

Synthèse par synthèse additive et TFR inverse.

## Syntaxe

```
asig cudasynth kamp, kfreq, itab, iftab, iatab[, inum]
asig cudasynth fsig, kamp, kfreq[, inum]
asig cudasynth fsig
```

## Initialisation

```
itab --
iftab --
iatab --
inum -- (facultatif)
```

## Exécution

```
kamp --
kfreq --
fsig --
```

## Exemples

Voici un exemple de l'opcode cudasynth. Il utilise le fichier *cudasynth.csd* [examples/cudasynth.csd].

### Exemple 1245. Exemple de l'opcode cudasynth.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o cudsyth.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>
```

```
sr = 44100
ksmps = 64
nchnls = 1
odbfs = 1

i1 ftgen 1,0,512,7,1,512,0.001
i2 ftgen 2,0,512,-7,1,512,512
i3 ftgen 3,0,16384,10,1
schedule 1,0,10

instr 1
a1 cudasynth 0.001, 100,0, 2, 1,128
  out a1

endin
</CsInstruments>
<CsScore>
e 10
</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Victor Lazzarini  
2013

Nouveau dans la version 6.02



# cudasliding

**cudasliding** — Exécute un algorithme de vocodeur de phase glissant avec transformation FM simplifiée en utilisant un GPU. Expérimental et seulement disponible dans le code source pour le moment.

## Description

Exécute un algorithme de vocodeur de phase glissant avec transformation FM simplifiée en utilisant un GPU.

## Syntaxe

```
asig cudasliding ain, amod, iwinsize
```

## Initialisation

*iwinsize* -- La taille de la TFR en échantillons. Ne doit pas forcément être une puissance de deux. *ifftsize* détermine le nombre de bins d'analyse utilisés.

## Exécution

*ain* -- signal d'entrée à transformer.

*amod* -- signal modulant le signal analysé.

## Exemples

Voici un exemple de l'opcode **cudasliding**. Il utilise le fichier *cudasliding.csd* [examples/cudasliding.csd].

### Exemple 1246. Exemple de l'opcode **cudasliding**.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o cudasliding.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 64
0dbfs = 1
nchnls = 2

instr 1
asig = diskin:a("flutec3.wav",1,0,1)
amod = oscil:a(1,3)
asig2 = cudasliding(asig,amod)
```

```
asig = linenr(asig2,0.005,0.01,0.01)
      out(asig)
      endin
```

```
</CsInstruments>
<CsScore>
i1 0 60

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Russell Bradford  
Mars 2014

Nouveau dans la version 6.03

---

# Opcodes de l'orchestre et routines GEN obsolètes

## Opcodes de l'orchestre obsolètes

Les opcodes suivants sont obsolètes. Ils sont encore distribués avec Csound pour des raisons de compatibilité ascendante. Les nouveaux opcodes à utiliser à leur place sont indiqués dans les entrées suivantes.

# abetarand

abetarand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *betarand*.

# abexprnd

abexprnd — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *bexprnd*.

# acauchy

acauchy — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *cauchy*.

# aexprand

aexprand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *exprand*.

# agauss

agauss — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *gauss*.



# agogobel

agogobel — Obsolète.

## Description

Nouveau dans la version 3.47

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *gogobel*.

# alinrand

alinrand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *linrand*.

# apcauchy

apcauchy — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *pcauchy*.

# apoisson

apoisson — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *poisson*.

# apow

apow — Obsolète.

## Description

Obsolète depuis la version 3.48. Utiliser plutôt l'opcode *pow*.

# array

array — Obsolète.

## Description

Crée un vecteur (tableau unidimensionnel de taux-k) avec des valeurs initiales.

## Syntaxe

```
karray[] array ival1, ival2, ..., ivaln
```

## Initialisation

*ival1*, ..., *ivaln* -- valeurs à placer dans le vecteur.

## Exemples

Voici un exemple de l'opcode array. Il utilise le fichier *array.csd* [examples/array.csd].

### Exemple 1247. Exemple de l'opcode array.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-n
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1

    kS[] = array(1,7,5)
    printk2 kS[0]
    printk2 kS[1]
    printk2 kS[2]

endin

</CsInstruments>
<CsScore>
i 1 0 0
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

Opcodes vectoriels fillarray

## Crédits

Auteur : John ffitch  
Codemist Ltd  
2013

Nouveau dans la version 6.00

# atrirand

atrirand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *trirand*.



# aunirand

aunirand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *unirand*.

# aweibull

aweibull — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *weibull*.

# bformdec

bformdec — Obsolète. Décode un signal au format ambisonic B.

## Description

Décode un signal au format ambisonic B en signaux de haut-parleur spécifiques. Noter que cet opcode est obsolète et imprécis et qu'il est remplacé par l'opcode *bformdec1* bien meilleur qui reprend toutes les caractéristiques importantes.

## Syntaxe

```
ao1, ao2 bformdec isetup, aw, ax, ay, az [, ar, as, at, au, av \
    [, abk, al, am, an, ao, ap, aq]]

ao1, ao2, ao3, ao4 bformdec isetup, aw, ax, ay, az [, ar, as, at, \
    au, av [, abk, al, am, an, ao, ap, aq]]

ao1, ao2, ao3, ao4, ao5 bformdec isetup, aw, ax, ay, az [, ar, as, \
    at, au, av [, abk, al, am, an, ao, ap, aq]]

ao1, ao2, ao3, ao4, ao5, ao6, ao7, ao8 bformdec isetup, aw, ax, ay, az \
    [, ar, as, at, au, av [, abk, al, am, an, ao, ap, aq]]]
```

## Initialisation

*isetup* — réglage de haut-parleur. Il y a cinq réglages possibles : 1 indique le réglage stéréo. Il doit y avoir deux cellules de sortie avec les positions de haut-parleur supposées valoir (330/0, 30/0).

2 indique le réglage quadraphonique. Il doit y avoir quatre cellules de sortie. Les positions de haut-parleur sont supposées valoir (45°/0), (135°/0), (225/0), (315/0).

3 est un réglage surround 5.1. Il doit y avoir cinq cellules de sortie. Le canal LFE n'est pas supporté. Les positions de haut-parleur sont supposées valoir (330/0), (30/0), (0/0), (250/0), (110/0).

4 indique huit haut-parleurs en cercle. Il doit y avoir huit cellules de sortie. Les positions de haut-parleur sont supposées valoir (22.5/0), (67.5/0), (112.5/0), (157.5/0), (202.5/0), (247.5/0), (292.5/0), (337.5/0).

5 indique un réglage cubique de huit haut-parleurs. Il doit y avoir huit cellules de sortie. Les positions de haut-parleur sont supposées valoir (45/0), (45/30), (135/0), (135/30), (225/0), (225/30), (315/0), (315/30).

## Exécution

*aw, ax, ay, ...* -- signal d'entrée au format B.

*ao1 .. ao8* — signaux de haut-parleur spécifiques en sortie.

## Exemples

Voici un exemple de l'opcode bformdec. Il utilise le fichier *bformenc.csd* [examples/bformenc.csd].

### Exemple 1248. Exemple de l'opcode bformdec.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
;-odac      -iadc      -d      ;;;RT audio I/O
; For Non-realtime output leave only the line below:
-o bformenc.wav -W ;;; for file output any platform
</CsOptions>
<CsInstruments>
sr = 44100
kr = 4410
ksmps = 10
nchnls = 8

;bformenc is deprecated, please use bformenc1

instr 1
; generate pink noise
anoise pinkish 1000

; two full turns
kalpha line 0, p3, 720
kbeta = 0

; fade ambisonic order from 2nd to 0th during second turn
kord0 = 1
kord1 linseg 1, p3 / 2, 1, p3 / 2, 0
kord2 linseg 1, p3 / 2, 1, p3 / 2, 0

; generate B format
aw, ax, ay, az, ar, as, at, au, av bformenc anoise, kalpha, kbeta, kord0, kord1, kord2

; decode B format for 8 channel circle loudspeaker setup
a1, a2, a3, a4, a5, a6, a7, a8 bformdec 4, aw, ax, ay, az, ar, as, at, au, av

; write audio out
outo a1, a2, a3, a4, a5, a6, a7, a8
endin

</CsInstruments>
<CsScore>

; Play Instrument #1 for 20 seconds.
i 1 0 20
e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Samuel Groner  
2005

Nouveau dans la version 5.07. Obsolète dans la 5.09.

# bformenc

bformenc — Obsolète. Encode un signal dans le format ambisonic B.

## Description

Encode un signal dans le format ambisonic B. Noter que cet opcode est obsolète et imprécis et qu'il est remplacé par l'opcode *bformenc1* bien meilleur qui reprend toutes les caractéristiques importantes ; noter que les arguments de gain ne sont pas disponibles dans *bformenc1*.

## Syntaxe

```
aw, ax, ay, az bformenc asig, kalpha, kbeta, kord0, kord1  
aw, ax, ay, az, ar, as, at, au, av bformenc asig, kalpha, kbeta, \  
    kord0, kord1 , kord2  
aw, ax, ay, az, ar, as, at, au, av, ak, al, am, an, ao, ap, aq bformenc \  
    asig, kalpha, kbeta, kord0, kord1, kord2, kord3
```

## Exécution

*aw, ax, ay, ...* -- cellules de sortie au format B.

*asig* -- signal d'entrée.

*kalpha* — angle d'azimut en degrés (dans le sens des aiguilles d'une montre).

*kbeta* -- angle d'altitude en degrés.

*kord0* -- gain linéaire du format B d'ordre zéro.

*kord1* -- gain linéaire du format B du premier ordre.

*kord2* -- gain linéaire du format B du deuxième ordre.

*kord3* -- gain linéaire du format B du troisième ordre.

## Exemples

Voici un exemple de l'opcode bformenc. Il utilise le fichier *bformenc.csd* [examples/bformenc.csd].

### Exemple 1249. Exemple de l'opcode bformenc.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>  
<CsOptions>  
; Select audio/midi flags here according to platform  
; Audio out   Audio in   No messages  
;-odac       -iadc       -d      ;;;RT audio I/O  
; For Non-realtime ouput leave only the line below:  
-o bformenc.wav -W ;;; for file output any platform
```

```
</CsOptions>
<CsInstruments>
  sr = 44100
  kr = 4410
  ksmps = 10
  nchnls = 8

  ;bformenc is deprecated, please use bformenc1

  instr 1
    ; generate pink noise
    anoise pinkish 1000

    ; two full turns
    kalpha line 0, p3, 720
    kbeta = 0

    ; fade ambisonic order from 2nd to 0th during second turn
    kord0 = 1
    kord1 linseg 1, p3 / 2, 1, p3 / 2, 0
    kord2 linseg 1, p3 / 2, 1, p3 / 2, 0

    ; generate B format
    aw, ax, ay, az, ar, as, at, au, av bformenc anoise, kalpha, kbeta, kord0, kord1, kord2

    ; decode B format for 8 channel circle loudspeaker setup
    a1, a2, a3, a4, a5, a6, a7, a8 bformdec 4, aw, ax, ay, az, ar, as, at, au, av

    ; write audio out
    outo a1, a2, a3, a4, a5, a6, a7, a8
  endin

</CsInstruments>
<CsScore>

  ; Play Instrument #1 for 20 seconds.
  i 1 0 20
  e

</CsScore>
</CsoundSynthesizer>
```

## Crédits

Auteur : Samuel Groner  
2005

Nouveau dans la version 5.07. Obsolète dans la 5.09.

# clock

clock — Obsolète.

## Description

Obsolète. Utiliser plutôt l'opcode *rtclock*.

# hrtfer

hrtfer — Crée de l'audio 3D pour deux haut-parleurs.

## Description

La sortie audio en 3D est binaurale (casque stéréo).



### Note

Cet opcode est obsolète ; utiliser plutôt *hrtfstat* ou un opcode similaire.

## Syntaxe

```
aleft, aright hrtfer asig, kaz, kelev, « HRTFcompact »
```

## Initialisation

*kAz* -- valeur d'azimut en degrés. Les valeurs positives représentent les positions à droite, les valeurs négatives les positions à gauche.

*kElev* -- valeur d'élévation en degrés. Les valeurs positives représentent les positions au-dessus de l'horizontale, les valeurs négatives les positions sous l'horizontale.

Actuellement, le seul fichier qui peut être utilisé avec *hrtfer* est *HRTFcompact* [exemples/HRTFcompact]. Il doit être passé à l'opcode en dernier argument entre guillemets comme ci-dessus.

On peut aussi obtenir HRTFcompact par ftp anonyme depuis : <ftp://ftp.cs.bath.ac.uk/pub/dream/utilities/Analysis/HRTFcompact>

## Exécution

Ces générateurs unitaires placent un signal d'entrée mono dans un espace 3D virtuel autour de l'auditeur en faisant une convolution entre l'entrée et les données HRTF appropriées spécifiées par les valeurs d'azimut et d'élévation de l'opcode. *hrtfer* accepte que ces valeurs soient de taux-k, ce qui permet une spatialisation dynamique. *hrtfer* ne peut placer l'entrée qu'à la position demandée car le HRTF est chargé à l'initialisation (souvenez-vous qu'actuellement Csound limite à 20 le nombre de fichiers qu'il peut garder en mémoire sans causer d'erreur de segmentation). Il faut ajuster la sortie soit en utilisant *balance* soit en la multipliant par une constante de mise à l'échelle.



### Note

Le taux d'échantillonnage de l'orchestre doit être de 44.1 kHz. C'est le taux auquel les HRTFs ont été mesurés. Si l'on veut utiliser les HRTFs à un taux différent, il faut les rééchantillonner au taux désiré.

## Exemples

Voici un exemple de l'opcode *hrtfer*. Il utilise les fichiers *hrtfer.csd* [exemples/hrtfer.csd], *HRTFcompact* [exemples/HRTFcompact] et *beats.wav* [exemples/beats.wav].



## Exemple 1250. Exemple de l'opcode hrtfer.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
; Audio out  Audio in  No messages
-odac      -iadc      -d      ;;RT audio I/O
; For Non-realtime ouput leave only the line below:
; -o hrtfer.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

; Initialize the global variables.
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

instr 1
kaz      linseg 0, p3, -360 ; move the sound in circle
kel      linseg -40, p3, 45 ; around the listener, changing
                                ; elevation as its turning

asrc      soundin "beats.wav"
aleft,right hrtfer asrc, kaz, kel, "HRTFcompact"
aleftscale = aleft * 200
arightscale = aright * 200

outs      aleftscale, arightscale
endin

</CsInstruments>
<CsScore>

i 1 0 2
e

</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*hrtfmove, hrtfmove2, hrtfstat.*

## Crédits

Auteurs : Eli Breder et David MacIntyre  
Montréal  
1996

Correction de l'exemple grâce à un message d'Istvan Varga.

# ibetarand

ibetarand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *betarand*.

# ibexprnd

ibexprnd — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *bexprnd*.

# icauchy

icauchy — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *cauchy*.

# ictrl14

ictrl14 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *ctrl14*.

# ictrl21

ictrl21 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *ctrl21*.

# ictrl7

ictrl7 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *ctrl7*.

# iexprand

iexprand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *exprand*.



# igauss

igauss — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *gauss*.

# ilinrand

ilinrand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *linrand*.

# imidic14

imidic14 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *midic14*.

# imidic21

imidic21 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *midic21*.

# imidic7

imidic7 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *midic7*.

# instimek

instimek — Obsolète.

## Description

Obsolète depuis la version 3.62. Utiliser plutôt l'opcode *timeinstk*.

## Crédits

David M. Boothe est à l'origine du signalement de ce nom obsolète.

# instimes

instimes — Obsolète.

## Description

Obsolète depuis la version 3.62. Utiliser plutôt l'opcode *timeinsts*.

## Crédits

David M. Boothe est à l'origine du signalement de ce nom obsolète.

# **ioff**

ioff — Obsolète.

## **Description**

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *noteoff*.



# ion

ion — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *noteon*.

# iondur2

iondur2 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *noteondur2*.

# iondur

iondur — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *noteondur*.

# ioutat

ioutat — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outiat*.

# ioutc14

ioutc14 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outic14*.

# ioutc

ioutc — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outic*.

# ioutpat

ioutpat — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outipat*.

# ioutpb

ioutpb — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outipb*.



# ioutpc

ioutpc — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outipc*.

# ipcauchy

ipcauchy — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *pcauchy*.

# ipoisson

ipoisson — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *poisson*.

# ipow

ipow — Obsolète.

## Description

Obsolète depuis la version 3.48. Utiliser plutôt l'opcode *pow*.

# is16b14

is16b14 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *s16b14*.

# is32b14

is32b14 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *s32b14*.

# islider16

islider16 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *slider16*.

# islider32

islider32 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *slider32*.



# islider64

islider64 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *slider64*.

# islider8

islider8 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *slider8*.

# itablecopy

itablecopy — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *tableicopy*.

# itablegpw

itablegpw — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *tableigpw*.

# itablemix

itablemix — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *tableimix*.

# itablew

itablew — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *tableiw*.

# itrirand

itrirand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *trirand*.

# iunirand

iunirand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *unirand*.



# iweibull

iweibull — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *weibull*.

# kbetarand

kbetarand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *betarand*.

# kbexprnd

kbexprnd — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *bexprnd*.

# kcauchy

kcauchy — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *cauchy*.

# kdump2

kdump2 — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *dumpk2*.

# kdump3

kdump3 — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *dumpk3*.

# kdump4

kdump4 — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *dumpk4*.

# kdump

kdump — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *dumpk*.



# kexprand

kexprand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *exprand*.

# kfilter2

kfilter2 — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *filter2*.

## Crédits

Auteur : Michael A. Casey  
M.I.T.  
Cambridge, Mass.  
1997

Nouveau dans la version 3.47

# kgauss

kgauss — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *gauss*.

# klinrand

klinrand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *linrand*.

# kon

kon — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *midion*.

# koutat

koutat — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outkat*.

# koutc14

koutc14 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outkc14*.

# koutc

koutc — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outkc*.



# koutpat

koutpat — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outkpat*.

# koutpb

koutpb — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outkpb*.

# koutpc

koutpc — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *outkpc*.

# kpcauchy

kpcauchy — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *pcauchy*.

# kpoisson

kpoisson — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *poisson*.

# kpow

kpow — Obsolète.

## Description

Obsolète depuis la version 3.48. Utiliser plutôt l'opcode *pow*.

# kread2

kread2 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *readk2*.

# kread3

kread3 — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *readk3*.



# kread4

kread4 — Obsolète.

## Description

Obsolète depuis la version 3.52. Use the *readk4* opcode instead.

# kread

kread — Obsolète.

## Description

Obsolète depuis la version 3.52. Utiliser plutôt l'opcode *readk*.

# ktableseg

ktableseg — Obsolète.

## Description

Obsolète. Utiliser plutôt l'opcode *tableseg*.

## Syntaxe

```
ktableseg ifn1, idur1, ifn2 [, idur2] [, ifn3] [...]
```

# ktrirand

ktrirand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *trirand*.

# kunirand

kunirand — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *unirand*.

# kweibull

kweibull — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *weibull*.

# sndload

sndload — Charge un fichier son en mémoire pour être utilisé par *loscilx*

## Description

*sndload* charge un fichier son en mémoire pour être utilisé par *loscilx*.



### Note

Cet opcode charge le fichier en mémoire mais il n'est plus disponible pour un usage ultérieure. C'est juste une perte de mémoire.

## Syntaxe

```
sndload Sfname[, ifmt[, ichns[, isr[, ibas[, iamp[, istrtr \
[, ilpmod[, ilps[, ilpe]]]]]]]]]
```

## Initialisation

*Sfname* - nom du fichier sous la forme d'une constante, d'une variable ou d'un p-champ chaîne de caractères, ou bien un nombre utilisé comme index dans un ensemble de chaînes de caractères avec *strset* ou, s'il n'y a pas de chaîne disponible, pour générer un nom de fichier au format *soundin.n*. Si le nom de fichier ne comprend pas un chemin complet, le fichier est d'abord cherché dans le répertoire courant, puis dans celui qui est spécifié par *SSDIR* (si défini), et finalement par *SFDIR*. Si le même fichier a déjà été chargé antérieurement, il n'est pas relu, mais les paramètres *ibas*, *iamp*, *istrtr*, *ilpmod*, *ilps* et *ilpe* sont quand même mis à jour.

*ifmt* (facultatif, zéro par défaut) - format d'échantillon par défaut pour les fichiers son bruts (sans en-tête) ; si le fichier a un en-tête, cet argument est ignoré. Les valeurs possibles sont :

- 1 : interdit les fichiers sans en-tête (échec avec une erreur d'initialisation)
- 0 : utilise le format spécifié dans la ligne de commande
- 1 : entiers signés sur 8 bit
- 2 : a-law
- 3 : u-law
- 4 : entiers signés sur 16 bit
- 5 : entiers signés sur 32 bit
- 6 : flottants sur 32 bit
- 7 : entiers non signés sur 8 bit
- 8 : entiers signés sur 24 bit
- 9 : flottants sur 64 bit

*ichns* (facultatif, zéro par défaut) - nombre de canaux par défaut pour les fichiers son bruts (sans en-tête) ; si le fichier a un en-tête, cet argument est ignoré. Les valeurs nulle ou négatives sont interprétées comme 1 canal.

*isr* (facultatif, zéro par défaut) - taux d'échantillonnage par défaut pour les fichiers son bruts (sans en-tête) ; si le fichier a un en-tête, cet argument est ignoré. Les valeurs nulle ou négatives sont interprétées comme le taux d'échantillonnage de l'orchestre (*sr*).

*ibas* (facultatif, zéro par défaut) - fréquence de base en Hz. Si elle est positive, elle remplace la valeur spécifiée dans l'en-tête du fichier son ; sinon, la valeur de l'en-tête est utilisée si elle est présente, et 1.0 si le fichier ne contient pas cette information.

*iamp* (facultatif, zéro par défaut) - pondération de l'amplitude. Si elle est différente de zéro, elle remplace la valeur spécifiée dans l'en-tête du fichier son (note : les valeurs négatives sont permises, elles inversent la phase de la sortie) ; sinon, la valeur de l'en-tête est utilisée si elle est présente, et 1.0 si le fichier ne contient pas cette information.

*istrt* (facultatif, -1 par défaut) - position du début en trames d'échantillon, peut être fractionnaire. Si elle est non négative, elle remplace la valeur spécifiée dans l'en-tête du fichier son ; sinon, la valeur de l'en-tête est utilisée si elle est présente, et 0 si le fichier ne contient pas cette information. Note : même si cet argument est spécifié, le fichier entier est lu en mémoire.

*ilpmo*d (facultatif, -1 par défaut) - mode de boucle, l'un des suivants :

n'importe quelle valeur négative : utilise l'information de boucle spécifiée dans l'en-tête du fichier son, ignorant *ilps* et *ilpe*

0 : pas de boucle (*ilps* et *ilpe* sont ignorés)

1 : boucle à l'endroit (cycle autour de la fin de boucle si elle est traversée en avançant, et cycle autour du début du boucle s'il est traversé en reculant)

2 : boucle à l'envers (change de direction à la fin de boucle si elle est traversée en avançant, et cycle autour du début de boucle s'il est traversé en reculant)

3 : boucle à l'endroit et à l'envers (change de direction aux deux points de boucle s'ils sont traversés comme décrit ci-dessus)

*ilps* (facultatif, zéro par défaut) - début de boucle en trames d'échantillon (valeurs fractionnaires autorisées), ou fin de boucle si *ilps* est supérieur à *ilpe*. Ignoré sauf si *ilpmo*d vaut 1, 2 ou 3. Si les points de boucle sont égaux, la boucle se fait sur l'échantillon complet.

*ilpe* (facultatif, zéro par défaut) - fin de boucle en trames d'échantillon (valeurs fractionnaires autorisées), ou début de boucle si *ilps* est supérieur à *ilpe*. Ignoré sauf si *ilpmo*d vaut 1, 2 ou 3. Si les points de boucle sont égaux, la boucle se fait sur l'échantillon complet.

## Crédits

Écrit par Istvan Varga.

2006

Nouveau dans Csound 5.03



# peakk

peakk — Obsolète.

## Description

Obsolète depuis la version 3.63. Utiliser plutôt l'opcode *peak*.

# pop

push — Extrait des valeurs de la pile globale. Obsolète.

## Description

Opcodes du greffon stackops.

Extrait des valeurs de la pile globale.

## Syntaxe

```
xval1, [xval2, ... , xval31] pop
```

```
ival1, [ival2, ... , ival31] pop
```

## Initialisation

*ival1 ... ival31* -- valeurs à extraire de la pile.

## Exécution

*xval1 ... xval31* -- valeurs à extraire de la pile.

Les valeurs données sont extraites de la pile. La pile globale fonctionne en mode dernier entré, premier sorti : après de multiples appels à *push*, il faut utiliser *pop* dans l'ordre inverse.

Chaque opération *push* ou *pop* peut traiter un "paquet" de variables. Lorsque l'on utilise *pop*, le nombre, le type et l'ordre des éléments doivent être les mêmes que ceux utilisés par le *push* correspondant. Ainsi après un "push Sfoo, ibar", il faut un appel comme "Sbar, ifoo pop", et pas, par exemple deux instructions "pop" séparées.

Les opcodes *push* et *pop* acceptent des variables de n'importe quel type (taux-i, -k, -a et chaînes de caractères). On peut utiliser n'importe quelle combinaison de types-i, -k, -a ou -S. Les variables de type 'a' et 'k' ne sont passées que pendant l'exécution, tandis que celles de type 'i' et 'S' ne sont passées que pendant l'initialisation.

*push/pop* pour les types a, k, i et S copient les données par valeur. Au contraire, *push\_f* ne pousse qu'une référence du f-signal et le *pop\_f* correspondant copiera directement depuis la variable originale dans le signal de sortie. Pour cette raison, il n'est pas recommandé de changer le f-signal source de *push\_f* avant l'appel à *pop\_f*. De même, si l'instance d'instrument possédant la variable passée à *push\_f* est désactivée avant que *pop\_f* ne soit appelé, il peut en résulter un comportement indéfini.

Toutes les erreurs de pile (tentative de pousser des données alors qu'il n'y a plus d'espace ou d'extraire des données d'une pile vide, nombre ou types d'arguments inconsistants, etc) sont fatales et terminent l'exécution.

## Exemples

Voici un exemple de l'opcode pop. Il utilise le fichier *pop.csd* [examples/pop.csd].

## Exemple 1251. Exemple de l'opcode pop.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o pop.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

stack 100000

instr 1

a1 oscils 0.7, 220, 0
k1 line 0, p3, 1
    push "blah", 123.45, a1, k1
    push rnd(k1)

k_rnd pop
S01, i01, a01, k01 pop
    printf_i "S01 = '%s', i01 = %g\n", 1, S01, i01
ktrig metro 5.0
    printf "k01 = %.3f, k_rnd = %.3f\n", ktrig, k01, k_rnd
    outs a01, a01

endin
</CsInstruments>
<CsScore>

i 1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*stack*, *push*, *pop\_f* and *push\_f*.

L'utilisation de cet opcode relève un peu du bricolage comme on peut le lire ici : <http://csound.1045644.n5.nabble.com/passing-a-string-to-a-UDO-td1099284.html>.

## Crédits

Par Istvan Varga.

2006

Obsolète depuis la version 6.04.

# pop\_f

pop\_f — Extrait une trame de f-sig de la pile globale. Obsolète.

## Description

Opcodes du greffon stackops.

Extrait une trame de f-sig de la pile globale.

## Syntaxe

`f sig pop_f`

## Exécution

*f sig* -- f-signal à extraire de la pile.

Les valeurs sont extraites de la pile. La pile globale doit être initialisée avant utilisation et sa taille doit être fixée. La pile globale fonctionne en mode dernier entré, premier sorti : après plusieurs appels *push\_f*, il faut utiliser *pop\_f* dans l'ordre inverse.

*push/pop* pour les types a, k, i et S copient les données par valeur. Au contraire, *push\_f* ne pousse qu'une référence du f-signal et le *pop\_f* correspondant copiera directement depuis la variable originale dans le signal de sortie. Pour cette raison, il n'est pas recommandé de changer le f-signal source de *push\_f* avant l'appel à *pop\_f*. De même, si l'instance d'instrument possédant la variable passée à *push\_f* est désactivée avant que *pop\_f* ne soit appelé, il peut en résulter un comportement indéfini.

*push\_f* et *pop\_f* ne peuvent prendre qu'un seul argument et les données sont passées à la fois à l'initialisation et pendant l'exécution.

Toutes les erreurs de pile (tentative de pousser des données alors qu'il n'y a plus d'espace ou d'extraire des données d'une pile vide, nombre ou types d'arguments inconsistants, etc) sont fatales et terminent l'exécution.

## Voir aussi

*stack*, *push*, *pop* and *push\_f*.

## Crédits

Par Istvan Varga.

2006

Obsolète depuis la version 6.04.

# push

`push` — Pousse une valeur dans la pile globale. Obsolète.

## Description

Opcodes du greffon `stackops`.

Pousse une valeur dans la pile globale.

## Syntaxe

```
push  xval1, [xval2, ... , xval31]
```

```
push  ival1, [ival2, ... , ival31]
```

## Initialisation

*ival1 ... ival31* -- valeurs à pousser sur la pile.

## Exécution

*xval1 ... xval31* -- valeurs à pousser sur la pile.

Les valeurs données sont poussées dans la pile globale sous la forme d'un paquet. La pile globale fonctionne en mode dernier entré, premier sorti : après de multiples appels à *push*, il faut utiliser *pop* dans l'ordre inverse.

Chaque opération *push* ou *pop* peut traiter un "paquet" de variables. Lorsque l'on utilise *pop*, le nombre, le type et l'ordre des éléments doivent être les mêmes que ceux utilisés par le *push* correspondant. Ainsi après un "push Sfoo, ibar", il faut un appel comme "Sbar, ifoo pop", et pas, par exemple deux instructions "pop" séparées.

Les opcodes *push* et *pop* acceptent des variables de n'importe quel type (taux-i, -k, -a et chaînes de caractères). On peut utiliser n'importe quelle combinaison de types-i, -k, -a ou -S. Les variables de type 'a' et 'k' ne sont passées que pendant l'exécution, tandis que celles de type 'i' et 'S' ne sont passées que pendant l'initialisation.

*push/pop* pour les types a, k, i et S copient les données par valeur. Au contraire, *push\_f* ne pousse qu'une référence du f-signal et le *pop\_f* correspondant copiera directement depuis la variable originale dans le signal de sortie. Pour cette raison, il n'est pas recommandé de changer le f-signal source de *push\_f* avant l'appel à *pop\_f*. De même, si l'instance d'instrument possédant la variable passée à *push\_f* est désactivée avant que *pop\_f* ne soit appelé, il peut en résulter un comportement indéfini.

Toutes les erreurs de pile (tentative de pousser des données alors qu'il n'y a plus d'espace ou d'extraire des données d'une pile vide, nombre ou types d'arguments inconsistants, etc) sont fatales et terminent l'exécution.

## Exemples

Voici un exemple de l'opcode `push`. Il utilise le fichier *push.csd* [examples/push.csd].

## Exemple 1252. Exemple de l'opcode push.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o push.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

stack 100000

instr 1

a1 oscils 0.7, 220, 0
k1 line 0, p3, 1
    push "blah", 123.45, a1, k1
    push rnd(k1)

k_rnd pop
S01, i01, a01, k01 pop
    printf_i "S01 = '%s', i01 = %g\n", 1, S01, i01
ktrig metro 5.0
    printf "k01 = %.3f, k_rnd = %.3f\n", ktrig, k01, k_rnd
    outs a01, a01

endin
</CsInstruments>
<CsScore>

i 1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*stack*, *pop*, *pop\_f* and *push\_f*.

L'utilisation de cet opcode relève un peu du bricolage comme on peut le lire ici : <http://csound.1045644.n5.nabble.com/passing-a-string-to-a-UDO-td1099284.html>.

## Crédits

Par Istvan Varga.

2006

Obsolète depuis la version 6.04.

# push\_f

push\_f — Pousse une trame de f-sig dans la pile globale. Obsolète.

## Description

Opcodes du greffon stackops.

Pousse une trame de f-sig dans la pile globale.

## Syntaxe

```
push_f fsig
```

## Exécution

*fsig* -- f-signal à pousser sur la pile.

Les valeurs sont poussées dans la pile globale. La pile globale fonctionne en mode dernier entré, premier sorti : après de multiples appels à *push\_f*, il faut utiliser *pop\_f* dans l'ordre inverse.

*push/pop* pour les types a, k, i et S copient les données par valeur. Au contraire, *push\_f* ne pousse qu'une référence du f-signal et le *pop\_f* correspondant copiera directement depuis la variable originale dans le signal de sortie. Pour cette raison, il n'est pas recommandé de changer le f-signal source de *push\_f* avant l'appel à *pop\_f*. De même, si l'instance d'instrument possédant la variable passée à *push\_f* est désactivée avant que *pop\_f* ne soit appelé, il peut en résulter un comportement indéfini.

*pop\_f* et *push\_f* ne peuvent prendre qu'un seul argument et les données sont passées à la fois à l'initialisation et pendant l'exécution.

Toutes les erreurs de pile (tentative de pousser des données alors qu'il n'y a plus d'espace ou d'extraire des données d'une pile vide, nombre ou types d'arguments inconsistants, etc) sont fatales et terminent l'exécution.

## Voir aussi

*stack*, *push*, *pop* and *pop\_f*.

## Crédits

Par Istvan Varga.

2006

Obsolète depuis la version 6.04.

# soundout

soundout — Obsolète. Écrit la sortie audio dans un fichier sur disque.

## Description



### Note

L'utilisation de *soundout* est déconseillée. Il vaut mieux utiliser *fout*.

Écrit la sortie audio dans un fichier sur disque.

## Syntaxe

```
soundout  asig1, ifilcod [, iformat]
```

## Initialisation

*ifilcod* -- entier ou chaîne de caractères donnant le nom du fichier son destination. Un entier indique le fichier soundin.filcod ; une chaîne de caractères (entre guillemets, espaces autorisés) donne le nom de fichier lui-même, éventuellement un nom de chemin complet. Si ce n'est pas un nom de chemin complet, le fichier nommé est d'abord cherché dans le répertoire courant, puis dans celui qui est donné par la variable d'environnement *SSDIR* (si elle est définie) puis par *SFDIR*. Voir aussi *GEN01*.

*iformat* (facultatif, 0 par défaut) -- spécifie le format des données audio du fichier :

- 1 = caractères signés sur 8 bit (les 8 bit de poids fort d'un entier sur 16 bit)
- 2 = octets sur 8 bit A-law
- 3 = octets sur 8 bit U-law
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit

Si *iformat* = 0, il est déduit de l'en-tête du fichier, et s'il n'y a pas d'en-tête, de l'option de ligne de commande *-o* de Csound. La valeur par défaut est 0.

## Exécution

*soundout* écrit la sortie audio dans un fichier sur disque.



### Note

Il est recommandé d'utiliser *fout* plutôt que *soundout*

## Voir aussi

*fout*, *out*, *outh*, *outo*, *outq*, *outq1*, *outq2*, *outq3*, *outq4*, *outs*, *outs1*, *outs2* *soundouts*



## Crédits

Auteurs : Barry L. Vercoe, Matt Ingalls/Mike Berry  
MIT, Mills College  
1993-1997

# soundouts

soundouts — Obsolète. Ecrit la sortie audio dans un fichier sur disque.

## Description



### Note

L'utilisation de *soundouts* est déconseillée. Il vaut mieux utiliser *fout*.

Ecrit la sortie audio dans un fichier sur disque.

## Syntaxe

```
soundouts asigl, asigr, ifilcod [, iformat]
```

## Initialisation

*ifilcod* -- entier ou chaîne de caractères donnant le nom du fichier son destination. Un entier indique le fichier soundin.filcod ; une chaîne de caractères (entre guillemets, espaces autorisés) donne le nom de fichier lui-même, éventuellement un nom de chemin complet. Si ce n'est pas un nom de chemin complet, le fichier nommé est d'abord cherché dans le répertoire courant, puis dans celui qui est donné par la variable d'environnement SSDIR (si elle est définie) puis par SFDIR. Voir aussi *GEN01*.

*iformat* (facultatif, 0 par défaut) -- spécifie le format des données audio du fichier :

- 1 = caractères signés sur 8 bit (les 8 bit de poids fort d'un entier sur 16 bit)
- 4 = entiers courts sur 16 bit
- 5 = entiers longs sur 32 bit
- 6 = flottants sur 32 bit

Si *iformat* = 0, il est déduit de l'option de ligne de commande *-o* de Csound. La valeur par défaut est 0.

## Exécution

*soundouts* écrit la sortie audio stéréo dans un fichier sur disque au format brut (sans en-tête) et sans mise à l'échelle 0dbFS. L'intervalle d'amplitude attendu des signaux audio dépend du format d'échantillon choisi.



### Note

Il est recommandé d'utiliser *fout* plutôt que *soundouts*

## Voir aussi

*out, outh, outo, outq, outq1, outq2, outq3, outq4, outs, outs1, outs2 soundout*

## Crédits

Auteur : Istvan Varga

# stack

stack — Initialise la pile. Obsolète.

## Description

Opcodes du greffon stackops.

Initialise et fixe la taille de la pile globale.

## Syntaxe

```
stack iStackSize
```

## Initialisation

*iStackSize* -- taille de la pile en octets.

## Exécution

Csound implémente une pile globale unique. L'initialisation de la pile par l'opcode *stack* n'est pas requise - elle est facultative, et si elle n'a pas eu lieu, la première utilisation de *push* ou de *push\_f* créera automatiquement une pile de 32768 octets. Sinon, *stack* est normalement appelé depuis l'en-tête de l'orchestre et prend un paramètre de taille en octets (il y a une limite supérieure d'environ 16 MO). Une fois fixée, la taille de la pile reste constante et ne peut pas être modifiée durant l'exécution.

La pile globale fonctionne en mode dernier entré, premier sorti : après plusieurs appels *push\_f*, il faut utiliser *pop\_f* dans l'ordre inverse.

Chaque opération *push* ou *pop* peut traiter un "paquet" de variables. Lorsque l'on utilise *pop*, le nombre, le type et l'ordre des éléments doivent être les mêmes que ceux utilisés par le *push* correspondant. Ainsi après un "push Sfoo, ibar", il faut un appel comme "Sbar, ifoo pop", et pas, par exemple deux instructions "pop" séparées.

Les opcodes *push* et *pop* acceptent des variables de n'importe quel type (taux-i, -k, -a et chaînes de caractères). Les variables de type 'a' et 'k' ne sont passées que pendant l'exécution, tandis que celles de type 'i' et 'S' ne sont passées que pendant l'initialisation.

*push/pop* pour les types a, k, i et S copient les données par valeur. Au contraire, *push\_f* ne pousse qu'une référence du f-signal et le *pop\_f* correspondant copiera directement depuis la variable originale dans le signal de sortie. Pour cette raison, il n'est pas recommandé de changer le f-signal source de *push\_f* avant l'appel à *pop\_f*. De même, si l'instance d'instrument possédant la variable passée à *push\_f* est désactivée avant que *pop\_f* ne soit appelé, il peut en résulter un comportement indéfini.

Toutes les erreurs de pile (tentative de pousser des données alors qu'il n'y a plus d'espace ou d'extraire des données d'une pile vide, nombre ou types d'arguments inconsistants, etc) sont fatales et terminent l'exécution.

## Exemples

Voici un exemple de l'opcode stack. Il utilise le fichier *stack.csd* [examples/stack.csd].

## Exemple 1253. Exemple de l'opcode stack.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac    ;;realtime audio out
;-iadc    ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o stack.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
0dbfs = 1
nchnls = 2

stack 100000

instr 1

a1 oscils 0.7, 220, 0
k1 line 0, p3, 1
    push "blah", 123.45, a1, k1
    push rnd(k1)

k_rnd pop
S01, i01, a01, k01 pop
    printf_i "S01 = '%s', i01 = %g\n", 1, S01, i01
ktrig metro 5.0
    printf "k01 = %.3f, k_rnd = %.3f\n", ktrig, k01, k_rnd
    outs a01, a01

endin
</CsInstruments>
<CsScore>

i 1 0 5
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*pop*, *push*, *pop\_f* and *push\_f*.

L'utilisation de cet opcode relève un peu du bricolage comme on peut le lire ici : <http://csound.1045644.n5.nabble.com/passing-a-string-to-a-UDO-td1099284.html>.

## Crédits

Par Istvan Varga.

2006

Obsolète depuis la version 6.04.

## tb

tb0, tb1, tb2, tb3, tb4, tb5, tb6, tb7, tb8, tb9, tb10, tb11, tb12, tb13, tb14, tb15, tb0\_init, tb1\_init, tb2\_init, tb3\_init, tb4\_init, tb5\_init, tb6\_init, tb7\_init, tb8\_init, tb9\_init, tb10\_init, tb11\_init, tb12\_init, tb13\_init, tb14\_init, tb15\_init — Accès en lecture à une table depuis une expression.

## Description



### Note

Ces opcodes sont généralement remplacés par le forme fonctionnelle de l'opcode tab.

Permet de lire des tables de manière fonctionnelle, à utiliser dans des expressions. Anciennement, Csound ne supportait que les fonctions avec un seul argument en entrée. Cependant, pour accéder aux éléments d'une table, on doit fournir deux nombres : le numéro de la table et l'indice de l'élément. Donc, afin de pouvoir accéder à un élément d'une table par une fonction, il faut une étape de préparation.

## Syntaxe

```
tb0_init ifn
tb1_init ifn
tb2_init ifn
tb3_init ifn
tb4_init ifn
tb5_init ifn
tb6_init ifn
tb7_init ifn
tb8_init ifn
tb9_init ifn
tb10_init ifn
tb11_init ifn
tb12_init ifn
tb13_init ifn
tb14_init ifn
tb15_init ifn

iout = tb0(iIndex)
kout = tb0(kIndex)
iout = tb1(iIndex)
kout = tb1(kIndex)
iout = tb2(iIndex)
kout = tb2(kIndex)
```

```
iout = tb3(iIndex)
kout = tb3(kIndex)
iout = tb4(iIndex)
kout = tb4(kIndex)
iout = tb5(iIndex)
kout = tb5(kIndex)
iout = tb6(iIndex)
kout = tb6(kIndex)
iout = tb7(iIndex)
kout = tb7(kIndex)
iout = tb8(iIndex)
kout = tb8(kIndex)
iout = tb9(iIndex)
kout = tb9(kIndex)
iout = tb10(iIndex)
kout = tb10(kIndex)
iout = tb11(iIndex)
kout = tb11(kIndex)
iout = tb12(iIndex)
kout = tb12(kIndex)
iout = tb13(iIndex)
kout = tb13(kIndex)
iout = tb14(iIndex)
kout = tb14(kIndex)
iout = tb15(iIndex)
kout = tb15(kIndex)
```

## Exécution

Il y a 16 opcodes différents dont le nom est associé à un nombre compris entre 0 et 15. Il faut associer une table spécifique avec chaque opcode (ainsi le nombre maximum de tables accessibles de manière fonctionnelle est 16). Avant de pouvoir accéder à une table, celle-ci doit être associée avec l'un des 16 opcodes au moyen d'un opcode choisi parmi *tb0\_init*, ..., *tb15\_init*. Par exemple,

```
tb0_init 1
```

associe la table 1 avec la fonction *tb0*( ), si bien que chaque élément de la table 1 peut être atteint (de manière fonctionnelle) par :

```
kvar = tb0(k_some_index_of_table1) * k_some_other_var
```

```
ivar = tb0(i_some_index_of_table1) + i_some_other_var
```

etc...

En utilisant ces opcodes, on peut réduire considérablement le nombre de lignes d'un orchestre, ce qui améliore sa lisibilité.

## Crédits

Ecrit par Gabriel Maldonado.

# tableiw

tableiw — Obsolète.

## Description

Obsolète depuis la version 3.49. Utiliser plutôt l'opcode *tablew*.

## Syntaxe

```
tableiw isig, indx, ifn [, ixmode] [, ixoff] [, iwgmode]
```

## Initialisation

*isig* -- Valeur d'entrée à écrire dans la table.

*indx* -- Indice dans la table, un nombre positif compris entre 0 et la longueur de la table (*ixmode* = 0) ou entre 0 et 1 (*ixmode* différent de 0).

*ifn* -- Numéro de la table. Doit être  $\geq 1$ . Les nombres flottants sont arrondis à l'entier inférieur. Si un numéro de table ne pointe pas vers une table valide, ou si la table n'a pas encore été chargée (*GEN01*) une erreur est générée et l'instrument est désactivé.

*ixmode* (facultatif, 0 par défaut) -- mode d'indexation.

- 0 = *indx* et *ixoff* sont compris entre 0 et la longueur de la table.
- différent de 0 = *indx* et *ixoff* sont compris entre 0 et 1.

*ixoff* (facultatif, 0 par défaut) -- décalage de l'index.

- 0 = l'indice résultant est contrôlé directement par *indx*, l'indexation commençant depuis le début de la table.
- Différent de 0 = l'indexation démarre dans la table. La valeur doit être positive et inférieure à la longueur de la table (*ixmode* = 0) ou inférieure à 1 (*ixmode* différent de 0).

*iwgmode* (facultatif, 0 par défaut) -- mode cyclique et point de garde.

- 0 = mode limite.
- 1 = mode cyclique.
- 2 = mode point de garde.

## Exécution

### Mode limite (0)

Limite l'indice résultant (*indx* + *ixoff*) entre 0 et le point de garde. Pour une table de longueur 5, cela signifie que les positions allant de 0 à 3 et la position 4 (le point de garde) peuvent être écrites. Un indice résultant négatif provoque l'écriture en position 0.



## Mode cyclique (1)

Parcours cyclique de l'indice résultant dans les positions 0 à E, où E vaut soit la longueur de la table moins un, soit le facteur de 2 qui est égal à la longueur de la table moins un. Par exemple, un parcours cyclique entre 0 et 3, si bien que l'indice 6 signifie une écriture dans la position 2.

## Mode point de garde (2)

Le point de garde est écrit en même temps que la position 0 avec la même valeur.

Facilite l'écriture dans des tables prévues pour être lues avec interpolation pour produire des formes d'onde cycliques sans discontinuité. De plus, avant son utilisation, l'indice résultant est augmenté de la moitié de la distance entre une position et la suivante, avant d'être arrondi à l'adresse entière inférieure d'une position dans la table.

Normalement (*iwgmode* = 0 ou 1), pour une table de longueur 5, qui comprend les positions 0 à 3 en partie principale et la position 4 comme point de garde, un indice résultant compris entre 0 et 0.999 provoquera une écriture dans la position 0. ("0.999" signifie juste inférieur à 1.0), entre 1.0 et 1.999, l'écriture se fera dans la position 1, etc. La même interprétation a lieu pour les indices résultants compris entre 0 et 4.999 (*igwmode* = 0) ou 3.999 (*igwmode* = 1). *igwmode* = 0 permet l'écriture dans les positions 0 à 4, avec la possibilité d'avoir dans le point de garde (4) une valeur différente de celle de la position 0.

Avec une table de longueur 5 et *iwgmode* = 2, quand l'indice résultant est compris entre 0 et 0.499, l'écriture se fera dans les positions 0 et 4. S'il est compris entre 0.5 et 1.499, l'écriture se fera dans la position 1, etc. S'il est compris entre 3.5 et 4.0, l'écriture se fera également dans les positions 0 et 4.

Ainsi, l'écriture s'approche le plus possible des résultats de la lecture avec interpolation. Le mode point de garde ne doit être utilisé qu'avec des tables qui ont un point de garde.

Le mode point de garde se fait en ajoutant 0.5 à l'indice résultant, en l'arrondissant à l'entier inférieur le plus proche, puis en le réduisant modulo le facteur de deux égal à la longueur de la table moins un, enfin en écrivant dans la table (positions 0 à 3 dans notre exemple) et dans le point de garde si l'indice vaut 0.

## Exemples

Voici un exemple de l'opcode *tableiw*. Il utilise le fichier *tableiw.csd* [exemples/tableiw.csd].

### Exemple 1254. Exemple de l'opcode *tableiw*.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
; Select audio/midi flags here according to platform
-odac      ;;realtime audio out
;-iadc     ;;uncomment -iadc if realtime audio input is needed too
; For Non-realtime ouput leave only the line below:
; -o tableiw.wav -W ;; for file output any platform
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1
```

```
seed 0 ;generate new values every time the instr is played

instr 1

ifn = p4
isize = p5
ithresh = 0.5

itemp ftgen ifn, 0, isize, 21, 2

iwrite_value = 0
i_index = 0

loop_start:
    iread_value tablei i_index, ifn

    if iread_value > ithresh then
        iwrite_value = 1
    else
        iwrite_value = -1
    endif
    tableiw iwrite_value, i_index, ifn
    loop_lt i_index, 1, isize, loop_start
    turnoff

endin

instr 2

ifn = p4
isize = ftlen(ifn)
prints "Index\tValue\n"

i_index = 0
loop_start:
    ivalue tablei i_index, ifn
    prints "%d:\t%f\n", i_index, ivalue

    loop_lt i_index, 1, isize, loop_start ;read table 1 with our index

aout oscili .5, 100, ifn ;use table to play the polypulse
outs aout, aout

endin
</CsInstruments>
<CsScore>
i 1 0 1 100 16
i 2 0 2 100
e
</CsScore>
</CsoundSynthesizer>
```

## Voir aussi

*tablew, tablewkt*

Plus d'information sur cet opcode : <http://www.csoundjournal.com/issue12/genInstruments.html> [http://www.csoundjournal.com/issue12/genInstruments.html], écrit par Jacob Joaquin.

## Crédits

Auteur : Robin Whittle  
Australie

Mai 1997

Nouveau dans la version 3.47

Mise à jour en août 2002, grâce à Abram Hindle qui a indiqué le syntaxe correcte.

## Routines GEN obsolètes

La routine GEN suivante est obsolète. Elle est encore distribuée avec Csound pour des raisons de compatibilité ascendante. La nouvelle routine GEN à utiliser à sa place est indiquée dans l'entrée suivante.

# GEN22

GEN22 — Obsolète.

## Description

Obsolète depuis la version 4.19. Utiliser plutôt la routine *GEN18*.

---

# Les programmes utilitaires

Dan Ellis, MIT Media Lab

Les utilitaires de Csound sont des programmes de *prétraitement de fichier son* qui retournent de l'information sur un fichier son ou qui créent une version d'analyse de celui-ci à utiliser par certains générateurs de Csound. Bien que destinés à différents usages, ils ont en commun le mécanisme d'accès au fichier son et sont descriptibles comme un ensemble. Les programmes Utilitaires de Fichiers Son peuvent être appelés de deux manières équivalentes :

```
csound [-U nomutilitaire] [options] [noms_fichier]
```

```
nomutilitaire [options] [noms_fichier]
```

Dans le premier cas, l'utilitaire est appelé comme une partie de l'exécutable de Csound, tandis que dans le second il est appelé comme un programme autonome. Le second est plus petit d'environ 200 K, mais les deux formes fonctionnent de manière identique. La première est pratique pour éviter la maintenance et l'utilisation de plusieurs programmes indépendants - un programme fait tout. Quand on utilise cette forme, un *drapeau -U* détecté dans la ligne de commande provoquera l'interprétation des options et des noms suivants comme ceux de l'utilitaire nommé ; cela signifie que le mécanisme de génération de Csound ne sera pas invoqué et que le programme se terminera à la fin du traitement par l'utilitaire.

## Répertoires.

Les noms de fichier sont de deux sortes, fichiers son sources et fichiers d'analyse résultants. Chacun a une convention de nommage hiérarchique, influencée par le répertoire depuis lequel l'utilitaire est appelé. Les fichiers son sources avec un nom de chemin complet (commençant par un point (.), une barre oblique (/), ou pour ThinkC incluant un deux-points (:)), ne seront cherchés que dans le répertoire nommé. Les fichiers son sans chemin seront recherchés d'abord dans le répertoire courant, ensuite dans le répertoire nommé par la variable d'environnement *SSDIR* (si elle est définie), ensuite dans le répertoire nommé par *SFDIR*. Une recherche infructueuse retournera une erreur "cannot open".

Les fichiers d'analyse résultants sont écrits dans le répertoire courant, ou le répertoire nommé si un chemin est inclus. Pour être ordonné, il est bon de séparer les fichiers d'analyse des fichiers son, habituellement dans un répertoire différent référencé par la variable d'environnement *SADIR*. Il est commode de lancer l'analyse depuis le répertoire *SADIR*. Quand un fichier d'analyse est invoqué ultérieurement par un générateur de Csound il est cherché en premier dans le répertoire courant, puis dans le répertoire défini par *SADIR*.

## Formats des fichiers son.

Csound peut lire et écrire des fichiers audio dans différents formats. Les formats d'écriture sont décrits par des options de la commande Csound. En lecture, le format est déterminé par l'en-tête du fichier, et les données sont automatiquement converties en virgule flottante pendant le traitement interne. Quand Csound est installé sur un hôte qui a des conventions de fichier son locales (SUN, NeXT, Macintosh) il peut comprendre de manière conditionnelle du code local qui crée des fichiers son non portables vers d'autres hôtes. Cependant, sur tous les hôtes, Csound peut toujours générer et lire des fichiers du type AIFF, qui est ainsi un format portable. Les bibliothèques de sons échantillonnés sont typiquement en AIFF, et la variable d'environnement *SSDIR* pointe habituellement vers un répertoire contenant de tels sons. S'il est défini, le répertoire *SSDIR* fait partie des chemins de recherche pour l'accès aux fichiers son. Noter que certains sons échantillonnés AIFF ont un mécanisme de boucle audio pour les notes tenues ; les programmes d'analyse ne parcourent les segments de boucle qu'une fois.

Pour les fichiers son sans en-tête, une valeur *SR* peut être fournie par l'*option -r* (ou sa valeur par défaut). Si l'*en-tête SR* et l'option de ligne de commande sont tous deux présents, la valeur de l'option remplacera l'en-tête.

Quand les programmes d'Analyse accèdent à un son, un seul canal est lu. Pour les fichiers stéréo ou quadro, le canal par défaut est le canal un ; d'autres canaux peuvent être obtenus à la demande.

## Génération d'un fichier d'analyse (ATSA, CVANAL, HETRO, LPANAL, PVANAL)

Les utilitaires suivants existent pour l'analyse d'un fichier son :

- *ATSA* : analyse ATS à utiliser avec les opcodes de Csound de *resynthèse ATS*.
- *CVANAL* : analyse de Fourier d'une réponse impulsionnelle pour l'opérateur *convolve*.
- *HETRO* : analyse hétérodyne pour le générateur de Csound *adsyn*.
- *LPANAL* : analyse de codage prédictif linéaire pour les opcodes de Csound de *resynthèse par codage prédictif linéaire (LPC)*.
- *PVANAL* : analyse par vocodeur de phase pour le générateur de Csound *pvoc*.

## atsa

atsa — Effectue une analyse ATS sur un fichier son.

### Description

Analyse ATS à utiliser avec les opcodes de Csound de *Resynthèse ATS*.

### Syntaxe

```
csound -U atsa [options] nomfichier_entree nomfichier_sortie
```

### Initialisation

Les options suivantes peuvent être positionnées pour atsa. (Les valeurs par défaut sont mises entre parenthèses) :

- b début (0,000000 secondes)
- e durée (0,000000 secondes, signifie jusqu'à la fin)
- l fréquence la plus basse (20,000000 Hz)
- H fréquence la plus haute (20000,000000 Hz)
- d déviation en fréquence (0,100000 de la fréquence d'un partiel)
- c cycles par fenêtre (4 cycles)
- w type de fenêtre (type : 1) (Options : 0=BLACKMAN, 1=BLACKMAN\_H, 2=HAMMING, 3=VON-HANN)
- h taille de saut (0,250000 de la taille de fenêtre)
- m magnitude la plus faible (-60,000000)
- t longueur de trajectoire (3 trames)
- s longueur minimale de segment (3 trames)
- g longueur minimale des blancs (3 trames)
- T seuil du SMR (30,000000 dB SPL)
- S SMR Minimum de Segment (60,000000 dB SPL)
- P contribution du dernier pic (0,000000 des paramètres du dernier pic)
- M contribution du SMR (0,500000)
- F Type de Fichier (type : 4) (Options : 1=amp. et fréq. seulement, 2=amp., fréq. et phase, 3=amp., fréq. et résiduel, 4=amp., fréq., phase et résiduel)

### Paramètres

L'analyse ATS a été conçue par Juan Pampin. Pour une information complète sur ATS visiter : <http://www-ccrma.stanford.edu/~juan/ATS.html>.

Les paramètres d'analyse doivent être réglés soigneusement pour l'Algorithme d'Analyse (ATSA) afin de capturer la nature du signal à analyser. Comme ils sont nombreux, ATSH offre la possibilité de les Sauvegarder/Charger dans un Fichier Binaire portant l'extension ".apf". L'extension n'est pas obligatoire, mais recommandée. Une brève explication de chaque Paramètre d'Analyse suit :

1. Début (secs.): la date de début de l'analyse en secondes.
2. Durée (secs.): la durée de l'analyse en secondes. Un zéro signifie la durée entière du fichier son en entrée.
3. Fréquence la Plus Basse (Hz) : ce paramètre va déterminer partiellement la taille de la Fenêtre d'Analyse à utiliser. Pour calculer la taille de la Fenêtre d'Analyse, la période de la Fréquence la Plus Basse en échantillons ( $SR / LF$ ) est multipliée par le nombre de cycles de celle-ci que l'utilisateur veut caser dans la Fenêtre d'Analyse (voir paramètre 6). Cette valeur est arrondie à la plus proche puissance de deux supérieure pour déterminer la taille de la TFR pour l'analyse. Les échantillons en trop sont remplis par

des zéros. Si le signal est un son unique, harmonique, alors la valeur de la Fréquence la Plus Basse sera celle du fondamental ou d'un sous-harmonique de celui-ci. Si le son n'est pas harmonique, alors sa fréquence significative la plus basse pourra être une bonne valeur de départ.

4. Fréquence la Plus Haute (Hz) : fréquence la plus haute à prendre en compte pour la Détection de Pic. Une fois que l'on sait qu'aucune information pertinente ne se trouve au-delà d'une certaine fréquence, l'analyse peut être plus rapide et plus précise en réglant la Fréquence la Plus Haute à cette valeur.
5. Déviation de Fréquence (Rapport) : déviation de fréquence autorisée pour chaque pic dans l'Algorithme de Continuation des Pics, comme fraction de la fréquence concernée. Par exemple, si l'on considère un pic à 440 Hz et une Déviation de 0,1 l'Algorithme de Continuation des Pics n'essaiera de trouver des candidats pour la continuité qu'entre 396 et 484 Hz (10% au-dessus et en-dessous de la fréquence du pic). Une petite valeur produira probablement plus de trajectoires tandis qu'une grande valeur les réduira, mais au prix d'une plus grande difficulté à traiter l'information par la suite.
6. Nombre de Cycles de la Fréquence la Plus Basse à caser dans une Fenêtre d'Analyse : il déterminera aussi partiellement la taille de la Fenêtre de Transformation de Fourier à utiliser. Voir le paramètre 3. Pour des signaux à un seul harmonique, il est supposé être supérieur à 1 (typiquement 4).
7. Taille de Saut (Rapport) : taille de l'intervalle entre une Fenêtre d'Analyse et la suivante exprimée comme une fraction de la Taille de Fenêtre. Par exemple, une Taille de Saut de 0,25 "sautera" de 512 échantillons (les Fenêtres se chevaucheront sur 75% de leur taille). Ce paramètre déterminera aussi la taille des trames d'analyse obtenues. Les signaux qui changent leur spectre très rapidement (comme les sons de la Parole) peuvent nécessiter un taux de trame élevé afin de suivre au mieux leurs changements.
8. Limite d'Amplitude (dB) : la valeur d'amplitude la plus élevée à prendre en compte pour la Détection de Pic.
9. Type de Fenêtre : la forme de la fonction de lissage à utiliser pour l'Analyse de Fourier. Il y a quatre choix possibles pour le moment : Blackman, Blackman-Harris, Von Hann, et Hanning. Des spécifications précises sur celles-ci se trouvent facilement dans la bibliographie sur le traitement numérique du signal.
10. Longueur de Trajectoire (Trames) : L'Algorithme de Continuation des Pics regardera "en arrière" sur un nombre de trames égal à Longueur afin de réaliser sa tâche au mieux, et d'éviter que les trajectoires de fréquence ne s'incurvent trop et perdent leur stabilité. Cependant, une grande valeur pour ce paramètre ralentira l'analyse de manière significative.
11. Longueur Minimale de Segment (Trames) : une fois l'analyse réalisée, les données spectrales peuvent être "nettoyées" durant le post-traitement. Les trajectoires plus petites que cette valeur sont supprimées si leur SMR moyen est inférieur au SMR Minimum de Segment (voir les paramètres 16 et 14). Ceci peut aider à éviter les changements soudains non pertinents tout en gardant un taux de trames élevé, réduisant aussi le nombre de sinusoides épisodiques durant la synthèse.
12. Longueur Minimale des Blancs (Trames) : comme le paramètre 11, celui-ci est aussi utilisé pour nettoyer les données durant le post-traitement. Dans ce cas, les blancs (valeurs d'amplitude nulle, c'est-à-dire le "silence" théorique) contigus dont le nombre de trames est plus grand que Longueur sont remplis avec des valeurs d'amplitude/fréquence obtenues par interpolation linéaire des trames actives adjacentes. Ce paramètre empêche les interruptions soudaines des trajectoires stables tout en gardant un taux de trames élevé.
13. Seuil du SMR (dB SPL) : également un paramètre de post-traitement, le seuil du SMR est utilisé pour éliminer les partiels avec de faibles moyennes.
14. SMR Minimum de Segment (dB SPL) : ce paramètre est utilisé en combinaison avec le paramètre 11. Les segments courts ayant un SMR moyen inférieur à cette valeur seront supprimés durant le post-traitement.



15. Contribution du Dernier Pic (0 à 1) : comme c'est expliqué dans le Paramètre 10, l'Algorithme de Continuation des Pics regarde "en arrière" sur plusieurs trames afin de réaliser sa tâche au mieux. Ce paramètre aidera à pondérer la contribution du premier des pics précédents sur les autres. Une valeur de zéro signifie que tous les pics précédents (jusqu'à la taille du Paramètre 10) sont pris également en compte.

16. Contribution du SMR (0 à 1) : en plus de la proximité en fréquence des pics, l'Algorithme de Continuation des Pics ATS peut utiliser une information psychoacoustique (le Rapport Signal-Masque, ou SMR) pour améliorer les résultats perceptifs. Ce paramètre indique quelle quantité d'information SMR est utilisée durant la détection. Par exemple, une valeur de 0,5 fait que l'Algorithme de Continuation des Pics utilise 50% d'information SMR et 50% d'information de Proximité en Fréquence pour décider quel est le meilleur candidat pour continuer la trajectoire sinusoïdale.

## Exemples

La commande suivante :

```
atsa -b0.1 -e1 -l100 -H10000 -w2 fichieraudio.wav fichieraudio.ats
```

Génère le fichier d'analyse ATS 'fichieraudio.ats' à partir du fichier original 'fichieraudio.wav'. L'analyse commence à partir de 0,1 seconde dans le fichier et elle est effectuée sur 1 seconde. La fréquence la plus basse est 100 Hz et la plus haute est 10 kHz. Une fenêtre de Hamming est utilisée pour chaque trame d'analyse.

## cvanal

cvanal — Convertit un fichier son en une trame de transformée de Fourier.

### Description

Analyse de Fourier d'une réponse impulsionnelle pour l'opérateur *convolve*

### Syntaxe

```
csound -U cvanal [options] nomfichier_entree nomfichier_sortie
cvanal [options] nomfichier_entree nomfichier_sortie
```

### Initialisation

*cvanal* -- convertit un fichier son en une trame de transformée de Fourier. Le fichier de sortie peut être utilisé par l'opérateur *convolve* pour réaliser une Convolution Rapide entre un signal d'entrée et la réponse impulsionnelle originale. L'analyse est conditionnée par les options ci-dessous. Un espace est facultatif entre le drapeau et son argument.

*-s srate* -- taux d'échantillonnage du fichier audio d'entrée. Il remplacera la valeur *srate* de l'en-tête du fichier audio, qui s'applique autrement. Si aucun des deux n'est présent, la valeur par défaut est 10000.

*-c canal* -- numéro du canal à traiter. S'il est omis, tous les canaux sont traités par défaut. Si une valeur est donnée, seul le canal choisi sera traité.

*-b début* -- date du début (en secondes) du segment audio à analyser. La valeur par défaut est 0,0

*-d durée* -- durée (en secondes) du segment audio à analyser. La valeur par défaut de 0,0 signifie jusqu'à la fin du fichier.

*-X* -- écrire le fichier d'analyse dans un format indépendant de la machine.

### Exemples

```
cvanal unson fichiercv
```

analysera le fichier son "unson" pour produire le fichier "fichiercv" à utiliser avec *convolve*.

Pour utiliser des données qui ne sont pas déjà contenues dans un fichier son, un convertisseur de fichier son qui accepte des fichiers texte peut être utilisé pour créer un fichier audio standard, par exemple le format .DAT pour SOX. Ceci est utile pour implémenter des filtres RIF.

### Fichiers

Le fichier de sortie a un en-tête spécial *convolve*, contenant les détails du fichier source audio. Les données d'analyse sont stockées comme des nombres « virgule flottante », en forme rectangulaire (réel/imaginaire).



#### Note

Le fichier d'analyse n'est *pas* indépendant du système ! Assurez-vous que les données originales de la réponse impulsionnelle sont conservées. Si nécessaire, le fichier d'analyse pourra être recréé.

### Crédits

Auteur : Greg Sullivan

Basé sur l'algorithme donné dans *Elements Of Computer Music*, par F. Richard Moore.

## hetro

hetro — Décompose un fichier son en entrée en composantes sinusoïdales.

### Description

Analyse par filtre hétérodyne pour le générateur de Csound *adsyn*.

### Syntaxe

```
csound -U hetro [options] nomfichier_entree nomfichier_sortie
hetro [options] nomfichier_entree nomfichier_sortie
```

### Initialisation

*hetro* prend un fichier son en entrée, le décompose en composantes sinusoïdales, et sort une description de ces composantes sous la forme de pistes de points charnière d'amplitude et de fréquence. L'analyse est conditionnée par les options de contrôle ci-dessous. Un espace est facultatif entre drapeau et argument.

*-s srate* -- taux d'échantillonnage du fichier audio en entrée. Il remplacera la valeur *srate* de l'en-tête du fichier audio, qui s'applique autrement. Si aucun des deux n'est présent, la valeur par défaut est 10000. Noter que pour la synthèse *adsyn* le taux d'échantillonnage du fichier source et de l'orchestre générateur n'ont pas à être les-mêmes.

*-c canal* -- numéro du canal à traiter. La valeur par défaut est 1.

*-b début* -- date de début (en secondes) du segment audio à analyser. La valeur par défaut est 0,0

*-d durée* -- durée (en secondes) du segment audio à analyser. La valeur par défaut de 0,0 signifie jusqu'à la fin du fichier. La longueur maximale est de 32,766 secondes.

*-f freqdeb* -- fréquence de départ estimée du fondamental, nécessaire pour initialiser l'analyse par le filtre. La valeur par défaut est 100 (cps).

*-h partiels* -- nombre d'harmoniques recherchés dans le fichier audio. La valeur par défaut est 10, la valeur maximale dépend de la mémoire disponible.

*-M ampmax* -- amplitude maximale obtenue par addition sur toutes les pistes simultanées. La valeur par défaut est 32767.

*-m ampmin* -- seuil d'amplitude en-dessous duquel une paire de pistes amplitude/fréquence sera considérée comme inactive et ne contribuera pas à la somme en sortie. Valeurs typiques : 128 (48 dB en-dessous de l'échelle complète, 64 (54 dB en-dessous), 32 (60 dB en-dessous), 0 (pas de seuillage). Le seuil par défaut est 64 (54 dB en-dessous).

*-n brkpts* -- nombre initial de points charnière de l'analyse dans chaque piste d'amplitude et de fréquence, avant le seuillage (*-m*) et la consolidation linéaire des points charnière. Les points initiaux sont répartis uniformément sur toute la durée. La valeur par défaut est 256.

*-l cutfreq* -- substitue un filtre passe-bas de Butterworth du 3ème ordre avec une fréquence de coupure *cutfreq* (en Hz), à la place du filtre par défaut qui est un filtre de moyenne en peigne. La valeur par défaut est 0 (ne pas utiliser).

### Exécution

A partir de Csound 4.08, *hetro* peut écrire des fichiers de sortie SDIF si le nom du fichier de sortie se termine par ".sdif" ou ".SDIF". Voir l'utilitaire *sdif2ad* pour plus d'information sur le support de SDIF dans Csound.

## Exemples

```
hetro -s44100 -b.5 -d2.5 -hl6 -M24000 fichieraudio.test adsynfile7
```

Ceci analyse 2,5 secondes du canal 1 du fichier "fichieraudio.test", enregistré à 44,1 kHz, commençant 0,5 secondes après le début, et place le résultat dans le fichier "adsynfile7". Nous ne voulons que les 16 premiers harmoniques du son, avec 256 points charnière par piste d'amplitude ou de fréquence, et un pic de la somme des amplitudes de 24000. Le fondamental est estimé au commencement à 100 Hz. Le seuil d'amplitude est de 54 dB en-dessous de l'échelle complète.

Le filtre passe-bas de Butterworth n'est pas activé.

## Format de Fichier

Le fichier de sortie contient des suites temporelles de valeurs d'amplitude et de fréquence pour chaque harmonique d'une source audio additive complexe. L'information se présente sous la forme de points charnière (date, valeur, date, valeur, ...) en utilisant des entiers sur 16 bit dans l'intervalle 0 - 32767. Le temps est donné en millisecondes, et les fréquences en Hz (cps). Les données des points charnières sont exclusivement non-négatives, et les valeurs -1 et -2 signifient uniquement le début de nouvelles pistes d'amplitude et de fréquence. Une piste se termine par la valeur 32767. Avant d'être écrite en sortie, chaque piste subit une réduction de données par seuillage d'amplitude et consolidation linéaire des points charnière.

Un composant harmonique est défini par deux ensembles de points charnière : un ensemble d'amplitudes, et un ensemble de fréquences. Dans un fichier composé ces ensembles peuvent apparaître dans n'importe quel ordre (amplitude, fréquence, amplitude ....; ou amplitude, amplitude, ..., puis fréquence, fréquence, ...). Durant la resynthèse par *adsyn* les ensembles sont automatiquement appariés (amplitude, fréquence) dans l'ordre dans lequel ils sont trouvés. Il doit y avoir un nombre égal de chaque sorte.

Un fichier de contrôle *adsyn* légal pourrait avoir le format suivant :

```
-1 temps1 valeur1 ... tempsK valeurK 32767 ; points charnière d'amplitude pour le partiel 1
-2 temps1 valeur1 ... tempsL valeurL 32767 ; points charnière de fréquence pour le partiel 1
-1 temps1 valeur1 ... tempsM valeurM 32767 ; points charnière d'amplitude pour le partiel 2
-2 temps1 valeur1 ... tempsN valeurN 32767 ; points charnière de fréquence pour le partiel 2
-2 temps1 valeur1 .....
-2 temps1 valeur1 ..... ; pistes appariables pour les partiels 3 et 4
-1 temps1 valeur1 .....
-1 temps1 valeur1 .....
```

## Crédits

Auteur : Tom Sullivan  
1992

Auteur : John ffitich  
1994

Auteur : Richard Dobson  
2000

Octobre 2002. Merci à Rasmus Ekman, pour l'addition d'une note au sujet du format SDIF.

## lpanal

**lpanal** — Effectue une analyse par prédiction linéaire et par détection de hauteur sur un fichier son.

### Description

Analyse par prédiction linéaire pour les opcodes de Csound *Resynthèse par Codage Prédicatif Linéaire (LPC)*.

### Syntaxe

```
csound -U lpanal [options] nomfichier_entree nomfichier_sortie  
lpanal [options] nomfichier_entree nomfichier_sortie
```

### Initialisation

*lpanal* effectue à la fois une analyse par lpc et par détection de hauteur sur un fichier son pour produire une suite ordonnée de *trames* d'information de contrôle appropriée pour la resynthèse avec Csound. L'analyse est conditionnée par les options de contrôle ci-dessous. Un espace est facultatif entre le drapeau et sa valeur.

*-a* -- [stockage alternatif] demande à *lpanal* d'écrire un fichier avec les valeurs des pôles du filtre plutôt que les fichiers de coefficients de filtre habituels. Quand *lpread* / *lpreson* sont utilisés avec des fichiers de pôles, une stabilisation automatique est effectuée et le filtre ne deviendra pas incontrôlable. (C'est le réglage par défaut dans la GUI Windows) - Changé par Marc Resibois.

*-s srates* -- taux d'échantillonnage du fichier audio d'entrée. Il remplacera la valeur *srates* de l'en-tête du fichier audio, qui s'applique autrement. Si aucun des deux n'est présent, la valeur par défaut est 10000.

*-c canal* -- numéro du canal à traiter. La valeur par défaut est 1.

*-b début* -- date du début (en secondes) du segment audio à analyser. La valeur par défaut est 0,0

*-d durée* -- durée (en secondes) du segment audio à analyser. La valeur par défaut de 0,0 signifie jusqu'à la fin du fichier.

*-p npoles* -- nombres de pôles pour l'analyse. La valeur par défaut est 34, le maximum 50.

*-h taillesaut* -- taille du saut (en échantillons) entre les trames d'analyse. Détermine le nombre de trames par seconde (*srates* / *taillesaut*) dans le fichier de contrôle en sortie. La taille des trames d'analyse est de *taillesaut* \* 2 échantillons. La valeur par défaut est 200, le maximum 500.

*-C chaîne* -- texte pour le champ commentaire de l'en-tête du fichier lp. La valeur par défaut est une chaîne nulle.

*-P mincps* -- fréquence la plus basse (en Hz) pour la détection de hauteur. -P0 signifie pas de détection de hauteur.

*-Q maxcps* -- fréquence la plus haute (en Hz) pour la détection de hauteur. Plus l'intervalle de hauteurs est étroit, plus l'estimation de hauteur est précise. Les valeurs par défaut sont -P70, -Q200.

*-v verbosité* -- niveau d'information affiché sur le terminal pendant l'analyse.

- 0 = aucune
- 1 = verbeux
- 2 = débogage

La valeur par défaut est 0.

-X -- écrire le fichier d'analyse dans un format indépendant de la machine.

## Exemples

```
lpanal -a -p26 -d2.5 -P100 -Q400 fichieraudio.test lpfil22
```

analysera les premières 2,5 secondes du fichier "fichieraudio.test", produisant *srate* / 200 trames par seconde, chacune contenant les coefficients d'un filtre à 26 pôles et une estimation de hauteur entre 100 et 400 Hz. La sortie stabilisée (-a) sera placée dans "lpfil22" dans le répertoire courant.

## Format de Fichier

La sortie est un fichier constitué d'un en-tête identifiable plus un ensemble de trames de données d'analyse en virgule flottante. Chaque trame contient quatre valeurs d'information de hauteur et de gain, suivies par *npoles* coefficients de filtre. Le fichier est lisible par l'opcode *lpread* de Csound.

*lpanal* est une modification importante des programmes d'analyse lpc de Paul Lanksy.

## pvanal

pvanal — Convertit un fichier son en une série de trames de transformation de Fourier à court terme.

### Description

Analyse de Fourier pour le générateur de Csound *pvoc*

### Syntaxe

```
csound -U pvanal [options] nomfic_entree nomfic_sortie
```

```
pvanal [options] nomfic_entree nomfic_sortie
```

### Extension de pvanal pour créer un fichier PVOC-EX.

L'utilitaire standard de Csound pvanal a été étendu pour permettre la création d'un fichier au format PVOC-EX, en utilisant l'interface existante. Pour créer un fichier PVOC-EX, le nom de fichier doit avoir comme extension « .pvx », par exemple « test.pvx ». La nécessité pour la taille de TFR d'être une puissance de deux n'est plus obligatoire ici, et n'importe quelle valeur positive est acceptée ; les nombres impairs sont arrondis en interne. Cependant, les tailles en puissance de deux sont toujours préférables pour toutes les applications normales.

Les drapeaux de sélection de canal sont ignorés, et tous les canaux de la source seront analysés et écrits dans le fichier de sortie, jusqu'à la limite, fixée à la compilation, de huit canaux. La taille de la fenêtre d'analyse (*itailfen*) est fixée en interne au double de la taille de la TFR.

### Initialisation

*pvanal* convertit un fichier son en une série de trames de transformation de Fourier à court terme (STFT) à espacement temporel régulier (une représentation du domaine fréquentiel). Le fichier de sortie peut être utilisé par *pvoc* pour générer des fragments audio basés sur le son échantillonné original, avec des échelles de temps et des hauteurs arbitraires et modifiées dynamiquement. L'analyse est conditionnée par les options ci-dessous. Un espace est facultatif entre le drapeau et son argument.

*-s srate* -- taux d'échantillonnage du fichier audio d'entrée. Il remplacera la valeur *srate* de l'en-tête du fichier audio, qui s'applique autrement. Si aucun des deux n'est présent, la valeur par défaut est 10000.

*-c canal* -- numéro du canal à traiter. La valeur par défaut est 1.

*-b début* -- date du début (en secondes) du segment audio à analyser. La valeur par défaut est 0,0

*-d durée* -- durée (en secondes) du segment audio à analyser. La valeur par défaut de 0,0 signifie la fin du fichier.

*-n tailletrame* -- taille de trame STFT, le nombre d'échantillons dans chaque trame de l'analyse de Fourier. Doit être une puissance de deux dans l'intervalle 16 à 16384. Pour des résultats propres, une trame doit être plus grande que la période de hauteur la plus longue du son échantillonné. Cependant, des trames très longues donnent un "brouillage" temporel ou une réverbération. La largeur de bande de chaque bin de STFT est déterminée par le rapport *srate / tailletrame*. La taille de trame par défaut est la plus petite puissance de deux qui correspond à plus de 20 ms de la source (par exemple 256 points avec un échantillonnage à 10 kHz, donnant une trame de 25,6 ms).

*-w factfen* -- facteur de chevauchement de fenêtre. Il contrôle le nombre de trames de transformation de Fourier par seconde. *pvoc* interpolera entre les trames, mais un nombre insuffisant de trames générera des distorsions audibles ; trop de trames donneront un fichier d'analyse gigantesque. 4 est un bon compromis



pour *factfen*, signifiant que chaque point d'entrée apparaît dans 4 fenêtres de sortie, ou inversement que le décalage entre trames de STFT successives est *tailletrame* / 4. La valeur par défaut est 4. N'utilisez pas cette option en même temps que *-h*.

*-h taillesaut* -- décalage de trame STFT. Le contraire de l'option précédente, spécifiant l'incrément en échantillons entre les trames d'analyse successives (voir aussi *lpanal*). N'utilisez pas cette option en même temps que *-w*.

*-H* -- utilise une fenêtre de Hamming à la place de la fenêtre de von Hann employée par défaut.

*-K* -- utilise une fenêtre de Kaiser à la place de la fenêtre de von Hann employée par défaut. Le paramètre de la fenêtre de Kaiser vaut 6,8 par défaut, mais il peut être fixé avec l'option *-B*.

*-B beta* -- fixe le paramètre beta d'une fenêtre de Kaiser utilisée, à la valeur en virgule flottante *beta*.

## Exemples

```
pvanal unson fichierpv
```

analysera le fichier son "unson" en utilisant les valeurs par défaut de *tailletrame* et de *factfen* pour produire le fichier "pvfile" approprié pour une utilisation avec *pvoc*.

## Fichiers

Le fichier de sortie a un en-tête spécial *pvoc* contenant les détails du fichier source audio, le taux des trames d'analyse et le facteur de chevauchement. Les trames de données de l'analyse sont stockées en virgule flottante, avec la magnitude et la « fréquence » (en Hz) des  $N/2 + 1$  premiers bins de Fourier de chaque trame successive. La « fréquence » encode l'incrément de phase de façon à donner une bonne indication de la fréquence réelle pour les harmoniques à fort niveau. Pour les faibles amplitudes ou les harmoniques évoluant rapidement c'est moins significatif.

## Diagnostic

Imprime le nombre total de trames, et le nombre de trames complétées toutes les 20 trames.

## Crédits

Auteur : Dan Ellis

MIT Media Lab

Cambridge, Massachusetts

1990

## Requêtes sur un fichier (SNDINFO)

L'utilitaire suivant existe pour les requêtes sur un fichier son :

- *SNDINFO*: affiche de l'information sur un fichier son.

## sndinfo

sndinfo — Affiche de l'information sur un fichier son.

### Description

Fournit l'information de base sur un ou plusieurs fichiers son.

### Syntaxe

```
csound -U sndinfo [options] fichierson ...
```

```
sndinfo [options] fichierson ...
```

### Initialisation

*sndinfo* tentera de trouver chaque fichier nommé, de l'ouvrir en lecture, de lire l'en-tête du fichier son, pour ensuite imprimer un rapport sur l'information de base trouvée. L'ordre de recherche dans les répertoires de fichiers son est celle qui a été décrite précédemment. Si le fichier est de type AIFF, quelques détails plus avancés sont listés en premier.

Il y a deux types d'options :

1. *-i* ou *-il* imprimera l'information d'instrument, qui comprend les boucles. L'option continue jusqu'à une option *-i0*.
2. L'autre option est *-b* qui imprime l'information de diffusion pour les fichier WAV. Elle peut être arrêtée de façon similaire avec *-b0*.

### Exemples

```
csound -U sndinfo test Bosendorfer/"BOSEN mf A0 st" foo foo2
```

où l'on a les variables d'environnement SFDIR = /u/bv/sound, et SSDIR = /so/bv/Samples, pourra produire ceci :

```
util  SNDINFO:
      /u/bv/sound/test:
          srate 22050, monaural, 16 bit shorts, 1.10 seconds
          headersiz 1024, datasiz 48500  (24250 sample frames)

      /so/bv/Samples/Bosendorfer/BOSEN mf A0 st:  AIFF, 197586 stereo samples, base Frq 261.6 (MIDI 60),
      AIFF soundfile, looping with modes 1, 0
          srate 44100, stereo, 16 bit shorts, 4.48 seconds

          headersiz  402, datasiz 790344  (197586 sample frames)

      /u/bv/sound/foo:
          no recognizable soundfile header

      /u/bv/sound/foo2:
          couldn't find
```

## Conversion de fichier (, HET\_EXPORT, HET\_IMPORT, PVLOOK, PV\_EXPORT, PV\_IMPORT, SDIF2AD, SR-CONV)

Les utilitaires suivants existent pour la conversion de fichier :

- *HET\_EXPORT* : exporte un fichier *.het* (produit par *HETRO*) vers un fichier texte à séparateur virgule.
- *HET\_IMPORT* : génère un fichier *.het* (dans le format produit par *HETRO*) à partir d'un fichier texte à séparateur virgule pour l'utiliser avec le générateur *adsyn*.
- *PVLOOK* : affiche une sortie texte formatée de fichiers d'analyse STFT.
- *PV\_EXPORT* : convertit un fichier généré par *PVANAL* en un fichier texte.
- *PV\_IMPORT* : convertit un fichier texte (dans le format généré par *PV\_EXPORT*) en un fichier de format *PVANAL* à utiliser par l'opcode *pvoc*.
- *SDIF2AD* : convertit des fichiers SDIF en fichiers utilisables par *adsynt*.
- *SRCONV* : convertit le taux d'échantillonnage d'un fichier audio.
- *SRC\_CONV* : convertit le taux d'échantillonnage d'un fichier audio.

## dnoise

dnoise — Réduit le bruit dans un fichier.

### Description

C'est un schéma de réduction de bruit au moyen du seuillage de bruit dans le domaine fréquentiel.

### Syntaxe

```
dnoise [options] -i ficref_bruit -o ficson_sortie ficson_entree
```

### Initialisation

Options spécifiques à dnoise :

- (*pas d'option*) fichier son en entrée à débruiter
- *-i nomfic* fichier de référence du bruit en entrée
- *-o nomfic* fichier son de sortie
- *-N fnum* nombre de filtres passe-bande (par défaut : 1024)
- *-w fovlp* facteur de chevauchement des filtres : {0,1,(2),3} NE PAS UTILISER *-w* ET *-M*
- *-M longfa* longueur de la fenêtre d'analyse (par défaut : N-1 à moins que *-w* ne soit spécifié)
- *-L longfs* longueur de la fenêtre de synthèse (par défaut : M)
- *-D factd* facteur de décimation (par défaut : M/8)
- *-b datedeb* date de début dans le fichier de référence du bruit (par défaut : 0)
- *-B smpdeb* échantillon de départ dans le fichier de référence du bruit (par défaut : 0)
- *-e datefin* date de fin dans le fichier de référence du bruit (par défaut : fin du fichier)
- *-E smpfin* échantillon de fin dans le fichier de référence du bruit (par défaut : fin du fichier)
- *-t seuil* seuil au-dessus du bruit de référence en dB (par défaut : 30)
- *-S gfact* raideur de la coupure au seuil de bruit, intervalle : 1 à 5 (par défaut : 1)
- *-n nbrtrm* nombre de trames de TFR sur lesquelles calculer la moyenne (par défaut : 5)
- *-m gainmin* gain minimum du seuillage de bruit lorsqu'il est fermé (par défaut : -40)

Options de format du fichier son :

- *-A* format de sortie AIFF
- *-W* format de sortie WAV
- *-J* format de sortie IRCAM
- *-h* pas d'en-tête de fichier (non valide pour une sortie AIFF/WAV)
- *-8* échantillons en caractères non signés sur 8 bit

- *-c* échantillons en caractères signés sur 8 bit
- *-a* échantillons en alaw
- *-u* échantillons en ulaw
- *-s* échantillons en entiers courts
- *-l* échantillons en entiers longs
- *-f* échantillons en virgule flottante. Les nombres en virgule flottante sont aussi supportés par les fichiers WAV. (Nouveau dans Csound 3.47.)

Options supplémentaires :

- *-R* verbose - impression d'une information d'état
- *-H [N]* imprime un caractère de type pulsation à chaque écriture dans le fichier son.
- *-- nomfic* sortie de journal dans le fichier nomfic
- *-V* verbose - impression d'une information d'état



## Note

DNOISE consulte aussi la variable d'environnement SFOUTYP pour déterminer le format du fichier de sortie.

L'option *-i* est utilisée pour un fichier de référence du bruit (créé normalement à partir d'un court extrait du fichier à débruiter, dans lequel seul le bruit est audible). Le fichier son d'entrée à débruiter peut être donné n'importe où dans la ligne de commande, sans drapeau.

## Exécution

C'est un schéma de réduction de bruit au moyen du seuillage de bruit dans le domaine fréquentiel. Il fonctionnera mieux dans le cas d'un rapport signal/bruit élevé avec un bruit de type souffle.

L'algorithme est celui suggéré par Moorer & Berger dans « Linear-Phase Bandsplitting: Theory and Applications » présenté à la 76ème Convention, 8-11 Octobre 1984 à New York, de l'Audio Engineering Society (préimpression #2132) sauf qu'il utilise la formulation par Chevauchement-Addition Pondéré pour l'analyse et la synthèse de Fourier à court terme au lieu de la formulation récursive proposée par Moorer & Berger. Le gain pour chaque bin de fréquence est calculé indépendamment selon la formule

$$\text{gain} = g0 + (1-g0) * [\text{moy} / (\text{moy} + \text{th} * \text{th} * \text{nref})] ^ \text{sh}$$

où *moy* et *nref* sont la moyenne quadratique du signal et du bruit respectivement pour le bin en question. (Ceci diffère légèrement de la version dans Moorer & Berger.)

Les paramètres critiques *th* et *g0* sont spécifiés en dB et convertis en interne en valeurs décimales. Les valeurs *nref* sont calculées au début du programme sur la base d'un fichier de bruit (spécifié dans la ligne de commande) qui contient du bruit sans signal.

Les valeurs *moy* sont calculée sur une fenêtre rectangulaire de *m* trames de TFR centrée sur la date courante. Cela correspond à une extension temporelle de *m*\*D/R (qui vaut typiquement (*m*\*N/8)/R ). Le réglage par défaut de N, m, et D devrait convenir pour la plupart des utilisations. Un taux d'échantillonnage supérieur à 16 kHz pourrait signifier un N plus grand.

## Crédits

Auteur : Mark Dolson

26 août 1989

Auteur : John ffitch

30 décembre 2000

Mis à jour par Rasmus Ekman le 11 mars 2002.

## het\_export

`het_export` — Convertit un fichier .het en fichier texte à séparateur virgule.

### Syntaxe

```
het_export fichier_het fichier_textecsv  
csound -U het_export fichier_het fichier_textecsv
```

### Initialisation

*fichier\_het* - Nom du fichier d'entrée .het.

*fichier\_textecsv* - Nom du fichier texte à séparateur virgule.

L'utilitaire *het\_export* génère un fichier texte à séparateur virgule pour pouvoir éditer manuellement un fichier .het produit par l'utilitaire *HETRO*. On peut l'utiliser en combinaison avec *het\_import* pour produire des données pour le générateur *adsyn*.

### Crédits

Auteur : John ffitch

1995

## het\_import

het\_import — Convertit un fichier texte à séparateur virgule en un fichier .het

### Syntaxe

```
het_import fichier_textecsv fichier_het  
csound -U het_import fichier_textecsv fichier_het
```

### Initialisation

*fichier\_textecsv* - Nom du fichier texte à séparateur virgule.

*fichier\_het* - Nom du fichier .het de sortie.

L'utilitaire *het\_import* génère un fichier *.het* utilisable avec le générateur *adsyn*. Il peut être utilisé en combinaison avec *het\_export* pour modifier l'analyse du son faite par l'utilitaire *HETRO*.

### Crédits

Auteur : John ffitch

1995





---

3452

Bin 1	Amps.	0.180	0.066	0.252	0.248	0.245	0.246	0.246	0.249
0.252	0.251	0.250	0.248	0.244	0.245	0.248	0.250	0.254	0.251
0.248	0.247	0.244	0.246	0.249	0.250	0.253	0.251	0.247	0.246
0.245	0.246	0.250	0.251	0.252	0.250	0.247	0.245	0.246	0.247
0.251	0.252	0.250	0.249	0.246	0.245	0.248	0.249	0.252	0.253
0.249	0.248	0.245	0.245	0.249	0.251	0.252	0.252	0.249	0.246
0.246	0.245	0.249	0.252	0.252	0.251	0.249	0.245	0.246	0.248
0.250	0.253	0.251	0.249	0.247	0.244	0.247	0.249	0.250	0.253
0.251	0.248	0.247	0.245	0.247	0.250	0.252	0.252	0.251	0.247
0.246	0.246	0.247	0.251	0.252	0.251	0.249	0.246	0.245	0.248
0.249	0.252	0.252	0.249	0.248	0.246	0.245	0.249	0.250	0.252
0.252	0.249	0.247	0.246	0.246	0.249	0.252	0.252	0.251	0.248
0.245	0.246	0.247	0.249	0.253	0.251	0.249	0.247	0.245	0.246
0.248	0.250	0.253	0.251	0.248	0.247	0.244	0.246	0.250	0.251
0.252	0.250	0.247	0.246	0.246	0.248	0.251	0.252	0.251	0.250
0.246	0.245	0.247	0.248	0.251	0.252	0.250	0.248	0.246	0.245
0.248	0.249	0.252	0.252	0.248	0.247	0.245	0.245	0.249	0.251
0.251	0.251	0.248	0.246	0.246	0.247	0.250	0.252	0.251	0.250
0.248	0.244	0.246	0.248	0.250	0.253	0.251	0.248	0.247	0.245
0.247	0.249	0.250	0.252	0.250	0.247	0.246	0.245	0.247	0.251
0.252	0.251	0.250	0.246	0.245	0.247	0.248	0.252	0.252	0.249
0.248	0.245	0.245	0.248	0.249	0.251	0.252	0.248	0.247	0.245
0.245	0.249	0.250	0.251	0.251	0.248	0.246	0.245	0.246	0.249
0.252	0.251	0.250	0.247	0.244	0.246	0.247	0.249	0.252	0.251
0.249	0.247	0.244	0.247	0.249	0.250	0.252	0.250	0.247	0.246
0.245	0.247	0.250	0.251	0.251	0.250	0.246	0.245	0.246	0.248
0.251	0.252	0.250	0.249	0.245	0.245	0.247	0.248	0.251	0.252
0.249	0.247	0.245	0.245	0.248	0.250	0.251	0.251	0.247	0.246
0.245	0.245	0.249	0.251	0.251	0.250	0.247	0.245	0.246	0.246
0.249	0.252	0.251	0.249	0.247	0.244	0.247	0.248	0.250	0.252
0.250	0.247	0.246	0.245	0.247	0.250	0.251	0.252	0.249	0.246
0.245	0.245	0.247	0.251	0.251	0.250	0.249	0.246	0.245	0.247
0.248	0.251	0.251	0.249	0.248	0.245	0.245	0.248	0.249	0.251
0.251	0.248	0.246	0.245	0.245	0.249	0.251	0.251	0.251	0.247

---

3454

3.288 3.290 3.291 3.290 3.294 3.293 3.290 3.292 3.289 3.289  
3.293 3.290 3.292 3.293 3.289 3.291 3.290 3.289 3.293 3.292  
3.291 3.293 3.289 3.289 3.291 3.289 3.292 3.293 3.290 3.292  
3.290 3.288 3.292 3.291 3.291 3.294 3.290 3.290 3.291 3.288  
3.291 3.292 3.291 3.293 3.291 3.288 3.291 3.289 3.290 3.293  
3.290 3.292 3.292 3.288 3.291 3.291 3.290 3.293 3.291 3.290  
3.292 3.288 3.289 3.292 3.290 3.292 3.293 3.289 3.291 3.289  
3.288 3.293 3.291 3.291 3.292 3.288 3.289 3.290 3.288 3.292  
3.293 3.290 3.292 3.289 3.288 3.291 3.290 3.291 3.293 3.289  
3.290 3.290 3.287 3.291 3.291 3.290 3.293 3.290 3.288 3.290  
3.288 3.290 3.293 3.291 3.292 3.291 3.288 3.290 3.289 3.289  
3.293 3.290 3.290 3.291 3.287 3.289 3.291 3.289 3.292 3.291  
3.288 3.290 3.288 3.288 3.292 3.290 3.291 3.292 3.288 3.289  
3.290 3.288 3.292 3.292 3.290 3.292 3.289 3.288 3.291 3.289  
3.291 3.293 3.289 3.291 3.290 3.287 3.291 3.290 3.290 3.293  
3.289 3.289 3.290 3.287 3.290 3.292 3.290 3.292 3.290 3.287  
3.290 3.289 3.289 3.292 3.290 3.290 3.291 3.287 3.289 3.290  
3.289 3.292 3.291 3.289 3.291 3.288

*etc...*

## Crédits

Auteur : Richard Karpen

Seattle, Wash

1993 (Nouveau dans la version 3.57 de Csound)

## **pv\_export**

**pv\_export** — Convertit un fichier .pvx en fichier texte à séparateur virgule.

### **Syntaxe**

```
pv_export fichier_pv fichier_texte_csv  
csound -U pv_export fichier_pv fichier_texte_csv
```

### **Initialisation**

*fichier\_pv* - Nom du fichier d'entrée .pvx.

*fichier\_texte\_csv* - Nom du fichier texte à séparateur virgule de sortie.

L'utilitaire *pv\_export* génère un fichier texte à séparateur virgule pour une édition manuelle d'un fichier .pvx produit par l'utilitaire *PVANAL*. Il peut être utilisé en combinaison avec *pv\_import* pour produire des données pour le générateur *pvoc*.

### **Crédits**

Auteur : John ffitch

1995

## pv\_import

`pv_import` — Convertit un fichier texte à séparateur virgule en un fichier .pvx.

### Syntaxe

```
pv_import fichier_texte_csv fichier_pv  
csound -U pv_import fichier_texte_csv fichier_pv
```

### Initialisation

*fichier\_texte\_csv* - Nom du fichier texte à séparateur virgule en entrée.

*fichier\_pv* - Nom du fichier .pvx de sortie.

L'utilitaire *pv\_import* génère un fichier .pvx utilisable avec le générateur *pvoc*. Il peut être utilisé en combinaison avec *pv\_export* pour modifier une analyse de son faite par l'utilitaire *PVANAL*.

### Crédits

Auteur : John ffitch

1995

## sdif2ad

sdif2ad — Convertit des fichiers SDIF en fichiers utilisables par adsyn.

### Description

Convertit des fichiers Sound Description Interchange Format (SDIF) dans le format utilisable par l'opcode de Csound *adsyn*. A partir de la version 4.10 de Csound, *sdif2ad* n'est plus disponible que comme un programme autonome pour console Windows et pour DOS.

### Syntaxe

```
sdif2ad [options] fichier_entree fichier_sortie
```

### Initialisation

Options :

- **-sN** -- applique le facteur d'échelle d'amplitude N
- **-pN** -- ne garde que les N premiers partiels. Limité à 1024 partiels. Les indices de piste de partiels de la source sont utilisés directement pour sélectionner le stockage interne. Comme ils peuvent avoir des valeurs arbitraires, le maximum de 1024 partiels peut ne pas être réalisé dans tous les cas.
- **-r** -- fichier de données de sortie en octets inversés. L'option octets inversés est là pour faciliter le transfert entre plates-formes, car le format de fichier *adsyn* de Csound n'est pas portable.

Si le nom de fichier passé à *hetro* a l'extension « .sdif », les données seront écrites en format SDIF comme des trames 1TRC de données de synthèse additive. Le programme utilitaire *sdif2ad* peut être utilisé pour convertir tout fichier SDIF contenant un flot de données 1TRC dans le format *adsyn* de Csound. *sdif2ad* permet à l'utilisateur de limiter le nombre de partiels retenus, et d'appliquer un facteur d'échelle d'amplitude. Ceci est souvent nécessaire, car la spécification SDIF, depuis la réalisation de *sdif2ad*, ne nécessite pas que les amplitudes soient dans un intervalle particulier. *sdif2ad* rapporte sur la console l'information sur le fichier, y compris l'intervalle de fréquence.

Les principaux avantages de SDIF sur le format *adsyn*, pour les utilisateurs de Csound, sont que les fichiers SDIF sont totalement portables d'une plate-forme à l'autre (les données sont en « big-endian »), et qu'ils n'ont pas la limite de durée de 32,76 secondes imposée par le format *adsyn* sur 16 bit. Cette limite est nécessairement imposée par *sdif2ad*. Dans le futur, la lecture du format SDIF pourra être incorporée directement dans *adsyn*, permettant ainsi l'analyse et le traitement de fichiers de n'importe quelle longueur (seulement limitée par la capacité mémoire du système).

Les utilisateurs doivent se souvenir que les formats SDIF sont toujours en développement. Bien que le format 1TRC soit maintenant bien établi, il peut encore changer.

Pour des informations détaillées sur le Sound Description Interchange Format, se référer au site web du CNMAT : <http://cnmat.CNMAT.Berkeley.EDU/SDIF>

D'autres ressources SDIF (y compris un visionneur) sont disponibles via le site web de NC\_DREAM : <http://www.bath.ac.uk/~masjpf/NCD/dreamhome.html>

### Crédits

Auteur : Richard Dobson

Somerset, England



Août 2000

Nouveau dans la version 4.08 de Csound

## srconv

srconv — Convertit le taux d'échantillonnage d'un fichier audio.

### Description

Convertit le taux d'échantillonnage d'un fichier audio de *Rin* à *Rout*. Optionnellement le rapport (*Rin* / *Rout*) peut varier linéairement dans le temps selon un ensemble de paires (temps, rapport) dans un fichier auxiliaire.



#### Note

Cet utilitaire n'est pas fiable. Il vaut mieux utiliser libsampleate. *srconv* sera supprimé et remplacé prochainement.

### Syntaxe

```
srconv [options] fichier_entree
```

### Initialisation

Options :

- *-P num* = rapport de transposition en hauteur (*srate* / *r*) [ne pas spécifier à la fois *P* et *r*]
- *-Q num* = facteur de qualité (1, 2, 3 ou 4 : par défaut = 2)
- *-i nomfic* = fichier auxiliaire de points charnière (pas de point charnière par défaut, c'est-à-dire pas de changement de rapport)
- *-r num* = taux d'échantillonnage en sortie (doit être spécifié)
- *-o nomfic* = nom du fichier son de sortie
- *-A* = crée un fichier son de sortie au format AIFF
- *-J* = crée un fichier son de sortie au format IRCAM
- *-W* = crée un fichier son de sortie au format WAV
- *-h* = pas d'en-tête dans le fichier son de sortie
- *-c* = échantillons en caractères signés sur 8 bit
- *-a* = échantillons alaw
- *-8* = échantillons en caractères non-signés sur 8 bit
- *-u* = échantillons ulaw
- *-s* = échantillons en entiers courts
- *-l* = échantillons en entiers longs
- *-f* = échantillons en virgule flottante
- *-r N* = remplace le *srate* de l'orchestre
- *-K* = ne génère pas de bloc de pics d'amplitude

- *-R* = réécrit continuellement l'en-tête pendant l'écriture du fichier son (WAV/AIFF)
- *-H#* = imprime une pulsation dans le style 1, 2 ou 3 à chaque écriture dans le fichier son
- *-N* = notification (cloche système) quand le traitement est fini
- *-- nomfic* = compte-rendu dans un fichier

Ce programme effectue une conversion arbitraire du taux d'échantillonnage en haute fidélité. La méthode consiste à parcourir le fichier d'entrée avec un pas d'incrément conforme au taux d'échantillonnage désiré, et de calculer les points de sortie comme moyennes convenablement pondérées des points voisins. Il y a deux cas à considérer :

1. les taux d'échantillonnage sont dans un petit rapport entier - les poids sont obtenus de la table
2. les taux d'échantillonnage sont dans un grand rapport entier - les poids sont linéairement interpolés de la table.

Calcul de l'incrément : pour une décimation, la fenêtre est la réponse impulsionnelle d'un filtre passe-bas avec une fréquence de coupure située à la moitié de la fréquence d'échantillonnage en sortie ; pour une interpolation, la fenêtre est la réponse impulsionnelle d'un filtre passe-bas avec une fréquence de coupure située à la moitié de la fréquence d'échantillonnage de l'entrée.

## Voir aussi

*src\_conv*

## Crédits

Auteur : Mark Dolson

26 août 1989

Auteur : John ffitich

30 décembre 2000

## src\_conv

src\_conv — Convertit le taux d'échantillonnage d'un fichier audio.

### Description

Convertit le taux d'échantillonnage d'un fichier audio de *Rin* à *Rout*. Optionnellement le rapport (*Rin* / *Rout*) peut varier linéairement dans le temps selon un ensemble de paires (temps, rapport) dans un fichier auxiliaire.

### Syntaxe

```
src_conv [options] fichier_entree
```

### Initialisation

Options :

- *-P num* = rapport de transposition en hauteur (*srate* / *r*) [ne pas spécifier à la fois *P* et *r*]
- *-Q num* = facteur de qualité (1, 2, 3, 4 ou 5 : par défaut = 3)
- *-i nomfic* = fichier auxiliaire de points charnière (pas de point charnière par défaut, c'est-à-dire pas de changement de rapport)
- *-r num* = taux d'échantillonnage en sortie (doit être spécifié si *P* est absent)
- *-o nomfic* = nom du fichier son de sortie
- *-A* = crée un fichier son de sortie au format AIFF
- *-J* = crée un fichier son de sortie au format IRCAM
- *-W* = crée un fichier son de sortie au format WAV
- *-h* = pas d'en-tête dans le fichier son de sortie
- *-c* = échantillons en caractères signés sur 8 bit
- *-a* = échantillons alaw
- *-8* = échantillons en caractères non-signés sur 8 bit
- *-u* = échantillons ulaw
- *-s* = échantillons en entiers courts
- *-l* = échantillons en entiers longs
- *-f* = échantillons en virgule flottante
- *-r N* = remplace le *srate* de l'orchestre
- *-K* = ne génère pas de bloc de pics d'amplitude
- *-R* = réécrit continuellement l'en-tête pendant l'écriture du fichier son (WAV/AIFF)
- *-H#* = imprime une pulsation dans le style 1, 2 ou 3 à chaque écriture dans le fichier son

- *-N* = notification (cloche système) quand le traitement est fini
- *-- nomfic* = compte-rendu dans un fichier

Ce programme effectue une conversion arbitraire du taux d'échantillonnage en haute fidélité en utilisant la bibliothèque *libsamplerate*.

Les cinq niveaux de précision sont :

- 1 : conversion linéaire. La qualité est médiocre mais la conversion est incroyablement rapide.
- 2 : conversion avec maintien d'ordre zéro (la valeur interpolée est égale à la dernière valeur). La qualité est médiocre mais la conversion est incroyablement rapide.
- 3 : c'est l'interpolateur à bande limitée le plus rapide et il a un rapport signal bruit de 97 dB et une bande passante de 80%.
- 4 : un autre interpolateur à bande limitée ressemblant au précédent. Il a un rapport signal bruit de 97 dB et une bande passante de 90%. La vitesse de conversion est plus rapide que la suivante. Très rapide.
- 5 : c'est un interpolateur à bande limitée dérivé de la fonction mathématique *sinc* et c'est le convertisseur basé sur *sinc* qui a la meilleure qualité, avec un rapport signal bruit de 97 dB dans le pire des cas et une bande passante de 97%.

## Voir aussi

*srconv*

## Crédits

Auteur : John ffitch basé sur du code d'Erik de Castro Lopo.

Mai 2015

## Autres utilitaires de Csound (CS, CSB64ENC, ENVEXT, EXTRACTOR, MAKECSD, MIXER, SCALE, MKDB)

Les divers utilitaires suivants sont disponibles :

- *CS* : démarre Csound avec un ensemble d'options qui peuvent être contrôlées par des variables d'environnement, et des fichiers d'entrée et de sortie déterminés par la racine de nom de fichier spécifiée.
- *CSB64ENC* : convertit un fichier binaire en un fichier texte encodé en Base64.
- *ENVEXT* : extrait l'enveloppe d'un fichier vers une liste textuelle.
- *EXTRACTOR* : extrait une section audio d'un fichier audio.
- *MAKECSD* : crée un fichier CSD à partir des fichiers d'entrée spécifiés.
- *MIXER* : mélange ensemble plusieurs fichiers son.
- *SCALE* : calibre l'amplitude d'un fichier son.
- *MKDB* : crée un catalogue des opcodes greffons.

## CS

**cs** — Démarre Csound avec un ensemble d'options qui peuvent être contrôlées par des variables d'environnement, et des fichiers d'entrée et de sortie déterminés par la racine de nom de fichier spécifiée.

### Description

Démarre Csound avec un ensemble d'options qui peuvent être contrôlées par des variables d'environnement, et des fichiers d'entrée et de sortie déterminés par la racine de nom de fichier spécifiée.

### Syntaxe

```
cs [-OPTIONS] <nom> [OPTIONS DE CSOUND ... ]
```

### Initialisation

Drapeaux :

- - *OPTIONS* = OPTIONS est une séquence de caractères alphabétiques qui peut être utilisée pour sélectionner l'exécutable Csound à lancer, aussi bien que les options de ligne de commande (voir ci-dessous). L'option 'r' est une valeur par défaut (sélection de la sortie en ), mais on peut la remplacer.
- <nom> = c'est la racine de nom de fichier pour sélectionner les fichiers arguments ; elle peut contenir un chemin. Les fichiers qui ont pour extension .csd, .orc, ou .sco sont recherchés, et soit un CSD soit une paire orc/sco qui correspond à <nom>, le meilleur des deux, est sélectionné. Des fichiers MIDI avec une extension .mid sont aussi recherchés, et si l'un deux correspond à <nom> au moins autant que le CSD ou la paire orc/sco, il est utilisé avec l'option -F.



#### NOTE

Le fichier MIDI n'est pas utilisé si une option -M ou -F est spécifiée par l'utilisateur (nouveau dans la version 4.24.0). A moins qu'il y ait une option (-n ou -o) relative à la sortie audio, un nom de fichier de sortie avec l'extension appropriée est généré automatiquement (basé sur le nom des fichiers d'entrée sélectionnés et sur les options de format). Le fichier de sortie est toujours écrit dans le répertoire courant.



#### NOTE

les extensions de nom de fichier ne sont pas sensibles à la casse.

- [*OPTIONS DE CSOUND ...*] = n'importe quel nombre d'options supplémentaires pour Csound qui sont simplement copiées dans la ligne de commande finale qui sera exécutée.

La ligne de commande qui est exécutée est générée à partir de quatre origines :

1. L'exécutable de Csound (éventuellement avec options). Une seule possibilité est choisie parmi les trois qui suivent (la dernière à la plus haute priorité) :
  - une valeur par défaut
  - la valeur d'une variable d'environnement de CSOUND
  - des variables d'environnement avec un nom de la forme CSOUND\_x où x est une lettre majuscule choisie parmi les caractères de la chaîne -OPTIONS. Ainsi, si l'option -dcb est utilisée, et si les variables d'environnement CSOUND\_B et CSOUND\_C sont définies, la valeur de CSOUND\_B sera effective.

2. N'importe quel nombre de listes d'option, ajoutées dans l'ordre suivant :

- soit quelques valeurs par défaut, soit la valeur de la variable d'environnement CSFLAGS si elle est définie.
- des variables d'environnement avec un nom de la forme CSFLAGS\_x où x est une lettre majuscule choisie parmi les caractères de la chaîne -OPTIONS. Ainsi, si l'option -dcbA est utilisée, et si les variables d'environnement CSFLAGS\_A et CSFLAGS\_C sont définies par '-M 1 -o dac' et '-m231 -H0', respectivement, la chaîne '-m231 -H0 -M 1 -o dac' sera ajoutée.

3. Les options explicites de [OPTIONS DE CSOUND ... ].

4. Toutes les options et les noms de fichiers générés à partir de <nom>.



## NOTE

Les options entre apostrophes qui contiennent des espaces sont autorisées.

## Exemples

Avec les variables d'environnement suivantes :

```
CSOUND      = csoundfltk.exe -W
CSOUND_D    = csound64.exe -J
CSOUND_R    = csoundfltk.exe -h

CSFLAGS     = -d -m135 -H1 -s
CSFLAGS_D   = -f
CSFLAGS_R   = -m0 -H0 -o dac1 -M "MIDI Yoke NT: 1" -b 200 -B 6000
```

Et un répertoire qui contient :

```
foo.orc      piano.csd
foo.sco      piano.mid
im.csd       piano2.mid
ImproSculpt2_share.csd foobar.csd
```

Les commandes suivantes s'exécuteront comme il est montré :

```
cs foo      => csoundfltk.exe -W -d -m135 -H1 -s -o foo.wav \
foo.orc foo.sco

cs foob     => csoundfltk.exe -W -d -m135 -H1 -s          \
-o foobar.wav foobar.csd

cs -r imp -i adc => csoundfltk.exe -h -d -m135 -H1 -s -m0 -H0 \
-o dac1 -M "MIDI Yoke NT: 1" \
-b 200 -B 6000 -i adc \
ImproSculpt2_share.csd

cs -d im     => csound64.exe -J -d -m135 -H1 -s -f -o im.sf \
im.csd

cs piano    => csoundfltk.exe -W -d -m135 -H1 -s          \
-F piano.mid -o piano.wav \
piano.csd

cs piano2    => csoundfltk.exe -W -d -m135 -H1 -s          \
-F piano2.mid -o piano2.wav \
piano.csd
```

## **Crédits**

Auteur : Istvan Varga

Janvier 2003



## csb64enc

csb64enc — Convertit un fichier binaire en un fichier texte encodé en Base64.

### Description

L'utilitaire *csb64enc* génère un fichier texte encodé en Base64 à partir d'un fichier binaire, tel qu'un fichier MIDI standard (.mid) ou n'importe quel type de fichier audio. Il est utile pour convertir un fichier dans le format accepté par la section *<CsFileB>* d'un fichier csd, pour y inclure le fichier converti.

### Syntaxe

```
csb64enc [OPTIONS ... ] fichier1 [ fichier2 [ ... ]]
```

### Initialisation

Options :

- - *w n* = fixe la largeur de ligne du fichier de sortie à *n* (par défaut : 72)
- - *o nomfic* = nom du fichier de sortie (par défaut : stdout)

### Exemples

```
csb64enc -w 78 -o fichier.txt fichier.mid
```

La commande produit un fichier texte encodé en Base64 à partir d'un fichier MIDI standard, *fichier.mid*. Ce fichier peut maintenant être collé dans la section *<CsFileB>* d'un fichier csd.

### Voir aussi

*makecsd*

### Crédits

Auteur : Istvan Varga

Janvier 2003

## envext

envext — Extrait l'enveloppe d'un fichier son vers un fichier texte.

### Syntaxe

```
envext [-options] fichierson  
csound -U envext [-options] fichierson
```

### Initialisation

*fichierson* - Nom du fichier son en entrée.

Les options suivantes sont disponibles pour *envext*. (Les valeurs par défaut sont mises entre parenthèses) :

-o *nomfic* Nom du fichier de sortie (newenv)  
-w *taille* (en secondes) de la fenêtre d'analyse (0.25)

L'utilitaire *envext* génère un fichier texte contenant des paires de temps et d'amplitude en trouvant les pics absolus dans chaque fenêtre.

### Exemples

En tapant la commande (depuis le répertoire manual-fr) :

```
csound -U envext examples/mary.wav
```

on obtiendra un fichier texte contenant :

```
0.000 0.000  
0.000 0.000  
0.250 0.000  
0.500 0.000  
0.750 0.000  
1.249 0.170  
1.499 0.269  
1.530 0.307  
1.872 0.263  
2.056 0.304  
2.294 0.241  
2.570 0.216  
2.761 0.178  
3.077 0.011  
3.251 0.001  
3.500 0.000
```

qui montre le temps pour le pic d'amplitude dans chaque fenêtre mesurée.

### Crédits

Auteur : John ffitch

1995

## extractor

extractor — Extrait une section audio d'un fichier audio.

### Description

Extrait une section audio, par temps ou échantillon, d'un fichier son existant.

### Syntaxe

```
extractor [OPTIONS ... ] fichierentree
```

### Initialisation

Options :

- *-S entier* = Démarre l'extraction à l'échantillon dont le numéro est donné.
- *-Z entier* = Termine l'extraction à l'échantillon dont le numéro est donné.
- *-Q entier* = Extrait le nombre donné d'échantillons.
- *-T fpnum* = Démarre l'extraction au temps donné en secondes.
- *-E fpnum* = Termine l'extraction au temps donné en secondes.
- *-D fpnum* = Extrait la durée donnée en secondes.
- *-R* = Réécrit continuellement l'en-tête lors de l'écriture du fichier son (WAV/AIFF).
- *-H entier* = Montre une "pulsation" pour indiquer la progression, dans le style 1, 2 ou 3.
- *-N* = Signal d'alerte (habituellement la cloche système) à la fin.
- *-v* = Mode verbeux.
- *-o nomfic* = Nom du fichier de sortie (par défaut : test.wav)

### Exemples

Les valeurs par défaut sont :

```
extractor -S 0 -Z fin-du-fichier -o test
```

Par exemple

```
extractor -S 10234 -D 2.13 in.aiff -o out.wav
```

Cela crée un nouveau fichier son extrait à partie de l'échantillon 10234 et durant 2,13 secondes.

### Crédits

Auteur : John ffitch

1994

## makecsd

makecsd — Crée un fichier CSD à partir des fichiers spécifiés en entrée.

### Description

Crée un fichier CSD à partir des fichiers spécifiés en entrée. Le premier fichier d'entrée qui a une extension .orc (la casse n'est pas significative) est mis dans la section <CsInstruments>, et le premier fichier d'entrée qui a une extension .sco devient <CsScore>. Tous les fichiers restants sont encodés en Base64 et ajoutés dans des balises <CsFileB>. Une section <CsOptions> vide est toujours ajoutée.

Un filtrage du texte est effectué sur les fichiers d'orchestre et de partition :

- les caractères de nouvelle ligne sont convertis dans le format natif du système sur lequel *makecsd* est exécuté.
- les lignes vides sont enlevées du début et de la fin des fichiers.
- tous les espaces restant en fin de ligne sont supprimés.
- en option, les tabulations peuvent être développées en espaces avec une taille de tabulation spécifiée par l'utilisateur.
- en option, un fichier MIDI peut être inclus.
- en option, une licence peut être spécifiée soit comme un fichier soit comme une licence commune.
- si on veut utiliser un processeur de partition, on peut l'indiquer pour la section <CsScore>.

### Syntaxe

```
makecsd [OPTIONS ... ] fichier1 [ fichier2 [ ... ]]
```

### Initialisation

Options :

- - *t n* = développe les tabulations en espaces en utilisant une taille de tabulation égale à *n* (désactivé par défaut). Ceci s'applique seulement à l'orchestre et à la partition.
- - *w n* = fixe la largeur de ligne Base64 à *n* (par défaut : 72). Note : l'orchestre et la partition ne sont pas concernés.
- - *o nomfic* = nom du fichier de sortie (par défaut : stdout)
- - *m nomfic* = nom d'un fichier MIDI à inclure (aucun par défaut)
- - *b nomprog* = spécifie le programme pour traiter la partition (aucun par défaut)
- - *L nomfic* = nom du fichier contenant le texte de la licence (aucun par défaut)
- - *l entier* = spécifie une licence standard (aucune par défaut). Les valeurs utilisables sont :
  - 0 : Tous droits réservés
  - 1 : CC BY-NC-ND
  - 2 : CC BY-NC-SA

- 3 : CC BY-NC
- 4 : CC BY-ND
- 5 : CC BY-SA
- 6 : CC BY
- 7 : Sous licence BSD

## Exemples

```
makecsd -t 6 -w 78 -o fichier.csd fichier.mid fichier.orc fichier.sco sample.aif
```

Crée un fichier CSD à partir de fichier.orc et de fichier.sco (les tabulations sont développées en espaces sachant qu'une tabulation vaut 6 caractères), et fichier.mid et sample.aif sont ajoutés dans des balises <CsFileB> contenant les données encodées en Base64 avec une largeur de ligne de 78 caractères. Le fichier de sortie est fichier.csd.

## Crédits

Auteur : Istvan Varga

Janvier 2003

Auteur : John ffitich

Février 2011

Les options pour le MIDI, le traitement de partition et la licence ont été ajoutées dans la version 5.14

## mixer

mixer — Mélange ensemble plusieurs fichiers son.

### Description

Mélange ensemble plusieurs fichiers son, démarrant à des temps différents et avec une sélection individuelle des canaux dans les fichiers d'entrée.

### Syntaxe

```
mixer [OPTIONS ... ] fichier [[OPTIONS... ] fichier] ...
```

### Initialisation

Options :

- **-A** = Génère un fichier de sortie en AIFF.
- **-W** = Génère un fichier de sortie en WAV.
- **-h** = Génère un fichier de sortie sans en-tête.
- **-c** = Génère des échantillons en caractères signés sur 8 bit.
- **-a** = Génère des échantillons alaw.
- **-u** = Génère des échantillons ulaw.
- **-s** = Génère des échantillons en entiers courts.
- **-l** = Génère des échantillons en entiers longs (32 bit).
- **-f** = Génère des échantillons en virgule flottante.
- **-F arg** = Spécifie le gain à appliquer au fichier d'entrée qui suit. Si arg est un nombre en virgule flottante ce gain est appliqué uniformément à l'entrée. Alternativement ça peut être un nom de fichier qui spécifie un fichier de points charnière pour varier le gain sur différentes périodes.
- **-S entier** = Indique à partir de quel échantillon commencer le mixage dans le fichier d'entrée suivant.
- **-T fpnum** = Indique à quel date (en secondes) commencer le mixage dans le fichier d'entrée suivant.
- **-1** = Mixer le canal 1 du fichier son suivant.
- **-2** = Mixer le canal 2 du fichier son suivant.
- **-3** = Mixer le canal 3 du fichier son suivant.
- **-4** = Mixer le canal 4 du fichier son suivant.
- **-^ entx enty** = Mixer le canal x du fichier son suivant vers le canal y dans le fichier de sortie.
- **-v** = Mode verbeux.
- **-R** = Réécrit continuellement l'en-tête lors des opérations d'écriture du fichier son (WAV/AIFF)
- **-H entier** = Montre une "pulsation" pour indiquer la progression, dans le style 1, 2 ou 3.

- *-N* = Alerte (habituellement la cloche système) lorsque le mixage est fini.
- *-o nomfic* = nom du fichier de sortie (par défaut : test.wav)

## Exemples

Les valeurs par défaut sont :

```
mixer -s -otest -F 1.0 -S 0
```

Par exemple

```
mixer -F 0.96 in1.wav -S 300 -2 in2.aiff -S 300 -^4 1 in3.wav -o out.wav
```

Cela crée un nouveau fichier son avec un gain constant de 0,96 pour in1.wav, le second canal de in2.aiff mixé après 300 échantillons et le canal 4 de in3.wav sorti comme le canal 1 après 300 échantillons.

## Crédits

Auteur : John ffitich

1994

## scale

scale — Calibre l'amplitude d'un fichier son.

### Description

Prend un fichier son et le calibre en appliquant un gain, constant ou variable. L'échelle peut être comme un multiplicateur, un maximum ou un pourcentage de 0dB.

### Syntaxe

```
scale [OPTIONS ... ] fichier
```

### Initialisation

Options :

- *-A* = Génère un fichier de sortie AIFF.
- *-W* = Génère un fichier de sortie WAV.
- *-h* = Génère un fichier de sortie sans en-tête.
- *-c* = Génère des échantillons en caractères signés sur 8 bit.
- *-a* = Génère des échantillons alaw.
- *-u* = Génère des échantillons ulaw.
- *-s* = Génère des échantillons en entiers courts.
- *-l* = Génère des échantillons en entiers longs (32 bit)
- *-f* = Génère des échantillons en virgule flottante.
- *-F arg* = Spécifie le gain à appliquer. Si *arg* est un nombre en virgule flottante ce gain est appliqué uniformément à l'entrée. Alternativement ça peut être un nom de fichier qui spécifie un fichier de points charnière pour varier le gain sur différentes périodes.
- *-M fnum* = Calibre l'entrée de façon telle que la valeur absolue du déplacement maximum soit la valeur donnée.
- *-P fnum* = Calibre l'entrée de façon telle que la valeur absolue du déplacement maximum soit le pourcentage donné de 0dB.
- *-R* = Réécrit continuellement l'en-tête pendant l'écriture du fichier son (WAV/AIFF).
- *-H entier* = Montre une "pulsation" pour indiquer la progression, dans le style 1, 2 ou 3.
- *-N* = Alerte (habituellement la cloche système) lorsque c'est fini.
- *-o nomfic* = nom du fichier de sortie (par défaut : test.wav)

### Exemples

```
scale -s -W -F 0.96 -o out.wav sound.wav
```

Ceci crée un nouveau fichier son avec un gain constant de 0,96. C'est particulièrement utile si le fichier d'entrée est en format virgule flottante.



## Crédits

Auteur : John ffitch

1994

## mkdb

mkdb — Sort un catalogue des modules d'opcodes et de greffons.

### Description

Sort un catalogue des modules d'opcodes et de greffons, classés par nom d'opcode ou par module.

### Syntaxe

```
mkdb [-m] [base_directory]
```

### Crédits

Auteur : John ffitch

Août 2011

Nouveau dans la version 5.14

### Crédits

Dan Ellis

MIT Media Lab

Cambridge, Massachussetts

---

# Cscore

*Cscore* est une API (interface de programmation d'application) pour générer et manipuler des fichiers de partition numérique. Elle fait partie de l'API plus grande de Csound et elle comprend un certain nombre de fonctions appelables depuis un programme écrit par l'utilisateur en langage C. *Cscore* peut être invoquée comme un préprocesseur de partition autonome ou comme élément d'une exécution de Csound en incluant l'option -C dans ses arguments :

```
cscore [fichier_partition_entree] [> fichier_partition_sortie]
```

(où *cscore* est le nom du programme que vous avez écrit), ou

```
csound [-C] [autresoptions] [nomorch] [nompartition]
```

Les fonctions de l'API disponibles augmentent la bibliothèque de fonctions du langage C ; elles peuvent lire des fichiers de partition numérique standard ou pré-triée, modifier et étendre les données de différentes manières, et ensuite les rendre disponibles pour une exécution par un orchestre de Csound.

Le programme écrit par l'utilisateur dans le langage C est compilé et lié à la bibliothèque de Csound (ou au programme de ligne de commande *csound*) par l'utilisateur. Il n'est pas indispensable de bien connaître le langage C pour écrire ce programme, car les appels de fonction ont une syntaxe simple, et sont suffisamment puissants pour faire la plus grande partie du travail compliqué. C pourra apporter plus de puissance par la suite selon les besoins.

Les sections suivantes expliquent toutes les étapes de l'utilisation de *Cscore* :

- *Evènements, Listes et Opérations* - Explique la syntaxe des fonctions de *Cscore* et les structures de données.
- *Ecrire un Programme de Contrôle Cscore* - Montre par l'exemple comment écrire votre propre programme de contrôle.
- *Compiler un Programme Cscore* - Décrit les étapes de la compilation et de l'édition des liens avec la bibliothèque de Csound.
- *Exemples Plus Avancés* - Traite de questions avancées comme plusieurs partitions en entrée et les détails de l'exécution de *Cscore* au sein d'une exécution de Csound.

## Evènements, listes et opérations

Un évènement dans *Cscore* est équivalent à une instruction d'une *partition numérique standard* ou d'une partition résolue en temps (le format dans lequel Csound écrit une partition triée -- consultez n'importe quel fichier *score.srt*), et il est stocké en interne en format de temps résolu. Il est important de noter que lorsque *Cscore* est utilisé en mode autonome, il est incapable de comprendre les « commodités » non numériques que Csound permet dans le format de partition en entrée. C'est pourquoi, les partitions utilisant des fonctionnalités telles que le report (carry), les rampes, les expressions et autres devront être triées au préalable avec l'utilitaire *scsort* ou bien utilisées avec un exécutable *Csound* modifié contenant le programme *Cscore* de l'utilisateur. Les opcodes de partition avec des argument macros (r, m, n, and { }) ne sont pas interprétés.

Les évènements de partition sont lus à partir d'un fichier de partition existant et stockés chacun dans une structure C. Les principaux composants de ces structures sont un opcode et un tableau de valeurs de p-

champs. *Cscore* gère la lecture des événements et leur mise en mémoire pour vous. Le format de la structure commence comme suit :

```
typedef struct {
    CSHDR h;          /* en-tête pour la gestion de l'espace */
    char *strarg;      /* adresse d'un argument chaîne facultatif */
    char op;           /* opcode-t, w, f, i, a, s ou e */
    short pcnt;
    MYFLT p2orig;      /* p2, p3 non résolus */
    MYFLT p3orig;
    MYFLT p[1];        /* tableau des p-champs p0, p1, p2 ... */
} EVENT;
```

MYFLT est l'un des types *C float* ou *double* selon la manière dont votre copie de la bibliothèque de Csound a été compilée. Vous avez juste à déclarer les variables en virgule flottante de votre programme avec le type MYFLT pour être compatible.

Toute fonction de *Cscore* qui crée, lit ou copie un événement retournera un pointeur sur la structure dans laquelle les données de l'événement sont stockées. Ce pointeur d'événement peut être utilisé pour accéder aux composants de la structure, de la forme *e->op* ou *e->p[n]*. Chaque événement nouvellement stocké provoquera la création d'un nouveau pointeur, et une séquence de nouveaux événements générera une séquence de pointeurs distincts qu'il faudra stocker. Les groupes de pointeurs d'événement sont stockés dans une liste d'événements qui a sa propre structure :

```
typedef struct {
    CSHDR h;
    int nslots;        /* nombre maximal d'événements dans cette liste */
    int nevents;       /* nombre d'événements présents */
    EVENT *e[1];       /* tableau de pointeurs d'événement e0, e1, e2.. */
} EVLIST;
```

Toute fonction qui crée ou modifie une liste retournera un pointeur sur la nouvelle liste. Ce pointeur de liste peut être utilisé pour accéder à ses composants pointeurs d'événement, de la forme *a->e[n]*. Les pointeurs d'événement et les pointeurs de liste sont ainsi les outils de base pour manipuler les données d'un fichier de partition. Les pointeurs et les listes de pointeurs peuvent être copiés et réordonnés sans modifier les valeurs des données auxquelles ils font référence. Cela signifie que l'on peut copier et manipuler les notes et les phrases depuis un niveau de contrôle élevé. Alternativement, les données d'un événement ou d'un groupe d'événements peuvent être modifiées sans changer les pointeurs d'événement ou de liste. Les fonctions de l'API *Cscore* permettent de créer et de manipuler des partitions de cette manière.

Avec Csound 5, les noms de toutes les fonctions de l'API *Cscore* ont été changés pour être plus explicites. De plus, chaque fonction nécessite maintenant un pointeur sur un objet CSOUND en premier argument. La structure de l'objet CSOUND n'a pas d'importance (en fait il ne peut pas être modifié dans un programme utilisateur). Le moyen d'obtenir ce pointeur sur un objet CSOUND sera montré dans la section suivante. Les fonctions de *Cscore* et ses structures de données sont déclarées dans le fichier d'en-tête *cscore.h* que vous devez inclure dans le code de votre programme avant leur utilisation.

Les noms des fonctions de *Cscore* spécifient si elles opèrent sur des événements ou sur des listes d'événements. Dans le sommaire suivant des appels de fonction disponibles, on utilise quelques conventions de nommage :

```
Le symbole cs est un pointeur vers un objet CSOUND (CSOUND *);
Les symboles e, f sont des pointeurs sur des événements (notes);
Les symboles a, b sont des pointeurs sur des listes (arrays) de tels événements;
Le symbole n est un paramètre entier de type int;
"..." indique un paramètre chaîne (soit une constante soit une variable de type char *);
Le symbole fp est un pointeur sur un fichier (FILE *) en flot d'entrée de partition;
```

syntaxe d'appel	description
-----	-----
/* Fonctions pour travailler avec des évènements */	
e = cscoreCreateEvent(cs, n);	crée un évènement vide avec n pchamps
e = cscoreDefineEvent(cs, "...");	définit un évènement par la chaîne de caractères ...
e = cscoreCopyEvent(cs, f);	fait une nouvelle copie de l'évènement f
e = cscoreGetEvent(cs);	lit l'évènement suivant dans le fichier de partition en
cscorePutEvent(cs, e);	écrit l'évènement e dans le fichier de partition en sor
cscorePutString(cs, "...");	écrit l'évènement défini par la chaîne dans la partiti
	en sortie
/* Fonctions pour travailler avec des listes d'évènements */	
a = cscoreListCreate(cs, n);	crée une liste d'évènements vide avec n emplacements
a = cscoreListAppendEvent(cs, a, e);	ajoute l'évènement e à la fin de la liste a
a = cscoreListAppendStringEvent(cs, a, "...");	ajoute l'évènement défini par la chaîne à la liste a
a = cscoreListCopy(cs, b);	copie la liste b (mais pas les évènements)
a = cscoreListCopyEvents(cs, b);	copie les évènements de b, en créant une nouvelle liste
a = cscoreListGetSection(cs);	lit tous les évènements de la partition en entrée, jusqu
	prochain s ou e
a = cscoreListGetNext(cs, nbeats);	lit les prochaines nbeats pulsations de la partition en
	(nbeats est un MYFLT)
a = cscoreListGetUntil(cs, beatno);	lit tous les évènements de la partition en entrée jusqu
	pulsation beatno (MYFLT)
a = cscoreListSeparateF(cs, b);	sépare les instructions f de la liste b vers la liste a
a = cscoreListSeparateTWF(cs, b);	sépare les instructions t,w & f de la liste b vers la l
a = cscoreListAppendList(cs, a, b);	ajoute la liste b à la liste a
a = cscoreListConcatenate(cs, a, b);	concaténation des listes a et b (identique au précédent)
cscoreListSort(cs, a);	trie la liste a en ordre chronologique selon p[2]
n = cscoreListCount(cs, a);	retourne le nombre d'évènements dans la liste a
a = cscoreListExtractInstruments(cs, b, "...");	extraît les notes des instruments ... (pas de nouveaux
	évènements)
a = cscoreListExtractTime(cs, b, from, to);	extraît les notes d'une période de temps, en créant de
	nouveaux évènements (from et to sont des MYFLT)
cscoreListPut(cs, a);	écrit les évènements de la liste a dans le fichier de p
	sortie
cscoreListPlay(cs, a);	envoie les évènements de la liste a vers l'orchestre de
	une exécution immédiate (ou les imprime s'il n'y a pas
/* Fonctions pour réclamer de la mémoire */	
cscoreFreeEvent(cs, e);	libère l'espace de l'évènement e
cscoreListFree(cs, a);	libère l'espace de la liste a (mais pas les évènements)
cscoreListFreeEvents(cs, a);	libère les évènements de la liste a, et l'espace de la
/* Fonctions pour travailler avec plusieurs fichiers de partition en entrée */	
fp = cscoreFileGetCurrent(cs);	récupère le pointeur du fichier de partition en entrée
	actif (au départ trouve le pointeur du fichier de par
	entrée de la ligne de commande)
fp = cscoreFileOpen(cs, "filename");	ouvre un autre fichier de partition en entrée (5 au max
cscoreFileSetCurrent(cs, fp);	fait de fp le pointeur sur le fichier de partition
	actuellement actif
cscoreFileClose(cs, fp);	ferme le fichier de partition en relation avec FILE *fp

Sous Csound 4, les noms des fonctions et leurs paramètres étaient les suivants :

syntaxe d'appel	description
-----	-----
e = createv(n);	crée un évènement vide avec n pchamps
e = defev("...");	définit un évènement par la chaîne de caractères ...
e = copyev(f);	fait une nouvelle copie de l'évènement f
e = getev();	lit l'évènement suivant dans le fichier de partition en entrée
putev(e);	écrit l'évènement e dans le fichier de partition en sortie
putstr("...");	écrit l'évènement défini par la chaîne dans la partition en sortie
a = lcreat(n);	crée une liste d'évènements vide avec n emplacements
int n;	
a = lappev(a,e);	ajoute l'évènement e à la fin de la liste a
a = lappstrev(a,"...");	ajoute l'évènement défini par la chaîne à la liste a

<code>a = lcopy(b);</code>	copie la liste b (mais pas les évènements)
<code>a = lcopyev(b);</code>	copie les évènements de b, en créant une nouvelle liste
<code>a = lget();</code>	lit tous les évènements de la partition en entrée, jusqu'au prochain s ou e
<code>a = lgetnext(nbeats);</code>	lit les prochaines nbeats pulsations de la partition en entrée
<code>float nbeats;</code>	
<code>a = lgetuntil(beatno);</code>	lit tous les évènements de la partition en entrée jusqu'à la pulsation beatno
<code>float beatno;</code>	
<code>a = lsepf(b);</code>	sépare les instructions f de la liste b vers la liste a
<code>a = lseptwf(b);</code>	sépare les instructions t,w & f de la liste b vers la liste a
<code>a = lcat(a,b);</code>	concaténation (ajout) de la liste b à la liste a
<code>lsort(a);</code>	trie la liste a en ordre chronologique selon p[2]
<code>a = lxins(b,"...");</code>	extraite les notes des instruments ... (pas de nouveaux évènements)
<code>a = lxtimev(b,from,to);</code>	extraite les notes d'une période de temps, en créant de nouveaux évènements
<code>float from, to;</code>	
<code>lput(a);</code>	écrit les évènements de la liste a dans le fichier de partition en sortie
<code>lplay(a);</code>	envoie les évènements de la liste a vers l'orchestre de Csound pour une exécution immédiate (ou les imprime s'il n'y a pas d'orchestre)
<code>relev(e);</code>	libère l'espace de l'évènement e
<code>lrel(a);</code>	libère l'espace de la liste a (mais pas les évènements)
<code>lrelev(a);</code>	libère les évènements de la liste a, et l'espace de la liste
<code>fp = getcurfp();</code>	récupère le pointeur du fichier de partition en entrée actuellement actif (au départ trouve le pointeur du fichier de partition en entrée de la ligne de commande)
<code>fp = filopen("filename");</code>	ouvre un autre fichier de partition en entrée (5 au maximum)
<code>setcurfp(fp);</code>	fait de fp le pointeur sur le fichier de partition actuellement actif
<code>filclose(fp);</code>	ferme le fichier de partition en relation avec FILE *fp

## Ecrire un programme de contrôle Cscore

Le format général d'un programme de contrôle *Cscore* est :

```
#include "cscore.h"
void cscore(CSOUND *cs)
{
    /* DECLARATIONS DES VARIABLES */
    /* CORPS DU PROGRAMME */
}
```

L'instruction *include* définira les structures d'évènement et de liste et toutes les fonctions de l'API *Cscore* pour le programme. Il faut que le nom de la fonction de l'utilisateur soit *cscore* si elle doit être liée avec le programme *main* standard dans *cscormai.c* ou liée comme routine *Cscore* interne pour un exécutable de Csound personnalisé. Cette fonction *cscore()* reçoit un argument de *cscormai.c* ou de Csound -- *CSOUND \*cs* -- qui est un pointeur sur un objet Csound. Le pointeur *cs* doit être passé en premier paramètre à toutes les fonctions de l'API *Cscore* que le programme appelle.

Le programme C suivant lira depuis une *partition numérique standard*, jusqu'à (mais sans l'inclure) la première *instruction s* ou *e*, puis il écrira ces données (inchangées) en sortie.

```
#include "cscore.h"
void cscore(CSOUND *cs)
{
    EVLIST *a;
    a = cscoreListGetSection(cs); /* a est autorisé à pointer sur une liste d'évènements */
    cscoreListPut(cs, a); /* lit les évènements, retourne le pointeur de liste */
    cscorePutString(cs, "e"); /* écrit ces évènements en sortie (inchangés) */
    /* écrit la chaîne e sur la sortie */
}
```

Après l'exécution de *cscoreListGetSection()*, la variable *a* pointe sur une liste d'adresses d'évènements, qui pointent chacune sur un évènement stocké. Nous avons utilisé ce même pointeur pour permettre à une

autre fonction de liste -- *cscoreListPut()* -- d'accéder à tous les événements qui ont été lus et de les écrire en sortie. Si nous définissons maintenant un autre symbole *e* comme pointeur d'évènement, alors l'instruction

```
e = a->e[4];
```

lui affectera le contenu du 4ème emplacement de la structure *EVLIST*, *a*. Ce contenu est un pointeur sur un évènement, qui comprend lui-même un tableau de valeurs de champs de paramètre. Ainsi le terme *e->p[5]* signifiera la valeur du champ de paramètre 5 du 4ème évènement dans la *EVLIST* dénotée par *a*. Le programme ci-dessous multipliera la valeur de ce *p-champ* par 2 avant de l'écrire en sortie.

```
#include "cscore.h"
void cscore(CSOUND *cs)
{
    EVENT *e;                /* un pointeur sur un évènement */
    EVLIST *a;
    a = cscoreListGetSection(cs); /* lit une partition comme une liste d'évènements */
    e = a->e[4];              /* pointe sur l'évènement 4 dans la liste a */
    e->p[5] *= 2;              /* trouve le p-champ 5, multiplie sa valeur par 2 */
    cscoreListPut(cs, a);      /* écrit en sortie la liste d'évènements */
    cscorePutString(cs, "e");  /* ajoute une instruction de "fin de partition" */
}
```

Considérez maintenant la partition suivante, dans laquelle *p[5]* contient la fréquence en Hz.

```
f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
e
```

Si cette partition est donnée au programme principal précédent, la sortie résultante ressemblera à ceci :

```
f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
i 1 1 3 0 440 10000
i 1 4 3 0 512 10000      ; p[5] est devenu 512 au lieu de 256.
i 1 7 3 0 880 10000
e
```

Notez que le 4ème évènement est en fait la seconde note de la partition. Jusqu'ici nous n'avons pas fait de distinction entre les notes et les tables de fonction mises en place dans une partition numérique. Les deux peuvent être classées comme évènement. Notez aussi que notre 4ème évènement a été stocké dans le champ *e[4]* de la structure. Pour être compatible avec la notation des *p-champs* de Csound, nous ignorerons *p[0]* et *e[0]* dans les structures d'évènement et de liste, en stockant *p1* dans *p[1]*, l'évènement 1 dans *e[1]*, etc. Les fonctions de *Cscore* adoptent toutes cette convention.

Pour étendre l'exemple ci-dessus, nous pourrions décider d'utiliser les mêmes pointeurs *a* et *e* pour examiner chacun des évènements dans la liste. Noter que *e* n'a pas été fixé au nombre 4, mais au contenu du 4ème emplacement de la liste. Pour inspecter le *p5* de l'évènement précédent dans la liste, nous n'avons qu'à redéfinir *e* avec l'affectation

```
e = a->e[3];
```

et référencer le 5ème emplacement du tableau de *p-champs* avec l'expression

```
e->p[5]
```

Plus généralement, nous pouvons utiliser une variable entière comme indice du tableau *e[]*, et accéder séquentiellement à chaque évènement en utilisant une boucle et en incrémentant l'indice. Le nombre d'évènements stockés dans une *EVLIST* est contenu dans le membre *nevents* de la structure.

```
int index;    /* démarre avec e[1] car e[0] n'est pas utilisé */
for (index = 1; index <= a->nevents; index++)
{
    e = a->e[index];
    /* faire quelque chose avec e */
}
```

L'exemple ci-dessus démarre avec *e[1]* et augmente l'indice à chaque passage dans la boucle (*index++*) jusqu'à ce qu'il soit plus grand que *a->nevents*, l'indice du dernier évènement dans la liste. Les instructions à l'intérieur de la boucle *for* sont exécutées une dernière fois quand *index* égale *a->nevents*.

Dans le programme suivant nous utiliserons la même partition en entrée. Cette fois nous séparerons les instructions de *fiable* des instructions de *note*. Nous écrirons ensuite en sortie les trois évènements de note stockés dans la liste *a*, puis nous créerons une seconde section de partition constituée de l'ensemble de hauteurs original et d'une version transposée de celui-ci. Cela apportera un doublement à l'octave.

Ici, notre indice dans le tableau est *n* et il est incrémenté dans un bloc *for* qui boucle *nevents* fois, ce qui permet d'appliquer une instruction au même *p-champ* des évènements successifs.

```
#include "cscore.h"
void cscore(CSOUND *cs)
{
    EVENT *e, *f;
    EVLIST *a, *b;
    int n;

    a = cscoreListGetSection(cs);          /* lit la partition dans la liste d'évènements "a" */
    b = cscoreListSeparateF(cs, a);        /* sépare les instructions f */
    cscoreListPut(cs, b);                  /* écrit les instructions f dans la partition en sortie */
    e = cscoreDefineEvent(cs, "t 0 120"); /* définit un évènement pour l'instruction de tempo */
    cscorePutEvent(cs, e);                 /* écrit l'instruction de tempo dans la partition */
    cscoreListPut(cs, a);                  /* écrit les notes */
    cscorePutString(cs, "s");              /* fin de section */
    cscorePutEvent(cs, e);                 /* écrit l'instruction de tempo encore une fois */
    b = cscoreListCopyEvents(cs, a);        /* fait une copie des notes dans "a" */
    for (n = 1; n <= b->nevents; n++)      /* répète les lignes suivantes nevents fois : */
    {
        f = b->e[n];
        f->p[5] *= 0.5;                    /* transpose la hauteur d'une octave vers le bas */
    }
    a = cscoreListAppendList(cs, a, b);    /* ajoute ces notes aux hauteurs originales */
    cscoreListPut(cs, a);
    cscorePutString(cs, "e");
}
```

La sortie de ce programme est :

```
f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
t 0 120
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
s
t 0 120
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
```



```
i 1 7 3 0 880 10000
i 1 1 3 0 220 10000
i 1 4 3 0 128 10000
i 1 7 3 0 440 10000
e
```

Si la sortie est écrite dans un fichier, le fait que les événements ne soient pas ordonnés n'est pas un problème. La sortie est écrite dans un fichier (ou sur la sortie standard) chaque fois que la fonction *cscoreListPut()* est utilisée. Cependant, si ce programme était appelé durant une exécution de Csound et que la fonction *cscoreListPlay()* était remplacée par *cscoreListPut()*, alors les événements seraient envoyés à l'orchestre au lieu du fichier et il faudrait qu'ils soient préalablement triés en appelant la fonction *cscoreListSort()*. Les détails de la sortie de la partition et de son exécution quand on utilise *Cscore* depuis Csound sont décrits dans la section suivante.

Ensuite nous étendons le programme ci-dessus en utilisant la boucle *for* pour lire *p[5]* et *p[6]*. Dans la partition originale *p[6]* dénote l'amplitude. Pour créer un diminuendo sur l'octave inférieure ajoutée, qui soit indépendant de l'ensemble de notes original, une variable appelée *dim* sera utilisée.

```
#include "cscore.h"
void cscore(CSOUND *cs)
{
    EVENT *e, *f;
    EVLIST *a, *b;
    int n, dim;                                /* déclare deux variables entières */

    a = cscoreListGetSection(cs);
    b = cscoreListSeparateF(cs, a);
    cscoreListPut(cs, b);
    cscoreListFreeEvents(cs, b);
    e = cscoreDefineEvent(cs, "t 0 120");
    cscorePutEvent(cs, e);
    cscoreListPut(cs, a);
    cscorePutString(cs, "s");
    cscorePutEvent(cs, e);                    /* écrit une autre instruction de tempo */
    b = cscoreListCopyEvents(cs, a);
    dim = 0;                                  /* initialise dim à 0 */
    for (n = 1; n <= b->nevents; n++)
    {
        f = b->e[n];
        f->p[6] -= dim;                        /* soustrait la valeur courante de dim */
        f->p[5] *= 0.5;                        /* transpose la hauteur une octave plus bas */
        dim += 2000;                          /* augmente dim pour chaque note */
    }
    a = cscoreListAppendList(cs, a, b);      /* ajoute ces notes aux hauteurs originales */
    cscoreListPut(cs, a);
    cscorePutString(cs, "e");
}
```

En utilisant à nouveau la même partition en entrée, la sortie de ce programme est :

```
f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
t 0 120
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
s
t 0 120
i 1 1 3 0 440 10000      ; Trois notes originales aux pulsations
i 1 4 3 0 256 10000    ; 1, 4 et 7 sans diminuendo.
i 1 7 3 0 880 10000
i 1 1 3 0 220 10000    ; Trois notes transposées une octave plus bas
```

```
i 1 4 3 0 128 8000      ; également aux pulsations 1, 4 et 7
i 1 7 3 0 440 6000      ; avec diminuendo.
e
```

Dans le programme suivant la même séquence de trois notes sera répétée à divers intervalles de temps. La date de début de chaque groupe est déterminée par les valeurs du tableau *cue*. Cette fois le *dim* se produira sur chaque groupe de notes plutôt que sur chaque note. Remarquez la position de l'instruction qui incrémente la variable *dim* en dehors de la boucle *for* intérieure.

```
#include "cscore.h"
int cue[3] = {0,10,17};          /* déclare un tableau de 3 entiers */
void cscore(CSOUND *cs)
{
    EVENT *e, *f;
    EVLIST *a, *b;
    int n, dim, cuecount;        /* déclare la nouvelle variable cuecount */

    a = cscoreListGetSection(cs);
    b = cscoreListSeparateF(cs, a);
    cscoreListPut(cs, b);
    cscoreListFreeEvents(cs, b);
    e = cscoreDefineEvent(cs, "t 0 120");
    cscorePutEvent(cs, e);
    dim = 0;
    for (cuecount = 0; cuecount <= 2; cuecount++) /* les éléments de cue sont numérotés 0, 1, 2 */
    {
        for (n = 1; n <= a->nevents; n++)
        {
            f = a->e[n];
            f->p[6] -= dim;
            f->p[2] += cue[cuecount];          /* ajoute les valeurs de cue */
        }
        printf("; diagnostic: cue = %d\n", cue[cuecount]);
        dim += 2000;
        cscoreListPut(cs, a);
    }
    cscorePutString(cs, "e");
}
```

Ici la boucle *for* intérieure lit les événements de la liste *a* (les notes) et la boucle *for* extérieure lit chaque *répétition* des événements de la liste *a* (les "répliques" du groupe de hauteurs). Ce programme démontre aussi un moyen utile de résolution de problème au moyen de la fonction *printf*. Le *point-virgule* commence la chaîne de caractères pour produire un commentaire dans le fichier de partition résultant. Dans ce cas, la valeur de *cue* est imprimée en sortie pour s'assurer que le programme prend le bon membre du *tableau* au bon moment. Lorsque les données de sortie sont fausses ou que des messages d'erreur sont rencontrés, la fonction *printf* peut aider à identifier le problème.

A partir du même fichier d'entrée, le programme C ci-dessus générera la partition suivante. Pouvez-vous expliquer pourquoi le dernier ensemble de notes ne démarre pas au bon moment et comment corriger le problème ?

```
f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
t 0 120
; diagnostic: cue = 0
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
; diagnostic: cue = 10
i 1 11 3 0 440 8000
i 1 14 3 0 256 8000
i 1 17 3 0 880 8000
```

```
; diagnostic: cue = 17
i 1 28 3 0 440 4000
i 1 31 3 0 256 4000
i 1 34 3 0 880 4000
e
```

## Compiler un programme Cscore

Un programme *Cscore* peut être invoqué comme un *programme autonome* ou comme une partie de Csound placée entre le tri de la partition et son exécution par l'orchestre :

```
cscore [fichier_partition_entrée] [> fichier_partition_sortie]
```

ou

```
csound [-C] [autresoptions] [nomorch] [nompartition]
```

Avant d'essayer de compiler votre propre programme *Cscore*, vous voudrez sans doute obtenir une copie du code source de Csound. Téléchargez la distribution des sources la plus récente pour votre plate-forme ou bien récupérez (check out) une copie du module csound5 depuis le CVS de Sourceforge. Il y a plusieurs fichiers dans les sources qui vous aideront. Il y a dans le répertoire `examples/cscore/` plusieurs exemples de programmes de contrôle *Cscore*, y compris tous les exemples contenus dans ce manuel. Et il y a dans le répertoire `frontends/cscore/` les deux fichiers *cscoremain.c* et *cscore.c*. *cscoremain.c* contient une simple fonction *main* qui réalise toute l'initialisation qu'un programme *Cscore* autonome doit faire avant d'appeler votre fonction de contrôle. Cette « souche » *main* initialise Csound, lit les arguments de la ligne de commande, ouvre les fichiers de partition en entrée et en sortie, et appelle ensuite une fonction *cscore()*. Comme il est décrit ci-dessus, vous êtes chargé d'écrire la fonction *cscore()* et de la fournir dans un autre fichier. Le fichier *frontends/cscore/cscore.c* montre l'exemple le plus simple d'une fonction *cscore()* qui lit une partition de n'importe quelle longueur et l'écrit inchangée sur la sortie.

Ainsi, pour créer un programme autonome, écrivez un programme de contrôle en suivant les indications de la section précédente. Supposons que vous ayez sauvegardé ce programme dans un fichier nommé "*myscore.c*". Vous devez ensuite compiler ce programme et le lier avec la bibliothèque de Csound et *cscoremain.c* pour créer un exécutable, en suivant l'ensemble de directives ci-dessous qui s'applique à votre système d'exploitation. Il sera utile d'avoir une certaine familiarité avec le compilateur C de votre ordinateur car l'information ci-dessous ne peut pas être exhaustive pour tous les systèmes existants.

## Linux et Unix

Les commandes suivantes supposent que vous ayez copié votre fichier *myscore.c* dans le même répertoire que *cscoremain.c*, que vous ayez ouvert un terminal sur ce même répertoire et que vous ayez installé au préalable une distribution binaire de Csound qui aura placé une bibliothèque *libcsound.a* ou *libcsound.so* dans `/usr/local/lib` et les fichiers d'en-tête pour l'API de Csound dans `/usr/local/include/csound`.

Pour la compilation et l'édition de liens, tapez :

```
gcc myscore.c cscoremain.c -o cscore -lcsound -L/usr/local/lib -I/usr/local/include/csound
```

Pour l'exécution (avec envoi des résultats sur la sortie standard), tapez :

```
./cscore test.sco
```

Il est possible que sur certains systèmes Unix le compilateur C soit nommé *cc* ou quelque chose d'autre que *gcc*.

## Windows

Csound est ordinairement compilé sur Windows au moyen de l'environnement MinGW qui fournit GCC -- le même compilateur utilisé sur Linux -- au travers d'un shell de commande (MSYS) à la Unix. Comme les bibliothèques pré-compilées pour Csound sur Windows sont construites de cette manière, vous utiliserez probablement MinGW pour la liaison avec celles-ci. Si vous avez construit Csound en utilisant un autre compilateur, vous serez sans doute capable de construire également *Cscore* avec ce compilateur.

La compilation de programmes *Cscore* autonomes en utilisant MinGW devrait être similaire à la procédure ci-dessus pour Linux avec les chemins de la bibliothèque et des en-têtes changés pour pointer là où Csound est installé sur le système Windows. *(Les contributions plus détaillées sur ces instructions seront les bienvenues car le rédacteur de cet article n'a pas pu tester Cscore sur une machine Windows).*

## OS X

Les commandes suivantes supposent que vous ayez copié votre fichier *myscore.c* dans le même répertoire que *cscoremain.c* et que vous ayez ouvert un terminal dans ce répertoire. De plus, les outils de développement fournis par Apple (incluant le compilateur GCC) doivent être installés sur votre système et vous devez avoir installé une distribution binaire de Csound qui aura placé le framework Csoundlib dans */Library/Frameworks*.

Utilisez cette commande pour la compilation et l'édition de liens. (Il peut y avoir un avertissement sur de "multiples définitions du symbole \_cscore").

```
gcc cscore.c cscoremain.c -o cscore -framework CsoundLib -I/Library/Frameworks/CsoundLib.framework/Headers
```

Pour l'exécution (avec envoi des résultats sur la sortie standard) :

```
./cscore test.sco
```

## MacOS 9

Vous devrez avoir installé CodeWarrior ou un autre environnement de développement sur votre ordinateur (MPW peut fonctionner). Téléchargez la distribution des sources pour OS 9 (elle aura un nom comme *Csound5.05\_OS9\_src.smi.bin*).

Si vous utiliser CodeWarrior, trouvez et ouvrez le fichier de projet "Cscore5.cw8.mcp" dans le répertoire "Csound5.04-OS9-source:macintosh:Csound5Library:". Ce fichier de projet est configuré pour utiliser les fichiers source *cscore.c* et *cscoremain\_MacOS9.c* situés dans l'arborescence des sources csound5 et la librairie partagée Csound5Lib produite lors de la compilation de Csound avec le fichier de projet "Csound5.cw8.mcp". Il vous faut substituer votre propre fichier du programme *Cscore* à la place de *cscore.c* et soit avoir compilé Csound5Lib avant, soit substituer une copie de la bibliothèque dans le projet à partir de la distribution binaire de Csound pour OS 9. Le fichier *cscoremain\_MacOS9.c* contient du code spécialisé pour la configuration de la bibliothèque de console SIOUX de CodeWarrior et permet l'entrée d'arguments de ligne de commande avant le lancement du programme.

Une fois que les fichiers nécessaires sont inclus dans la fenêtre du projet, cliquez sur le bouton "Make" et CodeWarrior produira une application nommée « *Cscore* ». Quand vous lancez cette application, elle affiche d'abord une fenêtre vous permettant de saisir les arguments pour la fonction principale. Vous n'avez qu'à taper le nom de fichier ou le nom de chemin complet de la partition en entrée -- ne tapez pas "cscore". Le fichier d'entrée doit se trouver dans le même répertoire que l'application sinon vous devrez taper un chemin complet ou relatif pour le fichier. La sortie sera affichée dans la fenêtre de console. Vous pouvez utiliser la commande *Save* du menu *File* avant de quitter la console. Alternativement, dans la fenêtre de dialogue de la ligne de commande, vous pouvez choisir de rediriger la sortie dans un fichier en cliquant sur

le bouton *File* sur le côté droit de la fenêtre de dialogue. (Notez que la fenêtre de console ne peut afficher qu'environ 32000 caractères, ce qui rend l'écriture dans un fichier nécessaire pour les grandes partitions).

## Rendre Cscore utilisable depuis Csound

Pour opérer depuis Csound, suivez d'abord les instructions pour compiler Csound (voir *Construire Csound*) qui concernent le système d'exploitation que vous utilisez. Une fois que vous avez réussi à construire un système Csound non modifié, substituez alors votre propre fonction *cscore()* à celle qui se trouve dans le fichier *Top/cscore\_internal.c*, et reconstruisez Csound.

L'exécutable résultant est votre Csound spécial, utilisable comme ci-dessus. L'option *-C* invoquera votre programme *Cscore* après le tri de la partition d'entrée dans « *score.srt* ». Les détails de ce qui se passe lorsque vous lancez Csound avec l'option *-C* flag sont donnés dans la section suivante.

Csound 5 fournit aussi un moyen supplémentaire d'exécuter votre propre programme *Cscore* depuis Csound. En utilisant l'API, une application hôte peut mettre en place une *fonction d'appel en retour (callback)* de *Cscore*, qui est une fonction que Csound appellera à la place de sa fonction interne *cscore()*. L'avantage de cette approche est qu'il n'est pas nécessaire de recompiler la totalité de Csound. Un autre bénéfice est que l'application hôte peut choisir pendant l'exécution la fonction de callback parmi plusieurs fonctions *Cscore*. L'inconvénient est que vous devez écrire une application hôte.

Une approche simple pour utiliser un callback *Cscore* via l'API serait de modifier le programme main standard de Csound -- qui est un hôte simple de Csound -- contenu dans le fichier *frontends/csound/csound\_main.c*. L'ajout d'un appel à *csoundSetCscoreCallback()* après l'appel à *csoundCreate()* mais avant l'appel à *csoundCompile()* devrait faire l'affaire. En recompilant ce fichier et en le liant à une bibliothèque de Csound existante, on obtiendra une version de Csound en ligne de commande qui fonctionne comme celle qui est décrite ci-dessus. N'oubliez pas de taper l'option *-C*.

## Notes au sujet des formats de partition et du comportement de l'exécutable

Comme indiqué précédemment, les fichiers d'entrée de *Cscore* peuvent se trouver dans leur forme originale ou résolue en temps et pré-triée ; cette modalité sera préservée (section par section) lors de la lecture, du traitement et de l'écriture des partitions. Le traitement autonome utilisera le plus souvent des sources non résolues en temps et créera de nouveau fichiers de même forme. Lors du traitement depuis Csound, la partition en entrée arrivera déjà résolue en temps et triée, et pourra ainsi être envoyée directement (normalement section par section) à l'orchestre. Un des avantages de cette façon d'utiliser *Cscore* est que toutes les commodités de syntaxe du langage de partition complet de Csound peuvent être utilisées -- macros, expressions arithmétiques, carry, rampes, etc. -- car la partition passera par les phases "Carry, Tempo, Tri" du traitement avant d'être transmise au programme *Cscore* fourni par l'utilisateur.

Lors du traitement dans Csound, une liste d'évènements peut être transmise à un orchestre de Csound en utilisant *cscoreListPlay()*. Il peut y avoir n'importe quel nombre d'appels de *cscoreListPlay()* dans un programme *Cscore*. Chaque liste ainsi transmise peut-être résolue ou non en temps, mais chaque liste doit être en ordre chronologique strict par rapport à *p2* (soit grâce au pré-traitement de tri soit en utilisant *cscoreListSort()*). S'il n'y a pas de *cscoreListPlay()* dans un module *Cscore* exécuté depuis Csound, tous les évènements écrits en sortie (via *cscorePutEvent()*, *cscorePutString()*, ou *cscoreListPut()*) sont envoyés dans une nouvelle partition dans le répertoire courant nommée « *cscore.out* ». Csound invoque alors à nouveau le tri de partition avant d'envoyer cette nouvelle partition à l'orchestre pour son exécution. La partition de sortie triée finale est écrite dans un fichier nommé « *cscore.srt* ».

Un programme *Cscore* autonome utilisera normalement la commande « put » pour écrire dans son fichier de sortie. Si un programme *Cscore* autonome appelle *cscoreListPlay()*, les évènements ainsi destinés à l'exécution seront envoyés sur la sortie comme s'ils provenaient de *cscoreListPut()*.

Une liste de notes envoyée par *cscoreListPlay()* pour exécution doit être distincte dans le temps des listes de notes suivantes. Aucune fin de note ne doit dépasser la date de début de la liste suivante, car *cscoreListPlay()* complètera chaque liste avant d'attaquer la suivante (comme un marqueur de Section qui ne ré-initialise pas le temps local à zéro). C'est important lorsque l'on utilise *cscoreListGetNext()* ou *cscoreListGetUntil()* pour charger et traiter des segments de partition avant exécution, car ces fonctions pourraient ne lire qu'une partie d'une section non triée.

## Exemples plus avancés

Le programme suivant démontre la lecture à partir de deux fichiers d'entrée différents. L'idée est d'alterner entre deux partitions de 2 sections, et d'écrire les sections entrelacées dans un seul fichier de sortie.

```
#include "cscore.h"                /* CSCORE_SWITCH.C */
cscore(CSOUND* cs)                /* appellable depuis Csound ou comme cscore autonome */
{
    EVLIST *a, *b;
    FILE *fp1, *fp2;               /* deux pointeurs sur des flots de fichier de partition */
    fp1 = cscoreFileGetCurrent(cs); /* la partition de la ligne de commande */
    fp2 = cscoreFileOpen(cs, "score2.srt"); /* une partition supplémentaire */
    a = cscoreListGetSection(cs);    /* lit une section de la partition 1 */
    cscoreListPut(cs, a);            /* l'écrit en sortie telle quelle */
    cscorePutString(cs, "s");
    cscoreFileSetCurrent(cs, fp2);
    b = cscoreListGetSection(cs);    /* lit une section de la partition 2 */
    cscoreListPut(cs, b);            /* l'écrit en sortie telle quelle */
    cscorePutString(cs, "s");
    cscoreListFreeEvents(cs, a);     /* facultatif, pour libérer de l'espace */
    cscoreListFreeEvents(cs, b);
    cscoreFileSetCurrent(cs, fp1);
    a = cscoreListGetSection(cs);    /* lit la section suivante de la partition 1 */
    cscoreListPut(cs, a);            /* l'écrit en sortie */
    cscorePutString(cs, "s");
    cscoreFileSetCurrent(cs, fp2);
    b = cscoreListGetSection(cs);    /* lit la section suivante de la partition 2 */
    cscoreListPut(cs, b);            /* l'écrit en sortie */
    cscorePutString(cs, "e");
}
```

Finalement, nous montrons comment prendre un fichier de partition littérale, non interprétée et lui insuffler un peu d'expressivité rythmique. La théorie des pulsations métriques liées au compositeur a été étudiée en profondeur par Manfred Clynes, et la suite est dans l'esprit de ce travail. Ici, la stratégie consiste à créer d'abord un *tableau* de nouvelles dates de *début* pour chaque début possible de double croche, puis par indexation dans ce tableau, d'ajuster le début et la durée de chaque note de la partition d'entrée aux dates interprétées. On montre aussi comment un orchestre de Csound peut être invoqué de façon répétitive depuis un générateur de partition pendant l'exécution.

```
#include "cscore.h"                /* CSCORE_PULSE.C */

/* programme pour appliquer une pulsation aux durées interprétées */
/* à une partition existante en 3/4, premiers temps sur 0, 3, 6 ... */

static float four[4] = { 1.05, 0.97, 1.03, 0.95 }; /* largeur de pulsation des 4 */
static float three[3] = { 1.03, 1.05, .92 };        /* largeur de pulsation des 3 */

cscore(CSOUND* cs)                /* Cet exemple doit être appelé depuis Csound */
{
    EVLIST *a, *b;
    EVENT *e, **ep;
```

```
float pulse16[4*4*4*4*3*4];    /* tableau de doubles croches, 3/4, 256 mesures */
float acc16, acc1,inc1, acc3,inc3, acc12,inc12, acc48,inc48, acc192,inc192;
float *p = pulse16;
int  n16, n1, n3, n12, n48, n192;

/* remplit le tableau avec les dates de début de l'interprétation */
for (acc192=0.,n192=0; n192<4; acc192+=192.*inc192,n192++)
    for (acc48=acc192,inc192=four[n192],n48=0; n48<4; acc48+=48.*inc48,n48++)
        for (acc12=acc48,inc48=inc192*four[n48],n12=0;n12<4; acc12+=12.*inc12,n12++)
            for (acc3=acc12,inc12=inc48*four[n12],n3=0; n3<4; acc3+=3.*inc3,n3++)
                for (acc1=acc3,inc3=inc12*four[n3],n1=0; n1<3; acc1+=inc1,n1++)
                    for (acc16=acc1,inc1=inc3*three[n1],n16=0; n16<4; acc16+=.25*inc1*four[n16],n16++)
                        *p++ = acc16;

/* for (p = pulse16, n1 = 48; n1--; p += 4) /* montre les valeurs & les différences */
/*   printf("%g %g %g %g %g %g %g %g\n", *p, *(p+1), *(p+2), *(p+3),
/*   *(p+1)-*p, *(p+2)-*(p+1), *(p+3)-*(p+2), *(p+4)-*(p+3)); */

a = cscoreListGetSection(cs);          /* lit une section de la partition résolue en temps */
b = cscoreListSeparateTWF(cs, a);      /* sépare les instructions de jeu et de fonction */
cscoreListPlay(cs, b);                 /* et les envoie à l'exécution */
a = cscoreListAppendStringEvent(cs, a, "s"); /* ajoute une instruction de section à la liste de notes */
cscoreListPlay(cs, a);                 /* joue la liste de notes sans interprétation */
for (ep = &a->e[1], n1 = a->nevents; n1--; ) { /* maintenant modifie les pulsations */
    e = *ep++;
    if (e->op == 'i') {
        e->p[2] = pulse16[(int)(4. * e->p2orig)];
        e->p[3] = pulse16[(int)(4. * (e->p2orig + e->p3orig))] - e->p[2];
    }
}

cscoreListPlay(cs, a);                 /* maintenant joue la liste modifiée */
}
```

---

# Csbeats

*Csbeats* est un langage de partition alternatif dont le but est de spécifier de simples partitions dans le système standard occidental de notation des hauteurs et des rythmes. On peut invoquer *Csbeats* via le composant *CsScore* d'une partition .csd standard avec *bin="csbeats"* ou comme un programme autonome qui génère une partition numérique standard.

Le programme autonome lit depuis l'entrée standard et écrit sur la sortie standard.

Le langage *csbeats* est très simple, n'ayant que cinq sortes d'instructions dont une seule présente quelque complexité. Généralement, le mot clé de chaque type d'instruction est insensible à la casse. Ainsi "QUIT", "quit", "Quit"... sont tous pareils. Les commentaires s'écrivent dans le format ANSI C89, ou celui de C++ (c'est-à-dire /\* ... \*/ ou bien // jusqu'à la fin de la ligne), ou encore avec le point-virgule de Csound.

- *QUIT*

Provoque la sortie de *csbeats*. Pour des raisons de souplesse la commande *END* est aussi admise pour la même action.

- *BEATS*=entier

Fixe le nombre de pulsations par minute pour la suite de la partition, jusqu'à la fin ou jusqu'à ce qu'il soit réinitialisé. La valeur par défaut est de 60 pulsations par minutes. Le mot *BPS* est aussi accepté à la place de *BEATS*.

- *PERMEASURE*=entier

Fixe le nombre de pulsations dans une mesure. La valeur par défaut est 4.

- *BAR*

Commence une nouvelle mesure.

- *BAR* entier

Commence la mesure dont le numéro est donné.

- *i* entier attributs

Spécifie un évènement de note pour l'instrument numéroté. Les attributs peuvent indiquer une hauteur, une durée, une dynamique, ou la position d'une note sur un temps ou dans une mesure, et ceci dans n'importe quel ordre.

Les hauteurs sont données avec un nom de note conventionnel (notation anglaise) en lettre majuscule éventuellement suivie par un caractère #, x (pour double dièse), b (pour bémol) ou bb (pour double bémol). La note Z est un silence (penser à zzz). Toutes les notes, sauf les silences, doivent être suivies par un numéro d'octave, A4 étant le la du diapason international (440 Hz). Les hauteurs sont transmises à Csound en Hz dans le paramètre p4, et s'expriment en tempérament égal à douze demi-tons.

Les durées sont codées par la première lettre de leur nom anglais en minuscule, *whole* (ronde), *half* (blanche), *quarter* (noire), *eighth* (croche), *sixteenth* (double croche) et *th* (triple croche). A l'exception de *w*, on peut les modifier en leur ajoutant un suffixe :

- *d* ou . Note pointée (allongée de la moitié de sa durée nominale)
- *dd* ou .. Note doublement pointée (allongée des trois-quarts de sa durée nominale)



- *t* Triolet (trois notes pour deux)
- *t*. Triolet de notes pointé
- *q* Quintuplet (cinq notes pour quatre)
- *s* Septuplet (Sept notes pour huit)

On peut ajouter les données entre elles en donnant plus d'une durée. On peut utiliser un signe + à la place d'un espace pour rendre ceci plus intuitif.

Les dynamiques sont écrites en notation conventionnelle, fff, ff, f, mf, mp, p, pp, ppp, pppp. Elles sont transmises à l'instrument dans p5 avec 0 pour fff et un dB en moins pour chaque niveau inférieur. La dynamique par défaut est fortissimo.

On peut spécifier des paramètres supplémentaires (p-champs) dans une instruction d'instrument sous une forme semblable à *p6=42*, qui donnerait la valeur 42 au champ p6 jusqu'à ce qu'il change. Le numéro du p-champ doit être supérieur ou égal à 6 car les 5 premiers champs sont utilisés avec une information définie. Chaque instrument numéroté possède ses propres paramètres supplémentaires.

Si l'un de ces attributs est omis, il est reporté de la note précédente, sa position étant incrémentée depuis la fin de la note précédente.

De plus, un évènement peut être placé dans une mesure particulière avec un attribut m, ou sur un temps particulier avec un attribut b.

On peut coder l'ouverture de la variation Goldberg n°3 de Bach comme ceci :

```
; Bach - Goldberg Variations - Variato 3
; by Brian Baughn 3-14-05
; bbaughn@berklee.net
beats = 120
permeasure = 6
```

```
i101    m1 b1 B4 mp qd+s
i101          C5    s
i101          D5
i101          C5
i101          D5
i101          E5
i101          A4    qd+s
i101          B4    s
i101          C5
i101          B4
i101          C5
i101          D5
```

```
i101    m2 b1 G4    qd
i101          G5    qd+e
i101          A5    s
i101          G5
i101          F#5
i101          G5
i101          A5    e
```

```
i101    m3 b1.5 D5    s
i101          C5
i101          B4
i101          A4
i101          B4    e
i101          C5    s
```

```

i101      B4
i101      A4
i101      B4
i101      G4      e
i101      E5
i101      D5
i101      C5
i101      F#5
i101      A5

i101  m4 b1  B4      q
i101      G5      e
i101      G5      q
i101      F#5      e
i101      Z      e      // Z is a rest (zzzzz..)
i101      e
i101      B5      e
i101      A5      q
i101      D5      e

quit

```

La sortie produite est

```

;;;setting bpm=120.000000
;;;setting permeasure=6
i101 0.000000 0.875000 493.883621 -4
i101 0.875000 0.125000 523.251131 -4
i101 1.000000 0.125000 587.329536 -4
i101 1.125000 0.125000 523.251131 -4
i101 1.250000 0.125000 587.329536 -4
i101 1.375000 0.125000 659.255114 -4
i101 1.500000 0.875000 440.000000 -4
i101 2.375000 0.125000 493.883621 -4
i101 2.500000 0.125000 523.251131 -4
i101 2.625000 0.125000 493.883621 -4
i101 2.750000 0.125000 523.251131 -4
i101 2.875000 0.125000 587.329536 -4
i101 3.000000 0.750000 391.995436 -4
i101 3.750000 1.000000 783.990872 -4
i101 4.750000 0.125000 880.000000 -4
i101 4.875000 0.125000 783.990872 -4
i101 5.000000 0.125000 739.988845 -4
i101 5.125000 0.125000 783.990872 -4
i101 5.250000 0.250000 880.000000 -4
i101 6.250000 0.125000 587.329536 -4
i101 6.375000 0.125000 523.251131 -4
i101 6.500000 0.125000 493.883621 -4
i101 6.625000 0.125000 440.000000 -4
i101 6.750000 0.250000 493.883621 -4
i101 7.000000 0.125000 523.251131 -4
i101 7.125000 0.125000 493.883621 -4
i101 7.250000 0.125000 440.000000 -4
i101 7.375000 0.125000 493.883621 -4
i101 7.500000 0.250000 391.995436 -4
i101 7.750000 0.250000 659.255114 -4
i101 8.000000 0.250000 587.329536 -4
i101 8.250000 0.250000 523.251131 -4
i101 8.500000 0.250000 739.988845 -4
i101 8.750000 0.250000 880.000000 -4
i101 9.000000 0.500000 493.883621 -4
i101 9.500000 0.250000 783.990872 -4
i101 9.750000 0.500000 783.990872 -4
i101 10.250000 0.250000 739.988845 -4

```

```
;;rest at 10.500000 for 0.250000
;;rest at 10.750000 for 0.250000
i101 11.000000 0.250000 987.767243 -4
i101 11.250000 0.500000 880.000000 -4
i101 11.750000 0.250000 587.329536 -4
e
```

## Un exemple complet

Voici un exemple simple du générateur de partition *csbeats*. Il utilise le fichier *csbeats.csd* [examples/csbeats.csd].

### Exemple 1255. Un exemple simple de csbeats.

Voir les sections *Audio en Temps Réel* et *Options de la Ligne de Commande* pour plus d'information sur l'utilisation des options de la ligne de commande.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>

<CsInstruments>
sr      =      44100
nchnls  =      2

gil ftgen 1, 0, 4096, 10, 1
gi2 ftgen 2, 0, 4096, 7, -1, 4096, 1 ; sawtooth
gi3 ftgen 3, 0, 4096, 7, 0, 1024, 1, 2048, -1, 1024, 0 ;triangle

instr 101,102,103
  iamp =      ampdbfs(p5)
  a1   oscil iamp, p4, p1-100
  kenv expseg 1, p3, .01
  a1   =      a1 * kenv
  outs a1, a1
endin

</CsInstruments>
<CsScore bin="csbeats">
; by Brian Baughn 3-14-05
; bbaughn@berklee.net

beats = 100
permeasure = 4

i101 m1 b1 q mp D3
i101 F3
i101 D3

i101 m2 b1 D3
i101 b3 D3

i101 m3 b1 D3
i101 F3
i101 D3

i101 m4 b1 D3
i101 b3 D3

i101 m5 b1 D3
i101 b4 G5

i101 m6 b1 E5
```

```

i101      b2      F5
i101      b3      e      Eb5
i101      b3.5    e
i101      b4      q

i101      m7 b1      e      D5
i101
i101      q
i101      e      Db5
i101
i101      q

i101      m8 b1      q      D5
i101
i101      E5
i101      D5

i102      m1 b2      q      D4
i102      b4      E4
i102      b4      Bb3

i102      m2 b2      F4
i102      b2      B3
i102      b4      C#4
i102      b4      Bb3

i102      m3 b2      q      D4
i102      b4      E4
i102      b4      Bb3

i102      m4 b2      F4
i102      b2      B3
i102      b4      C#4
i102      b4      Bb3

i103      m5 b2      e      F6
i103      b2      e      A5
i103      b2.5    e      D6
i103      b3      e      F6
i103      b3      e      A5
i103      b4      e      E6

i103      m6 b1      q      C#6
i103      q      D6
i103      e      C6
i103
i103      q

i103      m7 b1      e      B5
i103
i103      q
i103      e      Bb5
i103
i103      q

i103      m8 b1      e      F5
i103      b1      e      A5
i103      b1.5    e      D6
i103      b2      e      Bb5
i103      b2.5    e      D6
i103      b3      q      F5
i103      b3      A5
end
</CsScore>

</CsoundSynthesizer>

```

---

# **Partie IV. Référence Rapide des Opcodes**

---

---

# Table des matières

Référence Rapide des Opcodes ..... 3497

---

# Référence Rapide des Opcodes

## Syntaxe de l'orchestre : en-tête.

```
0dbfs = iarg
0dbfs

A4 = iarg

kr = iarg

ksmps = iarg

nchnls = iarg

nchnls_i = iarg

sr = iarg
```

## Syntaxe de l'orchestre : bloc d'instructions.

```
endin

endop

instr i, j, ...

opcode nom, outtypes, intypes
```

## Syntaxe de l'orchestre : macros.

```
#define NAME # replacement text #

#define NAME(a' b' c') # replacement text #

$NAME

#ifdef NAME
    ...
#else
    ...
#endif

#ifndef NAME
    ...
#else
    ...
#endif

#include "filename"

#undef NAME
```

## Générateurs de signal : synthèse/resynthèse additive.

```
ares adsyn kamod, kfmmod, ksmmod, ifilcod

ares adsynt kamp, kcps, iwfn, ifreqfn, iampfn, icnt [, iphs]

ar adsynt2 kamp, kcps, iwfn, ifreqfn, iampfn, icnt [, iphs]

aout beadsynt kFreqs[], kAmps[], kBws[] \
    [, inumosc, iflags, kfreq, kbw, ifn, iphs ]
aout beadsynt ifreqft, iampft, ibwft, inumosc \
    [, iflags, kfreq, kbw, ifn, iphs ]

aout beosc xfreq, kbw [, ifn, iphs, inoisetype ]

ares hsboscil kamp, ktone, kbrite, ibasfreq, iwfn, ioctfn \
    [, ioctcnt] [, iphs]
```

### Générateurs de signal : oscillateurs élémentaires.

```
kres lfo kamp, kcps [, itype]
ares lfo kamp, kcps [, itype]

ares oscbnk kcps, kamd, kfmd, kpmd, iovrlap, iseed, kllminf, kllmaxf, \
    kl2minf, kl2maxf, ilfomode, keqminf, keqmaxf, keqminl, keqmaxl, \
    keqminq, keqmaxq, iegmode, kfn [, illfn] [, il2fn] [, iegffn] \
    [, ieglfm] [, iegqfn] [, itabl] [, ioutfn]

ares oscil xamp, xcps [, ifn, iphs]
kres oscil kamp, kcps [, ifn, iphs]

ares oscil3 xamp, xcps [, ifn, iphs]
kres oscil3 kamp, kcps [, ifn, iphs]

ares oscili xamp, xcps [, ifn, iphs]
kres oscili kamp, kcps [, ifn, iphs]

ares oscilikt xamp, xcps, kfn [, iphs] [, istor]
kres oscilikt kamp, kcps, kfn [, iphs] [, istor]

ares osciliktp kcps, kfn, kphs [, istor]

ares oscilikts xamp, xcps, kfn, async, kphs [, istor]

ares osciln kamp, ifrq, ifn, itimes

ares oscils iamp, icps, iphs [, iflg]

ares poscil aamp, acps [, ifn, iphs]
ares poscil aamp, kcps [, ifn, iphs]
ares poscil kamp, acps [, ifn, iphs]
ares poscil kamp, kcps [, ifn, iphs]
ires poscil kamp, kcps [, ifn, iphs]
kres poscil kamp, kcps [, ifn, iphs]

ares poscil3 aamp, acps [, ifn, iphs]
ares poscil3 aamp, kcps [, ifn, iphs]
ares poscil3 kamp, acps [, ifn, iphs]
ares poscil3 kamp, kcps [, ifn, iphs]
ires poscil3 kamp, kcps [, ifn, iphs]
kres poscil3 kamp, kcps [, ifn, iphs]

kout vibr kAverageAmp, kAverageFreq, ifn

kout vibrato kAverageAmp, kAverageFreq, kRandAmountAmp, \
```



```
kRandAmountFreq, kAmpMinRate, kAmpMaxRate, kcpsMinRate, \
kcpsMaxRate, ifn [, iphs]
```

### Générateurs de signal : oscillateurs à spectre dynamique.

```
ares buzz xamp, xcps, knh, ifn [, iphs]

ares gbuzz xamp, xcps, knh, klh, kmul, ifn [, iphs]

ares mpulse kamp, kintvl [, ioffset]

aout [, asyncout] squnewave acps, aClip, aSkew, asyncin [, iMinSweep] [, iphase]
aout [, asyncout] squnewave acps, aClip, aSkew [, ksyncin] [, iMinSweep] [, iphase]

ares vco xamp, xcps, iwave, kpw [, ifn] [, imaxd] [, ileak] [, inyx] \
[, iphs] [, iskip]

ares vco2 kamp, kcps [, imode] [, kpw] [, kphs] [, inyx]

kfn vco2ft kcps, iwave [, inyx]

ifn vco2ift icps, iwave [, inyx]

ifn vco2init iwave [, ibasfn] [, ipmul] [, iminsiz] [, imaxsiz] [, isrcft]
```

### Générateurs de signal : synthèse MF.

```
a1, a2 crossfm xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]
a1, a2 crossfmi xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]
a1, a2 crossspm xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]
a1, a2 crosspmi xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]
a1, a2 crossfmpm xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]
a1, a2 crossfmpmi xfrq1, xfrq2, xndx1, xndx2, kcps, ifn1, ifn2 [, iphs1] [, iphs2]

ares fmb3 kamp, kfreq, kc1, kc2, kvdepth, kvrate[, ifn1, ifn2, ifn3, \
ifn4, ivfn]

ares fmbell kamp, kfreq, kc1, kc2, kvdepth, kvrate[, ifn1, ifn2, ifn3, \
ifn4, ivfn, isus]

ares fmmetal kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \
ifn4, ivfn

ares fmpercfl kamp, kfreq, kc1, kc2, kvdepth, kvrate[, ifn1, ifn2, \
ifn3, ifn4, ivfn]

ares fmrhode kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, \
ifn3, ifn4, ivfn

ares fmvoice kamp, kfreq, kvowel, ktilt, kvibamt, kvibrate[, ifn1, \
ifn2, ifn3, ifn4, ivibfn]

ares fmwurlie kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, \
ifn4, ivfn

ares foscil xamp, kcps, xcar, xmod, kndx, ifn [, iphs]

ares foscili xamp, kcps, xcar, xmod, kndx, ifn [, iphs]
```

### Générateurs de signal : synthèse granulaire.

```

asig diskgrain Sfname, kamp, kfreq, kpitch, kgrsize, kprate, \
    ifun, iolaps [,imaxgrsize , ioffset]

ares fof xamp, xfund, xform, koct, kband, kris, kdur, kdec, iolaps, \
    ifna, ifnb, itotdur [, iphs] [, ifmode] [, iskip]

ares fof2 xamp, xfund, xform, koct, kband, kris, kdur, kdec, iolaps, \
    ifna, ifnb, itotdur, kphs, kgliss [, iskip]

ares fog xamp, xdens, xtrans, aspd, koct, kband, kris, kdur, kdec, \
    iolaps, ifna, ifnb, itotdur [, iphs] [, itmode] [, iskip]

ares grain xamp, xpitch, xdens, kampoff, kpitchoff, kgdur, igfn, \
    iwfn, imgdur [, igrnd]

ares grain2 kcps, kfmd, kgdur, iovrlp, kfn, iwfn [, irpow] \
    [, iseed] [, imode]

ares grain3 kcps, kphs, kfmd, kpmf, kgdur, kdens, imaxovr, kfn, iwfn, \
    kfrpow, kprpow [, iseed] [, imode]

ares granule xamp, ivoice, iratio, imode, ithd, ifn, ipshift, igskip, \
    igskip_os, ilength, kgap, igap_os, kgsz, igsz_os, iatt, idec \
    [, iseed] [, ipitch1] [, ipitch2] [, ipitch3] [, ipitch4] [, ifnenv]

a1 [, a2, a3, a4, a5, a6, a7, a8] partikkel agrainfreq, \
    kdistribution, idisttab, async, kenv2amt, ienv2tab, ienv_attack, \
    ienv_decay, ksustain_amount, ka_d_ratio, kduration, kamp, igainmask, \
    kwavfreq, ksweepshape, iwavfreqstarttab, iwavfreqendtab, awavfm, \
    ifmamp, kfmenv, icosine, ktraincps, knumpartials, kchroma, \
    ichannelmask, krandmask, kwaveform1, kwaveform2, kwaveform3, \
    kwaveform4, iwaveamptab, asamplepos1, asamplepos2, asamplepos3, \
    asamplepos4, kwavekey1, kwavekey2, kwavekey3, kwavekey4, imax_grains \
    [, iopcode_id, ipanlaws]

kindex partikkelget kparameterindex, iopcode_id

partikkelset kparameterindex, kmaskindex, iopcode_id

async [,aphase] partikkelsync iopcode_id

ares [, ac] sndwarp xamp, xtimewarp, xresample, ifn1, ibeg, iwsz, \
    irandw, ioverlap, ifn2, itimemode

ar1, ar2 [,ac1] [, ac2] sndwarpst xamp, xtimewarp, xresample, ifn1, \
    ibeg, iwsz, irandw, ioverlap, ifn2, itimemode

asig syncgrain kamp, kfreq, kpitch, kgrsize, kprate, ifun1, \
    ifun2, iolaps

asig syncloop kamp, kfreq, kpitch, kgrsize, kprate, klstart, \
    klend, ifun1, ifun2, iolaps[,istart, iskip]

ar vosim kamp, kFund, kForm, kDecay, kPulseCount, kPulseFactor, ifn [, iskip]

```

### Générateurs de signal : synthèse hyper vectorielle.

```

hvs1 kx, inumParms, inumPointsX, iOutTab, iPositionsTab, iSnapTab [, iConfigTab]

```

```
hvs2 kx, ky, inumParms, inumPointsX, inumPointsY, iOutTab, iPositionsTab, iSnapTab [,
    iConfigTab]
```

```
hvs3 kx, ky, kz, inumParms, inumPointsX, inumPointsY, inumPointsZ, iOutTab, iPositions-
    Tab, iSnapTab [, iConfigTab]
```

### Générateurs de signal : générateurs linéaires et exponentiels.

```
ky bpf kx, kx1, ky1, kx2, ..., kxn, kyn
iy bpf ix, ix1, iy1, ix2, ..., ixn, iyn
kys[] bpf kxs[], kx1, ky1, kx2, ..., kxn, kyn
iys[] bpf ixs[], ix1, iy1, ix2, ..., ixn, iyn
ky bpf kx, kxs[], kys[]
iy bpf ix, ixs[], iys[]
ay bpf ax, kx1, ky1, kx2, ..., kxn, kyn
ay bpf ax, kxs[], kys[]
ky, kw bpf kx, kxs[], kys[], kws[]
```

```
ky bpfcos kx, kx1, ky1, kx2, ..., kxn, kyn
kys[] bpfcos kxs[], kx1, ky1, kx2, ..., kxn, kyn
ky bpfcos kx, kxs[], kys[]
ky bpfcos kx, ixs[], iys[]
ky, kz bpfcos kx, kxs[], kys[], kzs[]
ky, kz bpfcos kx, ixs[], iys[], izs[]
kys[] bpfcos kxs[], kx1, ky1, kx2, ..., kxn, kyn
ky bpfcos kx, ixs[], iys[]
ky, kz bpfcos kx, kxs[], kys[], kzs[]
```

```
ares cosseg ia, idur1, ib [, idur2] [, ic] [...]
kres cosseg ia, idur1, ib [, idur2] [, ic] [...]
```

```
ares cossegb ia, itim1, ib [, itim2] [, ic] [...]
kres cossegb ia, itim1, ib [, itim2] [, ic] [...]
```

```
ares cossegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
kres cossegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
```

```
kout expcurve kindex, ksteepness
```

```
ares expon ia, idur, ib
kres expon ia, idur, ib
```

```
ares expseg ia, idur1, ib [, idur2] [, ic] [...]
kres expseg ia, idur1, ib [, idur2] [, ic] [...]
```

```
ares expsega ia, idur1, ib [, idur2] [, ic] [...]
```

```
ares expsegb ia, itim1, ib [, itim2] [, ic] [...]
kres expsegb ia, itim1, ib [, itim2] [, ic] [...]
```

```
ares expsegba ia, itim1, ib [, itim2] [, ic] [...]
```

```
ares expsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
kres expsegr ia, idur1, ib [, idur2] [, ic] [...], irel, iz
```

```
kout gainslider kindex
```

```
ky lincos kx, ky0, ky1 [, kx0, kx1 ]
iy lincos ix, iy0, iy1 [, ix0, ix1 ]
```

```
ares line ia, idur, ib
kres line ia, idur, ib
```

```

ky linlin kx, ky0, kyl [, kx0, kx1 ]
iy linlin ix, iy0, iyl [, ix0, ix1 ]
kys[] linlin kxs[], ky0, kyl [, kx0, kx1 ]
iys[] linlin ixs[], ky0, kyl [, kx0, kx1 ]
kC[] linlin kx, kA[], kB[] [, kx0, kx1 ]

ares linseg ia, idurl, ib [, idur2] [, ic] [...]
kres linseg ia, idurl, ib [, idur2] [, ic] [...]

ares linsegb ia, itiml, ib [, itim2] [, ic] [...]
kres linsegb ia, itiml, ib [, itim2] [, ic] [...]

ares linsegr ia, idurl, ib [, idur2] [, ic] [...], irel, iz
kres linsegr ia, idurl, ib [, idur2] [, ic] [...], irel, iz

kout logcurve kindex, ksteepness

ksig loopseg kfreq, ktrig, iphase, kvalue0, ktime0 [, kvalue1] [, ktime1] \
    [, kvalue2] [, ktime2] [...]

ksig loopsegg kphase, kvalue0, kdur0, kvalue1 \
    [, kdurl, ... , kdurN-1, kvalueN]

ksig looptseg kfreq, ktrig, iphase, kvalue0, ktype0, ktime0 [, kvalue1] [,ktype1] [,
    ktime1] \
    [, kvalue2] [,ktype2] [, ktime2] [...] [, kvalueN] [,ktypeN] [, ktimeN]

ksig loopxseg kfreq, ktrig, iphase, ktime0, kvalue0 [, ktime1] [, kvalue1] \
    [, ktime2] [, kvalue2] [...]

ksig lpshold kfreq, ktrig, iphase, ktime0, kvalue0 [, kvalue1] [, ktime1] [, kvalue2]
    [, ktime2] [...]

ksig lpsholdp kphase, ktrig, ktime0, kvalue0 [, kvalue1] [, ktime1] \
    [, kvalue2] [, ktime2] [...]

kscl scale kinput, kmax, kmin

ares transeg ia, idur, itype, ib [, idur2] [, itype] [, ic] ...
kres transeg ia, idur, itype, ib [, idur2] [, itype] [, ic] ...

ares transegb ia, itim, itype, ib [, itim2] [, itype] [, ic] ...
kres transegb ia, itim, itype, ib [, itim2] [, itype] [, ic] ...

ares transegr ia, idur, itype, ib [, idur2] [, itype] [, ic] ...
kres transegr ia, idur, itype, ib [, idur2] [, itype] [, ic] ...

kout xyyscale kx, ky, k00, k10, k01, k11

```

### Générateurs de signal : générateurs d'enveloppe.

```

ares adsr iatt, idec, islev, irel [, idel]
kres adsr iatt, idec, islev, irel [, idel]

ares envlpx xamp, irise, idur, idec, ifn, iatss, iatdec [, ixmod]
kres envlpx kamp, irise, idur, idec, ifn, iatss, iatdec [, ixmod]

ares envlpxr xamp, irise, idec, ifn, iatss, iatdec [, ixmod] [,irind]
kres envlpxr kamp, irise, idec, ifn, iatss, iatdec [, ixmod] [,irind]

ares linen xamp, irise, idur, idec
kres linen kamp, irise, idur, idec

```

```
ares linenr xamp, irise, idec, iatdec
kres linenr kamp, irise, idec, iatdec

ares madsr iatt, idec, islev, irel [, idel] [, ireltim]
kres madsr iatt, idec, islev, irel [, idel] [, ireltim]

ares mxadsr iatt, idec, islev, irel [, idel] [, ireltim]
kres mxadsr iatt, idec, islev, irel [, idel] [, ireltim]

ares xadsr iatt, idec, islev, irel [, idel]
kres xadsr iatt, idec, islev, irel [, idel]
```

### Générateurs de signal : modèles et émulations.

```
ares bamboo kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \
    [, ifreq1] [, ifreq2]

ares barmodel kbcL, kbcR, iK, ib, kscan, iT30, ipos, ivel, iwid

ares cabasa iamp, idettack [, inum] [, idamp] [, imaxshake]

aI3, aV2, aVl chuap kL, kR0, kC1, kG, kGa, kGb, kE, kC2, iI3, iV2, iV1, ktime_step

ares crunch iamp, idettack [, inum] [, idamp] [, imaxshake]

ares dripwater kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \
    [, ifreq1] [, ifreq2]

ares gendy kamp, kampdist, kdurdist, kadpar, kddpar, kminfreq, kmaxfreq, \
    kampscl, kdurscl [, initcps] [, knum]
kres gendy kamp, kampdist, kdurdist, kadpar, kddpar, kminfreq, kmaxfreq, \
    kampscl, kdurscl [, initcps] [, knum]

ares gendyc kamp, kampdist, kdurdist, kadpar, kddpar, kminfreq, kmaxfreq, \
    kampscl, kdurscl [, initcps] [, knum]
kres gendyc kamp, kampdist, kdurdist, kadpar, kddpar, kminfreq, kmaxfreq, \
    kampscl, kdurscl [, initcps] [, knum]

ares gendyx kamp, kampdist, kdurdist, kadpar, kddpar, kminfreq, kmaxfreq, \
    kampscl, kdurscl, kcurveup, kcurvedown [, initcps] [, knum]
kres gendyx kamp, kampdist, kdurdist, kadpar, kddpar, kminfreq, kmaxfreq, \
    kampscl, kdurscl, kcurveup, kcurvedown [, initcps] [, knum]

ares gogobel kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivfn

ares guiro kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] [, ifreq1]

ax, ay, az lorenz ksv, krv, kbv, kh, ix, iy, iz, iskip [, iskipinit]

kiter, koutrig mandel ktrig, kx, ky, kmaxIter

ares mandol kamp, kfreq, kpluck, kdetune, kgain, ksize \
    [, ifn] [, iminfreq]

ares marimba kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec \
    [, idoubles] [, itriples]

ares moog kamp, kfreq, kfiltq, kfiltrate, kvibf, kvamp, iafn, iwfn, ivfn

ax, ay, az planet kmass1, kmass2, ksep, ix, iy, iz, ivx, ivy, ivz, idelta \
    [, ifriction] [, iskip]

ares prepiano ifreq, iNS, iD, iK, \
```

```

    iT30, iB, kbcl, kbcr, imass, ihvfreq, iinit, ipos, ivel, isfreq, \
    isspread[, irattles, irubbers]
al,ar prepiano ifreq, iNS, iD, iK, \
    iT30, iB, kbcl, kbcr, imass, ihvfreq, iinit, ipos, ivel, isfreq, \
    isspread[, irattles, irubbers]

ares sandpaper iamp, idettack [, inum] [, idamp] [, imaxshake]

ares sekere iamp, idettack [, inum] [, idamp] [, imaxshake]

ares shaker kamp, kfreq, kbeans, kdamp, ktimes [, idecay]

ares sleighbells kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \
    [, ifreq1] [, ifreq2]

ares stix iamp, idettack [, inum] [, idamp] [, imaxshake]

ares tambourine kamp, idettack [, inum] [, idamp] [, imaxshake] [, ifreq] \
    [, ifreq1] [, ifreq2]

ares vibes kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec

ares voice kamp, kfreq, kphoneme, kform, kvibf, kvamp, ifn, ivfn

```

### Générateurs de signal : phaseurs.

```

aexp,aphephase kfreq, kR

ares phase xcps [, iphs]
kres phase kcps [, iphs]

ares phasorbnk xcps, kndx, icnt [, iphs]
kres phasorbnk kcps, kndx, icnt [, iphs]

aindex sc_phase xtrig, xrate, kstart, kend [, kresetPos]
kindex sc_phase xtrig, xrate, kstart, kend [, kresetPos]

aphase, asyncout syncphase xcps, asyncin, [, iphs]

```

### Générateurs de signal : générateurs de nombres aléatoires (de bruit).

```

ares betarand krange, kalpha, kbeta
ires betarand krange, kalpha, kbeta
kres betarand krange, kalpha, kbeta

ares bexprnd krange
ires bexprnd krange
kres bexprnd krange

ares cauchy kalpha
ires cauchy kalpha
kres cauchy kalpha

ares cauchy kalpha, xamp, xcps
ires cauchy kalpha, xamp, xcps
kres cauchy kalpha, xamp, xcps

aout cuserrnd kmin, kmax, ktableNum
iout cuserrnd imin, imax, itableNum
kout cuserrnd kmin, kmax, ktableNum

```

```

aout duserrnd ktableNum
iout duserrnd itableNum
kout duserrnd ktableNum

ares dust kamp, kdensity
kres dust kamp, kdensity

ares dust2 kamp, kdensity
kres dust2 kamp, kdensity

ares exprand klambda
ires exprand klambda
kres exprand klambda

ares exprandi klambda, xamp, xcps
ires exprandi klambda, xamp, xcps
kres exprandi klambda, xamp, xcps

ares fractalnoise kamp, kbeta

ares gauss krange
ires gauss krange
kres gauss krange

ares gaussi krange, xamp, xcps
ires gaussi krange, xamp, xcps
kres gaussi krange, xamp, xcps

ares gausstrig kamp, kcps, kdev [, imode] [, ifrst1]
kres gausstrig kamp, kcps, kdev [, imode] [, ifrst1]

ians getseed
kans getseed

kout jitter kamp, kcpsMin, kcpsMax

kout jitter2 ktotamp, kamp1, kcps1, kamp2, kcps2, kamp3, kcps3[ ,iopt]

ares jspline xamp, kcpsMin, kcpsMax
kres jspline kamp, kcpsMin, kcpsMax

ares linrand krange
ires linrand krange
kres linrand krange

ares noise xamp, kbeta

ares pcauchy kalpha
ires pcauchy kalpha
kres pcauchy kalpha

ares pinker

ares pinkish xin [, imethod] [, inumbands] [, iseed] [, iskip]

ares poisson klambda
ires poisson klambda
kres poisson klambda

ares rand xamp [, iseed] [, isel] [, ioffset]
kres rand xamp [, iseed] [, isel] [, ioffset]

ares randh xamp, xcps [, iseed] [, isize] [, ioffset]
kres randh kamp, kcps [, iseed] [, isize] [, ioffset]

ares randi xamp, xcps [, iseed] [, isize] [, ioffset]
kres randi kamp, kcps [, iseed] [, isize] [, ioffset]

```

```

ares random kmin, kmax
ires random imin, imax
kres random kmin, kmax

ares randomh kmin, kmax, xcps [,imode] [,ifirstval]
kres randomh kmin, kmax, kcps [,imode] [,ifirstval]

ares randomi kmin, kmax, xcps [,imode] [,ifirstval]
kres randomi kmin, kmax, kcps [,imode] [,ifirstval]

ax rnd31 kscl, krpow [, iseed]
ix rnd31 iscl, irpow [, iseed]
kx rnd31 kscl, krpow [, iseed]

ares rspline xrangeMin, xrangeMax, kcpsMin, kcpsMax
kres rspline krangeMin, krangeMax, kcpsMin, kcpsMax

seed ival

kout trandom ktrig, kmin, kmax

ares trirand krange
ires trirand krange
kres trirand krange

ares unirand krange
ires unirand krange
kres unirand krange

ax urandom [imin, imax]
ix urandom [imin, imax]
kx urandom [imin, imax]

aout = urd(ktableNum)
iout = urd(itableNum)
kout = urd(ktableNum)

ares weibull ksigma, ktau
ires weibull ksigma, ktau
kres weibull ksigma, ktau

```

### Générateurs de signal : reproduction de sons échantillonnés.

```

a1 bbcutm asource, ibps, isubdiv, ibarlength, iphrasebars, inumrepeats \
    [, istutterspeed] [, istutterchance] [, ienvchoice ]

a1,a2 bbcuts asource1, asource2, ibps, isubdiv, ibarlength, iphrasebars, \
    inumrepeats [, istutterspeed] [, istutterchance] [, ienvchoice]

asig1[, asig2] flooper kamp, kpitch, istart, idur, ifad, ifn

asig1[,asig2] flooper2 kamp, kpitch, kloopstart, kloopend, kcrossfade, ifn \
    [, istart, imode, ifenv, iskip]

aleft, aright fluidAllOut

fluidCCi iEngineNumber, iChannelNumber, iControllerNumber, iValue

fluidCCk iEngineNumber, iChannelNumber, iControllerNumber, kValue

fluidControl ienginenum, kstatus, kchannel, \
    kdata1, kdata2 [,imgs]

```



```

ienginenum fluidEngine [iChorusEnabled] [, iReverbEnabled] [, iNumChannels] [, iPoly-
phony]

SPrograms[] fluidInfo ienginenum

isfnum fluidLoad soundfont, ienginenum[, ilistpresets]

fluidNote ienginenum, ichannelnum, imidikey, imidivel

aleft, aright fluidOut ienginenum

fluidProgramSelect ienginenum, ichannelnum, isfnum, ibanknum, ipresetnum

fluidSetInterpMethod ienginenum, ichannelnum, iInterpMethod

ar1 [,ar2] loscil xamp, kcps, ifn [, ibas] [, imod1] [, ibeg1] [, iend1] \
[, imod2] [, ibeg2] [, iend2]
aph, ar1 [,ar2] loscilphs xamp, kcps, ifn [, ibas] [, imod1] [, ibeg1] [, iend1] \
[, imod2] [, ibeg2] [, iend2]

ar1 [,ar2] loscil3 xamp, kcps, ifn [, ibas] [, imod1] [, ibeg1] [, iend1] \
[, imod2] [, ibeg2] [, iend2]
aph, ar1 [,ar2] loscil3phs xamp, kcps, ifn [, ibas] [, imod1] [, ibeg1] [, iend1] \
[, imod2] [, ibeg2] [, iend2]

ar1 [, ar2, ar3, ar4, ar5, ar6, ar7, ar8, ar9, ar10, ar11, ar12, ar13, ar14, \
ar15, ar16] loscilx xamp, kcps, ifn \
[, iwsiz, ibas, istr, imod, ibeg, iend]
ar[] loscilx xamp, kcps, ifn \
[, iwsiz, ibas, istr, imod, ibeg, iend]

ares lphasor xtrns [, ilps] [, ilpe] [, imode] [, istr] [, istor]

ares lposcil kamp, kfregratio, kloop, kend, ifn [, iphs]

ares lposcil3 kamp, kfregratio, kloop, kend, ifn [, iphs]

ar lposcila aamp, kfregratio, kloop, kend, ift [, iphs]

ar1, ar2 lposcilsa aamp, kfregratio, kloop, kend, ift [, iphs]

ar1, ar2 lposcilsa2 aamp, kfregratio, kloop, kend, ift [, iphs]

sfilist ifilhandle

ar1, ar2 sfinstr ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \
[, iflag] [, ioffset]

ar1, ar2 sfinstr3 ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \
[, iflag] [, ioffset]

ares sfinstr3m ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \
[, iflag] [, ioffset]

ares sfinstrm ivel, inotenum, xamp, xfreq, instrnum, ifilhandle \
[, iflag] [, ioffset]

ir sfload "filename"

ar1, ar2 sflooper ivel, inotenum, kamp, kpitch, ipreindex, kloopstart, kloopend,
kcrossfade \
[, istart, imode, ifenv, iskip]

sfpassign istartindex, ifilhandle[, imsgs]

ar1, ar2 sfplay ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]

```

```

ar1, ar2 sfplay3 ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]

ares sfplay3m ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]

ares sfplaym ivel, inotenum, xamp, xfreq, ipreindex [, iflag] [, ioffset] [, ienv]

sfplist ifilhandle

ir sfpreset iprog, ibank, ifilhandle, ipreindex

asig, krec sndloop ain, kpitch, ktrig, idur, ifad

ares waveset ain, krep [, ilen]

```

### Générateurs de signal : synthèse par balayage.

```

scanhammer isrc, idst, ipos, imult

ares scans kamp, kfreq, ifn, id [, iorder]

aout scantable kamp, kpch, ipos, imass, istiff, idamp, ivel

scanu init, irate, ifnvel, ifnmass, ifnstif, ifncentr, ifndamp, kmass, \
      kstif, kcentr, kdamp, ileft, irect, kpos, kstrngth, ain, idisp, id

kpos, kvel xscanmap iscan, kamp, kvamp [, iwhich]

ares xscans kamp, kfreq, ifntraj, id [, iorder]

xscansmap kpos, kvel, iscan, kamp, kvamp [, iwhich]

xscanu init, irate, ifnvel, ifnmass, ifnstif, ifncentr, ifndamp, kmass, \
      kstif, kcentr, kdamp, ileft, irect, kpos, kstrngth, ain, idisp, id

```

### Générateurs de signal : opcodes STK.

```

asignal STKBandedWG ifrequency, iamplitude, [kpress, kv1[, kmot, kv2[, klfo, kv3[, klfo-
depth, kv4[, kvel, kv5[, kstrk, kv6[, kinstr, kv7]]]]]]]]

asignal STKBeethree ifrequency, iamplitude, [kop4, kv1[, kop3, kv2[, klfo, kv3[, klfo-
depth, kv4[, kadsr, kv5]]]]]]

asignal STKBlowBotl ifrequency, iamplitude, [knoise, kv1[, klfo, kv2[, klfo-
depth, kv3[, kv1, kv4]]]]

asignal STKBlowHole ifrequency, iamplitude, [kreed, kv1[, knoise, kv2[, khole, kv3[,
kreg, kv4[, kbreath, kv5]]]]]]

asignal STKBowed ifrequency, iamplitude, [kpress, kv1[, kpos, kv2[, klfo, kv3[, klfo-
depth, kv4[, kv1, kv5]]]]]]

asignal STKBrass ifrequency, iamplitude, [klip, kv1[, kslide, kv2[, klfo, kv3[, klfo-
depth, kv4[, kv1, kv5]]]]]]

asignal STKClarinet ifrequency, iamplitude, [kstiff, kv1[, knoise, kv2[, klfo, kv3[,
klfo-
depth, kv4[, kbreath, kv5]]]]]]

asignal STKDrummer ifrequency, iamplitude

```

```

asignal STKFMVoices ifrequency, iamplitude, [kvowel, kv1[, kspec, kv2[, klfo, kv3[, kl-
fodepth, kv4[, kadsr, kv5]]]]]

asignal STKFlute ifrequency, iamplitude, [kjet, kv1[, knoise, kv2[, klfo, kv3[, klfo-
depth, kv4[, kbreath, kv5]]]]]

asignal STKHevyMetl ifrequency, iamplitude, [kmod, kv1[, kcross, kv2[, klfo, kv3[, kl-
fodepth, kv4[, kadsr, kv5]]]]]

asignal STKMandolin ifrequency, iamplitude, [kbody, kv1[, kpos, kv2[, ksus, kv3[, kde-
tune, kv4[, kmic, kv5]]]]]

asignal STKModalBar ifrequency, iamplitude, [khard, kv1[, kpos, kv2[, klfo, kv3[, klfo-
depth, kv4[, kmix, kv5[, kvol, kv6[, kinstr, kv7]]]]]]]

asignal STKMoog ifrequency, iamplitude, [kq, kv1[, krate, kv2[, klfo, kv3[, klfo-
depth, kv4[, kvol, kv5]]]]]

asignal STKPercFlut ifrequency, iamplitude, [kmod, kv1[, kcross, kv2[, klfo, kv3[, kl-
fodepth, kv4[, kadsr, kv5]]]]]

asignal STKPlucked ifrequency, iamplitude

asignal STKResonate ifrequency, iamplitude, [kfreq, kv1[, kpole, kv2[, knotch, kv3[,
kzero, kv4[, kenv, kv5]]]]]

asignal STKRhodey ifrequency, iamplitude, [kmod, kv1[, kcross, kv2[, klfo, kv3[, klfo-
depth, kv4[, kadsr, kv5]]]]]

asignal STKSaxofony ifrequency, iamplitude, [kstiff, kv1[, kapert, kv2[, kblow, kv3[,
knoise, kv4[, klfo, kv5[, klfo-
depth, kv6[, kbreath, kv7]]]]]]]

asignal STKShakers ifrequency, iamplitude, [kenerg, kv1[, kdecay, kv2[, kshake, kv3[,
knum, kv4[, kres, kv5[, kinstr, kv6]]]]]]]

asignal STKSimple ifrequency, iamplitude, [kpos, kv1[, kcross, kv2[, kenv, kv3[, kgain,
kv4]]]]]

asignal STKSitar ifrequency, iamplitude

asignal STKStifKarp ifrequency, iamplitude, [kpos, kv1[, ksus, kv2[, kstretch, kv3]]]

asignal STKTubeBell ifrequency, iamplitude, [kmod, kv1[, kcross, kv2[, klfo, kv3[, kl-
fodepth, kv4[, kadsr, kv5]]]]]

asignal STKVoicForm ifrequency, iamplitude, [kmix, kv1[, ksel, kv2[, klfo, kv3[, klfo-
depth, kv4[, kloud, kv5]]]]]

asignal STKWhistle ifrequency, iamplitude, [kmod, kv1[, knoise, kv2[, kfipfreq, kv3[,
kfipgain, kv4[, kvol, kv5]]]]]

asignal STKWurley ifrequency, iamplitude, [kmod, kv1[, kcross, kv2[, klfo, kv3[, klfo-
depth, kv4[, kadsr, kv5]]]]]

```

### Générateurs de signal : accès aux tables.

```

kres oscill idel, kamp, idur [, ifn]

kres oscilli idel, kamp, idur [, ifn]

ares ptable andx, ifn [, ixmode] [, ixoff] [, iwrap]
ires ptable indx, ifn [, ixmode] [, ixoff] [, iwrap]
kres ptable kndx, ifn [, ixmode] [, ixoff] [, iwrap]

```

```
ares ptable3 andx, ifn [, ixmode] [, ixoff] [, iwrap]
ires ptable3 indx, ifn [, ixmode] [, ixoff] [, iwrap]
kres ptable3 kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

```
ares ptablei andx, ifn [, ixmode] [, ixoff] [, iwrap]
ires ptablei indx, ifn [, ixmode] [, ixoff] [, iwrap]
kres ptablei kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

```
ir tab_i indx, ifn[, ixmode]
kr tab kndx, ifn[, ixmode]
ar tab xndx, ifn[, ixmode]
tabw_i isig, indx, ifn [,ixmode]
tabw ksig, kndx, ifn [,ixmode]
tabw asig, andx, ifn [,ixmode]
```

```
ares table andx, ifn [, ixmode] [, ixoff] [, iwrap]
ires table indx, ifn [, ixmode] [, ixoff] [, iwrap]
kres table kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

```
ares table3 andx, ifn [, ixmode] [, ixoff] [, iwrap]
ires table3 indx, ifn [, ixmode] [, ixoff] [, iwrap]
kres table3 kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

```
ares tablei andx, ifn [, ixmode] [, ixoff] [, iwrap]
ires tablei indx, ifn [, ixmode] [, ixoff] [, iwrap]
kres tablei kndx, ifn [, ixmode] [, ixoff] [, iwrap]
```

### Générateurs de signal : synthèse par terrain d'ondes.

```
aout wterrain kamp, kpch, k_xcenter, k_ycenter, k_xradius, k_yradius, \
      itabx, itaby
```

### Générateurs de signal : modèles physiques par guide d'onde.

```
ares pluck kamp, kcps, icps, ifn, imeth [, iparm1] [, iparm2]
```

```
ares repluck iplk, kamp, icps, kpick, krefl, axcite
```

```
ares streson asig, kfr, kfdbgain
```

```
ares wgbow kamp, kfreq, kpres, krat, kvibf, kvamp \
      [, ifn] [, iminfreq]
```

```
ares wgbowedbar kamp, kfreq, kpos, kbowpres, kgain [, iconst] [, itvell] \
      [, ibowpos] [, ilow]
```

```
ares wgbrass kamp, kfreq, ktens, iatt, kvibf, kvamp \
      [, ifn] [, iminfreq]
```

```
ares wgclar kamp, kfreq, kstiff, \
      iatt, idetk, kngain, kvibf, kvamp [, ifn] [, iminfreq]
```

```
ares wgflute kamp, kfreq, kjet, iatt, \
      idetk, kngain, kvibf, kvamp [, ifn] [, iminfreq] [, ijetrf] [, iendrf]
```

```
ares wgpluck icps, iamp, kpick, iplk, idamp, ifilt, axcite
```

```
ares wgpluck2 iplk, kamp, icps, kpick, krefl
```

### E/S de signal : E/S fichier.

```

dumpk ksig, ifilename, iformat, iprd

dumpk2 ksig1, ksig2, ifilename, iformat, iprd

dumpk3 ksig1, ksig2, ksig3, ifilename, iformat, iprd

dumpk4 ksig1, ksig2, ksig3, ksig4, ifilename, iformat, iprd

ficlose ihandle
ficlose Sfilename

fin ifilename, iskipframes, iformat, ain1 [, ain2] [, ain3] [,...]
fin ifilename, iskipframes, iformat, arr[]

fini ifilename, iskipframes, iformat, in1 [, in2] [, in3] [, ...]

fink ifilename, iskipframes, iformat, kin1 [, kin2] [, kin3] [,...]

ihandle fiopen ifilename, imode

fout ifilename, iformat, aout1 [, aout2, aout3,...,aoutN]
fout ifilename, iformat, array[]

fouti ihandle, iformat, iflag, iout1 [, iout2, iout3,...,ioutN]

foutir ihandle, iformat, iflag, iout1 [, iout2, iout3,...,ioutN]

foutk ifilename, iformat, kout1 [, kout2, kout3,...,koutN]

fprintks "filename", "string", [, kval1] [, kval2] [...]

fprints "filename", "string" [, ival1] [, ival2] [...]

xout1[, xout2, xout3, ..., xoutN] hdf5read ifilename, ivariablename1[, ivariablename2,
    ivariablename3, ..., ivariablenameN]

hdf5write ifilename, xout1[, xout2, xout3, ..., xoutN]

Sres, kline readf ifilename

Sres, iline readfi ifilename

kres readk ifilename, iformat, iprd

kr1, kr2 readk2 ifilename, iformat, iprd

kr1, kr2, kr3 readk3 ifilename, iformat, iprd

kr1, kr2, kr3, kr4 readk4 ifilename, iformat, iprd

xout1[, xout2, xout3, ..., xoutN] websocket iport, xin

```

### E/S de signal : entrée de signal.

```

ar1 [, ar2 [, ar3 [, ... arN]]] diskin ifilcod[, kpitch[, iskiptim \
    [, iwraparound[, iformat[, iskipinit]]]]]
ar1[] diskin ifilcod[, kpitch[, iskiptim] \
    [, iwraparound[, iformat[, iskipinit]]]]]

a1[, a2[, ... aN]] diskin2 ifilcod[, kpitch[, iskiptim \

```

```

        [, iwrap[, iformat[, iwsizel, ibufsize[, iskipinit]]]]]]]
ar1[] diskin2 ifilcod[, kpitch[, iskiptim \
        [, iwrap[, iformat[, iwsizel, ibufsize[, iskipinit]]]]]]]

ar1 in
aarray in

ar1, ar2, ar3, ar4, ar5, ar6, ar7, ar8, ar9, ar10, ar11, ar12, ar13, ar14, \
        ar15, ar16, ar17, ar18, ar19, ar20, ar21, ar22, ar23, ar24, ar25, ar26, \
        ar27, ar28, ar29, ar30, ar31, ar32 in32

ain1[, ...] inch kchan1[,...]

ar1, ar2, ar3, ar4, ar5, ar6 inh

ar1, ar2, ar3, ar4, ar5, ar6, ar7, ar8 ino

ar1, ar2, ar3, a4 inq

inrg kstart, ain1 [,ain2, ain3, ..., ainN]

ar1, ar2 ins

ivalue invalue "channel name"
kvalue invalue "channel name"
Sname invalue "channel name"

ar1, ar2, ar3, ar4, ar5, ar6, ar7, ar8, ar9, ar10, ar11, ar12, \
        ar13, ar14, ar15, ar16 inx

inz ksig1

ar1, ar2 mp3in ifilcod[, iskiptim, iformat, iskipinit, ibufsize]
ar1 mp3in ifilcod[, iskiptim, iformat, iskipinit, ibufsize]

ar1[, ar2[, ar3[, ... a24]]] soundin ifilcod [, iskiptim] [, iformat] \
        [, iskipinit] [, ibufsize]

```

## E/S de signal : sortie de signal.

```

mdelay kstatus, kchan, kd1, kd2, kdelay

aout1 [,aout2 ... aoutX] monitor
aarra monitor

out asig1[, asig2,...]
out aarray

out32 asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8, asig10, \
        asig11, asig12, asig13, asig14, asig15, asig16, asig17, asig18, \
        asig19, asig20, asig21, asig22, asig23, asig24, asig25, asig26, \
        asig27, asig28, asig29, asig30, asig31, asig32

outc asig1 [, asig2] [...]

outch kchan1, asig1 [, kchan2] [, asig2] [...]

outh asig1, asig2, asig3, asig4, asig5, asig6

outo asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8

outq asig1, asig2, asig3, asig4

```

```

outq1 asig

outq2 asig

outq3 asig

outq4 asig

outrg kstart, aout1 [,aout2, aout3, ..., aoutN]

outs asig1, asig2

outs1 asig

outs2 asig

outvalue "channel name", ivalue
outvalue "channel name", kvalue
outvalue "channel name", "string"

outx asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8, \
      asig9, asig10, asig11, asig12, asig13, asig14, asig15, asig16

outz ksig1

soundout asig1, ifilcod [, iformat]

soundouts asig1, asigr, ifilcod [, iformat]

```

### E/S de signal : bus logiciel.

```

kval chani kchan
aval chani kchan

chano kval, kchan
chano aval, kchan

chn_k Sname, imode[, itype, idflt, imin, imax, ix, iy, iwidth, iheight, Sattributes]
chn_a Sname, imode
chn_s Sname, imode

chnclear Sname1[, Sname2, ...]

gival chnexport Sname, imode[, itype, idflt, imin, imax]
gkval chnexport Sname, imode[, itype, idflt, imin, imax]
gaval chnexport Sname, imode
gSval chnexport Sname, imode

ival chnget Sname
kval chnget Sname
aval chnget Sname
Sval chnget Sname
Sval chngetks Sname
ival[] chngeti Sname[]
kval[] chngetk Sname[]
aval[] chngeta Sname[]
Sval[] chngets Sname[]

chnmix aval, Sname

itype, imode, ictltype, idflt, imin, imax chnparams Sname

chnset ival, Sname

```

```

chnset kval, Sname
chnset aval, Sname
chnset sval, Sname
chnsetks sval, Sname
chnseti ival[], []Sname
chnsetk kval[], []Sname
chnseta aval[], []Sname
chnsets sval[], []Sname

setksmps iksmps

xinarg1 [, xinarg2] ... [xinargN] xin

xout xoutarg1 [, xoutarg2] ... [, xoutargN]

```

### E/S de signal : impression et affichage.

```

dispfft xsig, iprd, iwsiz [, iwtyp] [, idbout] [, iwtflg] [,imin] [,imax]

display xsig, iprd [, inprds] [, iwtflg]

flashtxt iwhich, String

print iarg [, iarg1] [, iarg2] [...]

printf_i Sfmt, itrig, [iarg1[, iarg2[, ... ]]]
printf Sfmt, ktrig, [xarg1[, xarg2[, ... ]]]

printk itime, kval [, ispace] [, inamed]

printk2 kvar [, inumspaces] [, inamed]

printks "string", itime [, xval1] [, xval2] [...]

printks2 "string", kval

prints "string" [, xval1] [, xval2] [...]

```

### E/S de signal : requêtes sur les fichiers sons.

```

ir filebit ifilcod [, iallowraw]

ir filelen ifilcod, [iallowraw]

ir filenchnls ifilcod [, iallowraw]

ir filepeak ifilcod [, ichnl]

ir filesr ifilcod [, iallowraw]

ir filevalid ifilcod

ir mp3len ifilcod

```

### Modificateurs de signal : modificateurs d'amplitude.



```
ares balance asig, acomp [, ihp] [, iskip]

ares balance2 asig, acomp [, ihp] [, iskip]

ares clip asig, imeth, ilimit [, iarg]

ar compress aasig, acsig, kthresh, kloknee, khiknee, kratio, katt, krel, ilook

ar compress2 aasig, acsig, kthresh, kloknee, khiknee, kratio, katt, krel, ilook

ares dam asig, kthreshold, icomp1, icomp2, irtime, iftime

ares gain asig, krms [, ihp] [, iskip]
```

### Modificateurs de signal : convolution et morphing.

```
ar1 [, ar2] [, ar3] [, ar4] convolve ain, ifilcod [, ichannel]

ares cross2 ain1, ain2, isize, ioverlap, iwin, kbias

ares dconv asig, isize, ifn

a1[, a2[, a3[, ... a8]]] ftconv ain, ift, iplen[, iskip samples \
    [, iirlen[, iskipinit]]]

ftmorf kftndx, iftn, iresfn

ares liveconv ain, ift, iplen, kupdate, kclear

ar1 [, ar2] [, ar3] [, ar4] pconvolve ain, ifilcod [, ipartitions size, ichannel]

ares tvconv asig1, asig2, xfreez1,
    xfreez2, iparts, ifils
```

### Modificateurs de signal : retard.

```
ares delay asig, idlt [, iskip]

ares delay1 asig [, iskip]

kr delayk ksig, idel[, imode]
kr vdel_k ksig, kdel, imdel[, imode]

ares delayr idlt [, iskip]

delayw asig

ares deltap kdlt

ares deltap3 xdlt

ares deltapi xdlt

ares deltapn xnumsamps

aout deltapx adel, iwsiz e

deltapxw ain, adel, iwsiz e

ares multitap asig [, itime1, igain1] [, itime2, igain2] [...]
```

```
ares vdelay asig, adel, imaxdel [, iskip]

ares vdelay3 asig, adel, imaxdel [, iskip]

aout vdelayx ain, adl, imd, iws [, ist]

aout1, aout2, aout3, aout4 vdelayxq ain1, ain2, ain3, ain4, adl, imd, iws [, ist]

aout1, aout2 vdelayxs ain1, ain2, adl, imd, iws [, ist]

aout vdelayxw ain, adl, imd, iws [, ist]

aout1, aout2, aout3, aout4 vdelayxwq ain1, ain2, ain3, ain4, adl, \
    imd, iws [, ist]

aout1, aout2 vdelayxws ain1, ain2, adl, imd, iws [, ist]
```

### Modificateurs de signal : panoramique et spatialisation.

```
ao1, ao2 bformdec isetup, aw, ax, ay, az [, ar, as, at, au, av \
    [, abk, al, am, an, ao, ap, aq]]
ao1, ao2, ao3, ao4 bformdec isetup, aw, ax, ay, az [, ar, as, at, \
    au, av [, abk, al, am, an, ao, ap, aq]]
ao1, ao2, ao3, ao4, ao5 bformdec isetup, aw, ax, ay, az [, ar, as, \
    at, au, av [, abk, al, am, an, ao, ap, aq]]
ao1, ao2, ao3, ao4, ao5, ao6, ao7, ao8 bformdec isetup, aw, ax, ay, az \
    [, ar, as, at, au, av [, abk, al, am, an, ao, ap, aq]]]

ao1, ao2 bformdec1 isetup, aw, ax, ay, az [, ar, as, at, au, av \
    [, abk, al, am, an, ao, ap, aq]]
ao1, ao2, ao3, ao4 bformdec1 isetup, aw, ax, ay, az [, ar, as, at, \
    au, av [, abk, al, am, an, ao, ap, aq]]
ao1, ao2, ao3, ao4, ao5 bformdec1 isetup, aw, ax, ay, az [, ar, as, \
    at, au, av [, abk, al, am, an, ao, ap, aq]]
ao1, ao2, ao3, ao4, ao5, ao6, ao7, ao8 bformdec1 isetup, aw, ax, ay, az \
    [, ar, as, at, au, av [, abk, al, am, an, ao, ap, aq]]]
aout[] bformdec1 isetup, abform[]

aw, ax, ay, az bformenc asig, kalpha, kbeta, kord0, kord1
aw, ax, ay, az, ar, as, at, au, av bformenc asig, kalpha, kbeta, \
    kord0, kord1, kord2
aw, ax, ay, az, ar, as, at, au, av, ak, al, am, an, ao, ap, aq bformenc \
    asig, kalpha, kbeta, kord0, kord1, kord2, kord3

aw, ax, ay, az bformenc1 asig, kalpha, kbeta
aw, ax, ay, az, ar, as, at, au, av bformenc1 asig, kalpha, kbeta
aw, ax, ay, az, ar, as, at, au, av, ak, al, am, an, ao, ap, aq bformenc1 \
    asig, kalpha, kbeta
aarray[] bformenc1 asig, kalpha, kbeta

aleft, aright, irt60low, irt60high, imfp hrtfearly asrc, ksrcx, ksrcy, ksrcz, klstnrx,
    klstnry, klstnrz, \
    ifile1, ifiler, ideoform [, ifade, isr, iorder, ithreed, kheadrot, iroomx, iroomy,
    iroomz, iwallhigh, \
    iwallow, iwallgain1, iwallgain2, iwallgain3, ifloorhigh, ifloorlow, ifloorgain1,
    ifloorgain2, \
    ifloorgain3, iceilinghigh, iceilinglow, iceilinggain1, iceilinggain2, iceiling-
    gain3]

aleft, aright hrtfer asig, kaz, kelev, HRTFcompact

aleft, aright hrtfmove asrc, kAz, kElev, ifile1, ifiler [, imode, ifade, isr]
```

```

aleft, aright hrtfmove2 asrc, kAz, kElev, ifile1, ifiler [,ioverlap, iradius, isr]

aleft, aright, idel hrtfververb asrc, ilowrt60, ihighrt60, ifile1, ifiler [,isr, imfp,
iorder]

aleft, aright hrtfstat asrc, iAz, iElev, ifile1, ifiler [,iradius, isr]

a1, a2 locsend
a1, a2, a3, a4 locsend

a1, a2 locsig asig, kdegree, kdistance, kreverbsend
a1, a2, a3, a4 locsig asig, kdegree, kdistance, kreverbsend

a1, a2, a3, a4 pan asig, kx, ky, ifn [, imode] [, ioffset]

a1, a2 pan2 asig, xp [, imode]

a1, a2, a3, a4 space asig, ifn, ktime, kreverbsend, kx, ky

aW, aX, aY, aZ spat3d ain, kX, kY, kZ, idist, ift, imode, imdel, iovr [, istor]

aW, aX, aY, aZ spat3di ain, iX, iY, iZ, idist, ift, imode [, istor]

spat3dt ioutft, iX, iY, iZ, idist, ift, imode, irlen [, iftnocl]

k1 spdist ifn, ktime, kx, ky

a1, a2, a3, a4 spsend

ar1[, ar2...] vbap asig, kazim \
    kelev] [, kspread] [, ilayout]
array[] vbap asig, kazim [,
    kelev] [, kspread] [, ilayout]

ar1, ..., ar16 vbap16 asig, kazim [, kelev] [, kspread]

ar1, ..., ar16 vbap16move asig, idur, ispread, ifldnum, ifld1 \
    [, ifld2] [...]

ar1, ar2, ar3, ar4 vbap4 asig, kazim [, kelev] [, kspread]

ar1, ar2, ar3, ar4 vbap4move asig, idur, ispread, ifldnum, ifld1 \
    [, ifld2] [...]

ar1, ..., ar8 vbap8 asig, kazim [, kelev] [, kspread]

ar1, ..., ar8 vbap8move asig, idur, ispread, ifldnum, ifld1 \
    [, ifld2] [...]

k1[, k2...] vbapg kazim [,kelev] [, kspread] [, ilayout]
karray[] vbapg kazim [,kelev] [, kspread] [, ilayout]

kr1[, kr2...] vbapgmmove idur, ispread, ifldnum, ifld1 \
    [, ifld2] [...]
karray[] vbapgmmove idur, ispread, ifldnum, ifld1 \
    [, ifld2] [...]

vbaplsinit idim, ilsnum [, idir1] [, idir2] [...] [, idir32]
vbaplsinit idim, ilsnum, ilsarray

ar1[, ar2...] vbapmove asig, idur, ispread, ifldnum, ifld1 \
    [, ifld2] [...]
aarray[] vbapmove asig, idur, ispread, ifldnum, ifld1 \
    [, ifld2] [...]

```

```
vbapz inumchnls, istartndx, asig, kazim [, kelev] [, kspread]

vbapzmove inumchnls, istartndx, asig, idur, ispread, ifldnum, ifld1, \
    ifld2, [...]
```

### Modificateurs de signal : réverbération.

```
ares alpass asig, xrvt, ilpt [, iskip] [, insmps]

a1, a2 babo asig, ksrcx, ksrcy, ksrcz, irx, iry, irz [, idiff] [, ifno]

ares comb asig, krvt, ilpt [, iskip] [, insmps]

ares combinv asig, krvt, ilpt [, iskip] [, insmps]

aoutL, aoutR freeverb ainL, ainR, kRoomSize, kHFDamp[, iSRate[, iSkip]]

ares nestedap asig, imode, imaxdel, idel1, igain1 [, idel2] [, igain2] \
    [, idel3] [, igain3] [, istor]

ares nreverb asig, ktime, khdif [, iskip] [, inumCombs] [, ifnCombs] \
    [, inumAlpas] [, ifnAlpas]

a1[, a2, ...] platerrev itabexcite. itabouts, kbndry, iaspect, istiff, idecay, iloss,
    aexcite1[, aexcite2, ...]

ares reverb asig, krvt [, iskip]

ares reverb2 asig, ktime, khdif [, iskip] [, inumCombs] \
    [, ifnCombs] [, inumAlpas] [, ifnAlpas]

aoutL, aoutR reverbsc ainL, ainR, kfblvl, kfco[, israte[, ipitchm[, iskip]]]

ares valpass asig, krvt, xlpt, imaxlpt [, iskip] [, insmps]

ares vcomb asig, krvt, xlpt, imaxlpt [, iskip] [, insmps]
```

### Modificateurs de signal : opérateurs du niveau échantillon.

```
denorm a1[, a2[, a3[, ... ]]]

ares diff asig [, iskip]
kres diff ksig [, iskip]

kres downsamp asig [, iwlen]

ares fold asig, kincr

ares integ asig [, iskip]
kres integ ksig [, iskip]

ares interp ksig [, iskip] [, imode] [, ivalue]

ares ntrpol asig1, asig2, kpoint [, imin] [, imax]
ires ntrpol isig1, isig2, ipoint [, imin] [, imax]
kres ntrpol ksig1, ksig2, kpoint [, imin] [, imax]

a(x) (arguments de taux-k seulement)

i(x) (argument de taux-k ou de taux-i)
```

**i**(karray,index1, ...) (k-array with indices)

**k**(x) (arguments de taux-i seulement)

**k**(x) (arguments de taux-a seulement)

**S**(x) (control-rate or init-rate arg)

ares **samphold** asig, agate [, ival] [, ivstor]

kres **samphold** ksig, kgate [, ival] [, ivstor]

ares **upsamp** ksig

kval **valet** kndx, avar

**vaset** kval, kndx, avar

### Modificateurs de signal : limiteurs de signal.

ares **limit** asig, klow, khigh

ires **limit** isig, ilow, ihigh

kres **limit** ksig, klow, khigh

ires[] **limit** isig[], ilow, ihigh

kres[] **limit** ksig[], klow, khigh

ares **mirror** asig, klow, khigh

ires **mirror** isig, ilow, ihigh

kres **mirror** ksig, klow, khigh

ares **wrap** asig, klow, khigh

ires **wrap** isig, ilow, ihigh

kres **wrap** ksig, klow, khigh

### Modificateurs de signal : effets spéciaux.

ar **distort** asig, kdist, ifn[, ihp, istor]

ares **distort1** asig, kpregain, kpostgain, kshapel, kshape2[, imode]

ares **flanger** asig, adel, kfeedback [, imaxd]

ares **harmon** asig, kestfrq, kmaxvar, kgenfreq1, kgenfreq2, imode, \  
iminfrq, iprd

ares **harmon2** asig, koct, kfrq1, kfrq2, icpsmode, ilowest[, ipolarity]

ares **harmon3** asig, koct, kfrq1, \  
kfrq2, kfrq3, icpsmode, ilowest[, ipolarity]

ares **harmon4** asig, koct, kfrq1, \  
kfrq2, kfrq3, kfrq4, icpsmode, ilowest[, ipolarity]

ares **phaser1** asig, kfreq, kord, kfeedback [, iskip]

ares **phaser2** asig, kfreq, kq, kord, kmode, ksep, kfeedback

### Modificateurs de signal : filtres standard.

ares **atone** asig, khp [, iskip]

```

ares atonex asig, khp [, inumlayer] [, iskip]
ares atonex asig, ahp [, inumlayer] [, iskip]

ares biquad asig, kb0, kb1, kb2, ka0, ka1, ka2 [, iskip]
ares biquada asig, ab0, ab1, ab2, aa0, aa1, aa2 [, iskip]

ares butbp asig, kfreq, kband [, iskip]
ares butbr asig, kfreq, kband [, iskip]

ares buthp asig, kfreq [, iskip]
ares buthp asig, afreq [, iskip]

ares butlp asig, kfreq [, iskip]
ares butlp asig, afreq [, iskip]

ares butterbp asig, xfreq, xband [, iskip]
ares butterbr asig, xfreq, xband [, iskip]

ares butterhp asig, kfreq [, iskip]
ares butterhp asig, afreq [, iskip]

ares butterlp asig, kfreq [, iskip]
ares butterlp asig, afreq [, iskip]

ares clfilt asig, kfreq, itype, inpol [, ikind] [, ipbr] [, isba] [, iskip]
asig diode_ladder ain, xcf, xk [, inlp, isaturation, istor]

ashifted doppler asource, ksourceposition, kmicposition [, isoundspeed, ifiltercutoff]

asig K35_hpf ain, xcf, xQ [, inlp, isaturation, istor]
asig K35_lpf ain, xcf, xQ [, inlp, isaturation, istor]

ares median asig, ksize, imaxsize [, iskip]
kres mediank kin, ksize, imaxsize [, iskip]

aout mode ain, xfreq, xQ [, iskip]

ares tone asig, khp [, iskip]

ares tonex asig, khp [, inumlayer] [, iskip]
ares tonex asig, ahp [, inumlayer] [, iskip]

asig zdf_1pole ain, xcf [, kmode, istor]
alp, ahp zdf_1pole_mode ain, xcf [, istor]

asig zdf_2pole ain, xcf, xQ [, kmode, istor]
alp, abp, ahp zdf_2pole_mode ain, xcf, Q [, istor]

asig zdf_ladder ain, xcf, xQ [, istor]

```

### Modificateurs de signal : filtres standard : résonants.

```

ares areson asig, kcf, kbw [, iscl] [, iskip]
ares areson asig, acf, kbw [, iscl] [, iskip]
ares areson asig, kcf, abw [, iscl] [, iskip]

```

```
ares areson asig, acf, abw [, iscl] [, iskip]

ares bqrez asig, xfco, xres [, imode] [, iskip]

ares lowpass2 asig, kcf, kq [, iskip]

ares lowres asig, xcutoff, xresonance [, iskip]

ares lowresx asig, xcutoff, xresonance [, inumlayer] [, iskip]

ares lpf18 asig, xfco, xres, xdist [, iskip]

asig moogladder ain, kcf, kres[, istor]
asig moogladder ain, acf, kres[, istor]
asig moogladder ain, kcf, ares[, istor]
asig moogladder ain, acf, ares[, istor]

asig moogladder2 ain, kcf, kres[, istor]
asig moogladder2 ain, acf, kres[, istor]
asig moogladder2 ain, kcf, ares[, istor]
asig moogladder2 ain, acf, ares[, istor]

ares moogvcf asig, xfco, xres [,iscale, iskip]

ares moogvcf2 asig, xfco, xres [,iscale, iskip]

asig mvchpf ain, xcf[, istor]

asig mvclpf1 ain, xcf, xres[, istor]

asig mvclpf2 ain, xcf, xres[, istor]

asig mvclpf3 ain, xcf, xres[, istor]

asig1, asig2, asig3, asig4 mvclpf4 ain, xcf, xres[, istor]

ares reson asig, xcf, xbw [, iscl] [, iskip]

ares resonr asig, xcf, xbw [, iscl] [, iskip]

ares resonx asig, xcf, xbw [, inumlayer] [, iscl] [, iskip]

ares resony asig, kbf, kbw, inum, ksep [, isepmode] [, iscl] [, iskip]

ares resonz asig, xcf, xbw [, iscl] [, iskip]

ares rezzy asig, xfco, xres [, imode, iskip]

ahp,alp,abp,abr statevar ain, xcf, xq [, iosamps, istor]

alow, ahigh, aband svfilter asig, kcf, kq [, iscl][, iskip]

ares tbvcf asig, xfco, xres, kdist, kasym [, iskip]

ares vlowres asig, kfco, kres, iord, ksep
```

### Modificateurs de signal : filtres standard : contrôle.

```
kres aresonk ksig, kcf, kbw [, iscl] [, iskip]

kres atonek ksig, khp [, iskip]

kres lineto ksig, ktime
```

```

kres port ksig, ihtim [, isig]

kres portk ksig, khtim [, isig]

kres resonk ksig, kcf, kbw [, iscl] [, iskip]

kres resonxk ksig, kcf, kbw[, inumlayer, iscl, istor]

aout sc_lag ain, klagtime [, initialvalue=0]
kout sc_lag kin, klagtime [, initialvalue=0]

aout sc_lagud ain, klagup, klagdown
kout sc_lagud kin, klagup, klagdown

aout sc_trig ain, kdur
kout sc_trig kin, kdur

kres tlineto ksig, ktime, ktrig

kres tonek ksig, khp [, iskip]

```

### Modificateurs de signal : filtres spécialisés.

```

ares dcblock ain [, igain]

ares dcblock2 ain [, iorder] [, iskip]

asig eqfil ain, kcf, kbw, kgain[, istor]

ares exciter asig, kfreq, kceil, kharmonics, kblend

ares filter2 asig, iM, iN, ib0, ib1, ..., ibM, ia1, ia2, ..., iaN
kres filter2 ksig, iM, iN, ib0, ib1, ..., ibM, ia1, ia2, ..., iaN

am, af fmanal are, aim

asig fofilter ain, xcf, xris, xdec[, istor]

aout gtf ain, kfreq, idecay[, iorder, iphase]

ar1, ar2 hilbert asig

ar1, ar2 hilbert2 asig, ifftsize, ihopsize

ares nlfilt ain, ka, kb, kd, kC, kL

ares nlfilt2 ain, ka, kb, kd, kC, kL

ares pareq asig, kc, kv, kq [, imode] [, iskip]

ar rbjeq asig, kfco, klvl, kQ, kS[, imode]

ares zfilter2 asig, kdamp, kfreq, iM, iN, ib0, ib1, ..., ibM, \
    ia1, ia2, ..., iaN

```

### Modificateurs de signal : guides d'onde.

```

ares wguide1 asig, xfreq, kcutoff, kfeedback

ares wguide2 asig, xfreq1, xfreq2, kcutoff1, kcutoff2, \

```



kfeedback1, kfeedback2

### Modificateurs de signal : distorsion non-linéaire.

```
aout chebyshevpoly ain, k0 [, k1 [, k2 [...]]]
aout pdclip ain, kWidth, kCenter [, ibipolar [, ifullscale]]
aout pdhalf ain, kShapeAmount [, ibipolar [, ifullscale]]
aout pdhalfy ain, kShapeAmount [, ibipolar [, ifullscale]]
aout powershape ain, kShapeAmount [, ifullscale]
```

### Modificateurs de signal : comparateurs et accumulateurs.

```
aout cmp a1, S_operator, a2
aout cmp a1, S_operator, kx
kOut[] cmp kA, S_operator, kB
kOut[] cmp k1, S_operator1, kIn[], S_operator2, k2

amax max ain1, ain2 [, ain3] [, ain4] [...]
kmax max kin1, kin2 [, kin3] [, kin4] [...]
imax max iin1, iin2 [, iin3] [, iin4] [...]

knumkout max_k asig, ktrig, itype

amax maxabs ain1, ain2 [, ain3] [, ain4] [...]
kmax maxabs kin1, kin2 [, kin3] [, kin4] [...]

maxabsaccum aAccumulator, aInput

maxaccum aAccumulator, aInput

amin min ain1, ain2 [, ain3] [, ain4] [...]
kmin min kin1, kin2 [, kin3] [, kin4] [...]
imin min iin1, iin2 [, iin3] [, iin4] [...]

amin minabs ain1, ain2 [, ain3] [, ain4] [...]
kmin minabs kin1, kin2 [, kin3] [, kin4] [...]

minabsaccum aAccumulator, aInput

minaccum aAccumulator, aInput
```

### Contrôle d'instrument : contrôle d'horloge.

```
clockoff inum
clockon inum
```

### Contrôle d'instrument : valeurs conditionnelles.

```
(a == b ? v1 : v2)
(a >= b ? v1 : v2)
(a > b ? v1 : v2)
(a <= b ? v1 : v2)
(a < b ? v1 : v2)
(a != b ? v1 : v2)
```

### Contrôle d'instrument : compilation.

```
ires compilecsd Sfilename
ires compileorc Sfilename
ires compilestr Sorch
ires evalstr Scode
kres evalstr Scode, ktrig
return ival
```

### Contrôle d'instrument : contrôle de durée.

```
ihold

turnoff
turnoff inst
turnoff knst

turnoff2 kinsno, kmode, krelease

turnon insnum [, itime]
```

### Contrôle d'instrument : appel d'instrument.

```
event "scorechar", kinsnum, kdelay, kdur, [, kp4] [, kp5] [, ...]
event "scorechar", "insname", kdelay, kdur, [, kp4] [, kp5] [, ...]

event_i "scorechar", iinsnum, idelay, idur, [, ip4] [, ip5] [, ...]
event_i "scorechar", "insname", idelay, idur, [, ip4] [, ip5] [, ...]

mute insnum [, iswitch]
mute "insname" [, iswitch]

iHandle nstance insnum, iwhen, idur [, ip4] [, ip5] [...]
iHandle nstance "insname", iwhen, idur [, ip4] [, ip5] [...]

readscore Sin

remove insnum
```

```

schedkwhen ktrigger, kmintim, kmaxnum, kinsnum, kwhen, kdur \
[, ip4] [, ip5] [...]
schedkwhen ktrigger, kmintim, kmaxnum, "insname", kwhen, kdur \
[, ip4] [, ip5] [...]

schedkwhennamed ktrigger, kmintim, kmaxnum, "name", kwhen, kdur \
[, ip4] [, ip5] [...]

schedule insnum, iwhen, idur [, ip4] [, ip5] [...]
schedule "insname", iwhen, idur [, ip4] [, ip5] [...]

schedulek kinsnum, kwhen, kdur [, kp4] [, kp5] [...]
schedulek "insname", kwhen, kdur [, kp4] [, kp5] [...]

schedwhen ktrigger, kinsnum, kwhen, kdur [, ip4] [, ip5] [...]
schedwhen ktrigger, "insname", kwhen, kdur [, ip4] [, ip5] [...]

scoreline Sin, ktrig

scoreline_i Sin

```

### Contrôle d'instrument : contrôle séquentiel d'un programme.

```

cggoto condition, label

cigoto condition, label

ckgoto condition, label

cngoto condition, label

else

elseif xa R xb then

endif

goto label

if ia R ib igoto label
if ka R kb kgoto label
if xa R xb goto label
if xa R xb then

igoto label

kgoto label

loop_ge   indx, idocr, imin, label
loop_ge   kndx, kdecr, kmin, label

loop_gt   indx, idocr, imin, label
loop_gt   kndx, kdecr, kmin, label

loop_le   indx, incr, imax, label
loop_le   kndx, knocr, kmax, label

loop_lt   indx, incr, imax, label
loop_lt   kndx, knocr, kmax, label

tigoto label

timeout istrtr, idur, label

```

```
until condition do
... od

while condition do
... od
```

### Contrôle d'instrument : controle de l'exécution en temps réel.

```
ir active insnum [,iopt [,inorel]]
ir active Sinsname [,iopt [,inorel]]
kres active kinsnum [,iopt [,inorel]]

ktot[,kcpu1, kcpu2,...]cpumeter ifreq

cpuprc insnum, ipercent
cpuprc Sinsname, ipercent

exitnow [ivalue]

jacktransport icommand [, ilocation]

maxalloc insnum, icount
maxalloc Sinsname, icount

prealloc insnum, icount
prealloc "insname", icount
```

### Contrôle d'instrument : initialisation et réinitialisation.

```
ares = xarg
ires = iarg
kres = karg
ires, ... = iarg, ...
kres, ... = karg, ...
table [ kval] = karg

ares init iarg
ires init iarg
kres init iarg
ares, ... init iarg, ...
ires, ... init iarg, ...
kres, ... init iarg, ...
tab init isize[, ival]

insno nstrnum "name"

Sname nstrstrinsno
Sname nstrstrknsno

p(x)

ares += xarg
ires += iarg
kres += karg
table [ kval] += karg

pset icon1 [, icon2] [...]

reinit label
```

**rigoto** label

**rireturn**

ir **tival**

### Contrôle d'instrument : détection et contrôle.

kres **button** knum

ktrig **changed** kvar1 [, kvar2,..., kvarN]

ktrig **changed2** kvar1 [, kvar2,..., kvarN]

ktrig **changed2** karr[]

ktrig **changed2** aarr[]

kres **checkbox** knum

kres **control** knum

ares **follow** asig, idt

ares **follow2** asig, katt, krel

Svalue **getcfig** iopt

kres **joystick** kdevice ktab

ktrig **metro** kfreq [, initphase]

ktrig **metro2** kfreq, kswing [, iamp, initphase]

ksig **midifilestatus**

ksig **miditempo**

**p5gconnect**

kres **p5gdata** kcontrol

icount **pcount**

kres **peak** asig

kres **peak** ksig

ivalue **pindex** ipfieldIndex

koct, kamp **pitch** asig, iupdt, ilo, ihi, idbthresh [, ifrqs] [, iconf] \  
[, istrtr] [, iocts] [, iq] [, inptls] [, irolloff] [, iskip]

kcps, krms **pitchamdf** asig, imincps, imaxcps [, icps] [, imedi] \  
[, idowns] [, iexcps] [, irmsmedi]

acps, alock **plltrack**asig, kd [, kloopf, kloopq, klf, khf, kthresh]

kcps, kamp **ptrack** asig, ihopsize[,ipeaks]

ival **readscratch**[index]

**rewindscore**

kres **rms** asig [, ihp] [, iskip]

```

kres[, kkeydown] sensekey

ktrig_out seqtime ktime_unit, kstart, kloop, kinitndx, kfn_times

ktrig_out seqtime2 ktrig_in, ktime_unit, kstart, kloop, kinitndx, kfn_times

setctrl inum, ival, itype

setscorepos ipos

splitrig ktrig, kndx, imaxtics, ifn, kout1 [,kout2,...,koutN]

ktemp tempest kin, iprd, imindur, imemdur, ihp, ithresh, ihtim, ixfdbak, \
    istartempo, ifn [, idisprd] [, itweek]

tempo ktempo, istartempo

kres tempoval

ktrig timedseq ktempnt, ifn, kp1 [,kp2, kp3, ...,kpN]

kout trigger ksig, kthreshold, kmode

trigseq ktrig_in, kstart, kloop, kinitndx, kfn_values, kout1 [, kout2] [...]

ares vactrol asig [iup, idown]

ires wiiconnect [itimeout, imaxnum]

kres wiidata kcontrol[, knum]

wiirange icontrol, iminimum, imaximum[, inum]

kres wiisend kcontrol, kvalue[, knum]

writescratchival[, index]

kx, ky xyin iprd, ixmin, ixmax, iymin, ymax [, ixinit] [, iyinit]

```

### Contrôle d'instrument : piles.

```

xval1, [xval2, ... , xval31] pop
ival1, [ival2, ... , ival31] pop

fsig pop_f

push xval1, [xval2, ... , xval31]
push ival1, [ival2, ... , ival31]

push_f fsig

stack iStackSize

```

### Contrôle d'instrument : contrôle de sous-instrument.

```

a1, [...] [, a8] subinstr instrnum [, p4] [, p5] [...]
a1, [...] [, a8] subinstr "insname" [, p4] [, p5] [...]

subinstrinit instrnum [, p4] [, p5] [...]
subinstrinit "insname" [, p4] [, p5] [...]

```

### Contrôle d'instrument : lecture du temps.

```

ir[, inano] date
kr[, knano] date

Sir dates [ itime]

ir readclock inum

ires rtclock
kres rtclock

kres timeinstk

kres timeinsts

ires timek
kres timek

ires times
kres times

```

### Opcodes jacko.

```

asignal JackoAudioIn ScsoundPortName

JackoAudioInConnect SexternalPortName, ScsoundPortName

JackoAudioOut ScsoundPortName, asignal

JackoAudioOutConnect ScsoundPortName, SexternalPortName

JackoFreewheel [ienabled]

JackoInfo

JackoInit ServerName, SclientName

JackoMidiInConnect SexternalPortName, ScsoundPortName

JackoMidiOut ScsoundPortName, kstatus, kchannel, kdata1[, kdata2]

JackoMidiOutConnect ScsoundPortName, SexternalPortName

JackoNoteOut ScsoundPortName, kstatus, kchannel, kdata1[, kdata2]

JackoOn [iactive]

JackoTransport kcommand, [kposition]

```

### Contrôle des tables de fonction.

```

ftfree ifno, iwhen

gir ftgen ifn, itime, isize, igen, iarga [, iargb ] [...]
gir ftgen ifn, itime, isize, igen, iarray

```

```
ifno ftgentmp ip1, ip2dummy, isize, igen, iarga, iargb, ...

Sdst getftargs iftno, ktrig

sndload Sfname[, ifmt[, ichns[, isr[, ibas[, iamp[, istr \
    [, ilpmod[, ilps[, ilpe]]]]]]]]]
```

### Contrôle des tables de fonction : requêtes sur une table.

```
karray[] array ival1, ival2, ..., ivaln

karray[] fillarray ival1, ival2,....ivaln
karray fillarray ival1, ival2,....ivaln
karray fillarray kval1, kval2,....kvaln

ftchnls(x) (arg de taux-i seulement)

ftcps(x) (args de taux-i seulement)

iexists ftexists ifn
kexists ftexists kfn / ifn

ftlen(x) (arg de taux-i seulement)

ftlptim(x) (arg de taux-i seulement)

ftsr(x) (arg de taux-i seulement)

karray genarray kstart, kend[, inc]
iarray genarray istart, iens[, inc]

karray genarray_i istart, iend[, inc]

ir lenarray karray[, iwhich]
kr lenarray karray[, iwhich]

karray maparray kinarray, String
karray maparray_i kinarray, String

nsamp(x) (arg de taux-i seulement)

karray slicearray kinarray, istart, iend [,istride]

ires tbleng ifn
kres tbleng kfn

kr tabsum ifn[[, kmin] [, kmax]]

tb0_init ifn
tb1_init ifn
tb2_init ifn
tb3_init ifn
tb4_init ifn
tb5_init ifn
tb6_init ifn
tb7_init ifn
tb8_init ifn
tb9_init ifn
tb10_init ifn
tb11_init ifn
tb12_init ifn
tb13_init ifn
tb14_init ifn
```



```

tb15_init ifn
iout = tb0(iIndex)
kout = tb0(kIndex)
iout = tb1(iIndex)
kout = tb1(kIndex)
iout = tb2(iIndex)
kout = tb2(kIndex)
iout = tb3(iIndex)
kout = tb3(kIndex)
iout = tb4(iIndex)
kout = tb4(kIndex)
iout = tb5(iIndex)
kout = tb5(kIndex)
iout = tb6(iIndex)
kout = tb6(kIndex)
iout = tb7(iIndex)
kout = tb7(kIndex)
iout = tb8(iIndex)
kout = tb8(kIndex)
iout = tb9(iIndex)
kout = tb9(kIndex)
iout = tb10(iIndex)
kout = tb10(kIndex)
iout = tb11(iIndex)
kout = tb11(kIndex)
iout = tb12(iIndex)
kout = tb12(kIndex)
iout = tb13(iIndex)
kout = tb13(kIndex)
iout = tb14(iIndex)
kout = tb14(kIndex)
iout = tb15(iIndex)
kout = tb15(kIndex)

```

### Contrôle des tables de fonction : sélection dynamique.

```

ares tableikt xndx, kfn [, ixmode] [, ixoff] [, iwrap]
kres tableikt xndx, kfn [, ixmode] [, ixoff] [, iwrap]

ares tablekt xndx, kfn [, ixmode] [, ixoff] [, iwrap]
kres tablekt xndx, kfn [, ixmode] [, ixoff] [, iwrap]

ares tablexkt xndx, kfn, kwarp, iwsiz [, ixmode] [, ixoff] [, iwrap]

```

### Contrôle des tables de fonction : opérations de lecture/écriture.

```

ians ftaudio ifn, "filename", iformat
kans ftaudio ktrig, kfn, "filename", \
                                     kformat [, isync, kbeg, kend]

ftload Sfilename, iflag, ifn1 [, ifn2] [...]

ftloadk Sfilename, ktrig, iflag, ifn1 [, ifn2] [...]

ftprint ifn [, ktrig, kstart, kend, kstep, inumcols ]

iNumberOfFile ftsamplebank SDirectory, iFirstTableNumber, iSkipTime, iFormat, iChannel,
kNumberOfFile ftsamplebank SDirectory, kFirstTableNumber, kTrigger, kSkipTime, kFormat,
kChannel,

```

```

ftsave "filename", iflag, ifn1 [, ifn2] [...]

ftsavk "filename", ktrig, iflag, ifn1 [, ifn2] [...]

ptablew asig, andx, ifn [, ixmode] [, ixoff] [, iwgmde]
ptablew isig, indx, ifn [, ixmode] [, ixoff] [, iwgmde]
ptablew ksig, kndx, ifn [, ixmode] [, ixoff] [, iwgmde]

tablecopy kdft, ksft

knumpassed tablefilter kouttable, kintatble, kmode, kparam

inumpassed tablefilteri iouttable, iintatble, imode, iparam

tablegpw kfn

tableicopy idft, isft

tableigpw ifn

tableimix idft, idoff, ilen, islft, isloff, islg, is2ft, is2off, is2g

tablemix kdft, kdoff, klen, kslft, ksloff, kslg, ks2ft, ks2off, ks2g

ares tablera kfn, kstart, koff

tablew asig, andx, ifn [, ixmode] [, ixoff] [, iwgmde]
tablew isig, indx, ifn [, ixmode] [, ixoff] [, iwgmde]
tablew ksig, kndx, ifn [, ixmode] [, ixoff] [, iwgmde]

kstart tablewa kfn, asig, koff

tablewkt asig, andx, kfn [, ixmode] [, ixoff] [, iwgmde]
tablewkt ksig, kndx, kfn [, ixmode] [, ixoff] [, iwgmde]

kout tabmorph kindex, kweightpoint, ktabnum1, ktabnum2, \
    ifn1, ifn2 [, ifn3, ifn4, ...,ifnN]

aout tabmorpha aindex, aweightpoint, atabnum1, atabnum2, \
    ifn1, ifn2 [, ifn3, ifn4, ... ifnN]

aout tabmorphak aindex, kweightpoint, ktabnum1, ktabnum2, \
    ifn1, ifn2 [, ifn3, ifn4, ... ifnN]

kout tabmorphi kindex, kweightpoint, ktabnum1, ktabnum2, \
    ifn1, ifn2 [, ifn3, ifn4, ... ifnN]

tabplay ktrig, knumtics, kfn, kout1 [,kout2,..., koutN]

tabrec ktrig_start, ktrig_stop, knumtics, kfn, kin1 [,kin2,...,kinN]

```

## FLTK : conteneurs.

```

FLgroup "label", iwidth, iheight, ix, iy [, iborder] [, image]

FLgroupEnd

FLpack iwidth, iheight, ix, iy, itype, ispace, iborder

FLpackEnd

FLpanel "label", iwidth, iheight [, ix] [, iy] [, iborder] [, ikbdcapture] [, iclose]

```

**FLpanelEnd**

**FLscroll** iwidth, iheight [, ix] [, iy]

**FLscrollEnd**

**FLtabs** iwidth, iheight, ix, iy

**FLtabsEnd**

## FLTK : valueurs.

kout, ihandle **FLcount** "label", imin, imax, istep1, istep2, itype, \  
iwidth, iheight, ix, iy, iopcode [, kp1] [, kp2] [, kp3] [...] [, kpN]

koutx, kouty, ihandlex, ihandley **FLjoy** "label", iminx, imaxx, iminy, \  
imaxy, iexp, iexpy, idisp, idispy, iwidth, iheight, ix, iy

kout, ihandle **FLknob** "label", imin, imax, iexp, itype, idisp, iwidth, \  
ix, iy [, icursorsize]

kout, ihandle **FLroller** "label", imin, imax, istep, iexp, itype, idisp, \  
iwidth, iheight, ix, iy

kout, ihandle **FLslider** "label", imin, imax, iexp, itype, idisp, iwidth, \  
iheight, ix, iy

kout, ihandle **FLtext** "label", imin, imax, istep, itype, iwidth, \  
iheight, ix, iy

## FLTK : autres.

ihandle **FLbox** "label", itype, ifont, isize, iwidth, iheight, ix, iy [, image]  
ihandle **FLbox** istr, itype, ifont, isize, iwidth, iheight, ix, iy [, image]

kout, ihandle **FLbutBank** itype, inumx, inumy, iwidth, iheight, ix, iy, \  
iopcode [, kp1] [, kp2] [, kp3] [, kp4] [, kp5] [...] [, kpN]

kout, ihandle **FLbutton** "label", ion, ioff, itype, iwidth, iheight, ix, \  
iy, iopcode [, kp1] [, kp2] [, kp3] [, kp4] [, kp5] [...] [, kpN]

ihandle **FLcloseButton** "label", iwidth, iheight, ix, iy

ihandle **FLexecButton** "command", iwidth, iheight, ix, iy

inumsnap **FLgetsnap** index [, igroup]

ihandle **FLhvsBox** inumlinesX, inumlinesY, iwidth, iheight, ix, iy

**FLhvsBox** kx, ky, ihandle

kascii **FLkeyIn** [ifn]

**FLloadsnap** "filename" [, igroup]

kx, ky, kb1, kb2, kb3 **FLmouse** [imode]

**FLprintk** itime, kval, idisp

**FLprintk2** kval, idisp

## FLrun

**FLsavesnap** "filename" [, igroup]

inumsnap, inumval **FLsetsnap** index [, ifn, igroup]

**FLsetSnapGroup** igroup

**FLsetVal** ktrig, kvalue, ihandle

**FLsetVal\_i** ivalue, ihandle

**FLslidBnk** "names", inumsliders [, ioutable] [, iwidth] [, iheight] [, ix] \  
[, iy] [, itypetable] [, iexptable] [, istart\_index] [, iminmaxtable]

**FLslidBnk2** "names", inumsliders, ioutable, iconfigtable [,iwidth, iheight, ix, iy, istart\_index]

**FLslidBnk2** istring, inumsliders, ioutable, iconfigtable [,iwidth, iheight, ix, iy, istart\_index]

**FLslidBnk2Set** ihandle, ifn [, istartIndex, istartSlid, inumSlid]

**FLslidBnk2Setk** ktrig, ihandle, ifn [, istartIndex, istartSlid, inumSlid]

ihandle **FLslidBnkGetHandle**

**FLslidBnkSet** ihandle, ifn [, istartIndex, istartSlid, inumSlid]

**FLslidBnkSetk** ktrig, ihandle, ifn [, istartIndex, istartSlid, inumSlid]

## FLupdate

ihandle **FLvalue** "label", iwidth, iheight, ix, iy

**FLvkeybd** "keyboard.map", iwidth, iheight, ix, iy

**FLvslidBnk** "names", inumsliders [, ioutable] [, iwidth] [, iheight] [, ix] \  
[, iy] [, itypetable] [, iexptable] [, istart\_index] [, iminmaxtable]

**FLvslidBnk2** "names", inumsliders, ioutable, iconfigtable [,iwidth, iheight, ix, iy, istart\_index]

koutx, kouty, kinside **FLxyin** ioutx\_min, ioutx\_max, iouty\_min, iouty\_max, \  
iwindx\_min, iwindx\_max, iwindy\_min, iwindy\_max [, iexp\_x, iexp\_y, ioutx, iouty]

**vphaseseg** kphase, ioutab, ielems, itab1, idist1, itab2 \  
[,idist2, itab3, ... ,idistN-1, itabN]

## FLTK : apparence.

**FLcolor** ired, igreen, iblue [, ired2, igreen2, iblue2]

**FLcolor2** ired, igreen, iblue

**FLhide** ihandle

**FLlabel** isize, ifont, ialign, ired, igreen, iblue

**FLsetAlign** ialign, ihandle

**FLsetBox** itype, ihandle

**FLsetColor** ired, igreen, iblue, ihandle

```

FLsetColor2 ired, igreen, iblue, ihandle

FLsetFont ifont, ihandle

FLsetPosition ix, iy, ihandle

FLsetSize iwidth, iheight, ihandle

FLsetText "itext", ihandle
FLsetText istr, ihandle

FLsetTextColor ired, iblue, igreen, ihandle

FLsetTextSize isize, ihandle

FLsetTextType itype, ihandle

FLshow ihandle

```

### Opérations mathématiques : opérations arithmétiques et logiques.

```

a + b (no rate restriction)

a / b (no rate restriction)

a % b (no rate restriction)

a * b (no rate restriction)

a && b (ET logique ; pas de taux audio)

a & b (ET binaire)

~ a (NON binaire)

a | b (bitwise OR)

a << b (décalage binaire à gauche)

a >> b (décalage binaire à droite)

a # b (NON-EQUIVALENCE binaire)

! a (NON logique ; pas de taux audio)

a || b (logical OR; not audio-rate)

a ^ b (b not audio-rate)

a # b (no rate restriction)

```

### Opérations mathématiques : comparateurs et accumulateurs.

```

clear avar1 [, avar2] [, avar3] [...]

vincr accum, aincr

```

## Opérations mathématiques : fonctions mathématiques.

**abs**(x) (pas de restriction de taux)  
**abs**(k/i[]) (k- ou i-tableau)

**ceil**(x) (argument au taux d'initialisation, de contrôle ou audio)  
**ceil**(k/i[]) (k- ou i-tableau)

**exp**(x) (pas de restriction de taux)  
**exp**(k/i[]) (k- ou i-tableau)

**floor**(x) (argument au taux d'initialisation, de contrôle ou audio)  
**floor**(k/i[]) (k- ou i-tableau)

**frac**(x) (arguments de taux-i ou de taux-k ; fonctionne aussi au taux-a dans Csound5)  
**frac**(k/i[]) (k- ou i-tableau)

**int**(x) (taux-i ou taux-k ; fonctionne aussi au taux-a dans Csound5)

**log**(x) (pas de restriction de taux)  
**log**(k/i[]) (k- ou i-tableau)  
kout[]**log** kin[][,ibas]

**log10**(x) (pas de restriction de taux)  
**log10**(k/i[]) (k- ou i-tableau)

**log2**(x) (pas de restriction de taux)  
**log2**(k/i[]) (k- ou i-tableau)

**logbtwo**(x) (argument au taux d'initialisation ou de contrôle seulement)

**powoftwo**(x) (argument au taux d'initialisation ou de contrôle seulement)

**qinf**(x) (aucune restriction de taux)

**qnan**(x) (aucune restriction de taux)

**round**(x) (des arguments de taux-i, -k ou -a sont permis)  
**round**(k/i[]) (k- ou i-tableau)

**sqrt**(x) (pas de restriction de taux)  
**sqrt**(k/i[]) (k- ou i-tableau)

## Opérations mathématiques : fonctions trigonométriques.

**cos**(x) (pas de restriction de taux)  
**cos**(k/i[]) (k- ou i-tableau)

**cosh**(x) (pas de restriction de taux)  
**cosh**(k/i[]) (k- ou i-tableau)

**cosinv**(x) (pas de restriction de taux)  
**cosinv**(k/i[]) (k- ou i-tableau)

**signum**(x) (aucune restriction de taux)

**sin**(x) (pas de restriction de taux)  
**sin**(k/i[]) (k- ou i-tableau)

**sinh**(x) (pas de restriction de taux)  
**sinh**(k/i[]) (k- ou i-tableau)

**sininv**(x) (pas de restriction de taux)

`sininv`(k/i[]) (k- ou i-tableau)

`tan`(x) (pas de restriction de taux)

`tan`(k/i[]) (k- ou i-tableau)

`tanh`(x) (pas de restriction de taux)

`tanh`(k/i[]) (k- ou i-tableau)

`taninv`(x) (pas de restriction de taux)

`taninv`(k/i[]) (k- ou i-tableau)

### Opérations mathématiques : fonctions d'amplitude.

`ampdb`(x) (pas de restriction de taux)

`ampdbfs`(x) (pas de restriction de taux)

`db`(x)

`dbamp`(x) (arguments de taux-i ou -k seulement)

`dbfsamp`(x) (arguments de taux-i ou -k seulement)

### Opérations mathématiques : fonctions aléatoires.

`birnd`(x) (taux-i ou -k seulement)

`rnd`(x) (taux-i ou -k seulement)

### Opérations mathématiques : opcodes équivalents à des fonctions.

ares `divz` xa, xb, ksubst

ires `divz` ia, ib, isubst

kres `divz` ka, kb, ksubst

ares `mac` ksig1, asig1 [, ksig2] [, asig2] [, ksig3] [, asig3] [...]

ares `maca` asig1 , asig2 [, asig3] [, asig4] [, asig5] [...]

aout `polynomial` ain, k0 [, k1 [, k2 [...]]]

ares `pow` aarg, kpow [, inorm]

ires `pow` iarg, ipow [, inorm]

kres `pow` karg, kpow [, inorm]

ires[] `pow` iarg[], ipow[]

kres[] `pow` karg[], kpow[]

ires[] `pow` iarg[], ipow

kres[] `pow` karg[], kpow

ares `product` asig1, asig2 [, asig3] [...]

ares `sum` asig1 [, asig2] [, asig3] [...]

kres `sum` karr

ires `sum` iarr

ares `taninv2` ay, ax

```
ires taninv2 iy, ix
kres taninv2 ky, kx
```

### Conversion des hauteurs : fonctions.

```
cent(x)

cpsmidinn (MidiNoteNumber) (arguments de taux-i ou -k seulement)

cpsoct (oct) (pas de restriction de taux)

cpspch (pch) (arguments de taux-i ou -k seulement)

imidi ftom ifreq [,irnd]
kmidi ftom kfreq [,irnd]
imidis[] ftom ifreqs[] [,irnd]
kmidis[] ftom kfreqs[] [,irnd]

ifreq mtof imidi
kfreq mtof kmidi
ifreqs[] mtof imidis[]
kfreqs[] mtof kmidis[]

Snote mton kmidi
Snote mton imidi

kfreq ntof Snote
ifreq ntof Snote

kmidi ntom Snote
imidi ntom Snote

octave(x)

octcps (cps) (arguments de taux-i ou -k seulement)

octmidinn (MidiNoteNumber) (arguments de taux-i ou -k seulement)

octpch (pch) (arguments de taux-i ou -k seulement)

pchmidinn (MidiNoteNumber) (arguments de taux-i ou -k seulement)

pchoct (oct) (arguments de taux-i ou -k seulement)

imidi pchtom ipch
kmidi pchtom kpch

semitone(x)
```

### Conversion des hauteurs : opcodes de hauteurs.

```
icps cps2pch ipch, iequal
kcps cpstun ktrig, kindex, kfn
icps cpstuni index, ifn
icps cpsxpch ipch, iequal, irepeat, ibase
```



## MIDI en temps réel : entrée.

```

kaft aftouch [imin] [, imax]

ival chanctrl ichnl, ictlno [, ilow] [, ihigh]
kval chanctrl ichnl, ictlno [, ilow] [, ihigh]

idest ctrl14 ichan, ictlno1, ictlno2, imin, imax [, ifn]
kdest ctrl14 ichan, ictlno1, ictlno2, kmin, kmax [, ifn]

idest ctrl21 ichan, ictlno1, ictlno2, ictlno3, imin, imax [, ifn]
kdest ctrl21 ichan, ictlno1, ictlno2, ictlno3, kmin, kmax [, ifn]

idest ctrl7 ichan, ictlno, imin, imax [, ifn]
kdest ctrl7 ichan, ictlno, kmin, kmax [, ifn]
adest ctrl7 ichan, ictlno, kmin, kmax [, ifn] [, icutoff]

ctrlinit ichnl, ictlno1, ival1 [, ictlno2] [, ival2] [, ictlno3] \
    [, ival3] [, ...ival32]

initc14 ichan, ictlno1, ictlno2, ivalue

initc21 ichan, ictlno1, ictlno2, ictlno3, ivalue

initc7 ichan, ictlno, ivalue

massign ichnl, insnum[, ireset]
massign ichnl, "insname"[, ireset]

idest midic14 ictlno1, ictlno2, imin, imax [, ifn]
kdest midic14 ictlno1, ictlno2, kmin, kmax [, ifn]

idest midic21 ictlno1, ictlno2, ictlno3, imin, imax [, ifn]
kdest midic21 ictlno1, ictlno2, ictlno3, kmin, kmax [, ifn]

idest midic7 ictlno, imin, imax [, ifn]
kdest midic7 ictlno, kmin, kmax [, ifn]

ival midictrl inum [, imin] [, imax]
kval midictrl inum [, imin] [, imax]

ival notnum

ibend pchbend [imin] [, imax]
kbend pchbend [imin] [, imax]

pgmassign ipgm, inst[, ichn]
pgmassign ipgm, "insname"[, ichn]

ires polyaft inote [, ilow] [, ihigh]
kres polyaft inote [, ilow] [, ihigh]

ival veloc [ilow] [, ihigh]

```

## MIDI en temps réel : sortie.

```

nrpn kchan, kparmnum, kparmvalue

outiat ichn, ivalue, imin, imax

outic ichn, inum, ivalue, imin, imax

```

**outic14** ichn, imsb, ilsb, ivalue, imin, imax  
**outipat** ichn, inotenum, ivalue, imin, imax  
**outipb** ichn, ivalue, imin, imax  
**outipc** ichn, iprog, imin, imax  
**outkat** kchn, kvalue, kmin, kmax  
**outkc** kchn, knum, kvalue, kmin, kmax  
**outkc14** kchn, kmsb, klsb, kvalue, kmin, kmax  
**outkpat** kchn, knotenum, kvalue, kmin, kmax  
**outkpb** kchn, kvalue, kmin, kmax  
**outkpc** kchn, kprog, kmin, kmax

### MIDI en temps réel : convertisseurs.

iamp **ampmidi** iscal [, ifn]  
  
 igain **ampmidicurve** ivelocity, idynamicrange, iexponent  
 kgain **ampmidicurve** kvelocity, kdynamicrange, kexponent  
  
 iamplitude **ampmidid** ivelocity, idecibels  
 kamplitude **ampmidid** kvelocity, idecibels  
  
 icps **cpsmidi**  
  
 icps **cpsmidib** [irange]  
 kcps **cpsmidib** [irange]  
  
 icps **cpstmid** ifn  
  
 ioct **octmidi**  
  
 ioct **octmidib** [irange]  
 koct **octmidib** [irange]  
  
 ipch **pchmidi**  
  
 ipch **pchmidib** [irange]  
 kpch **pchmidib** [irange]

### MIDI en temps réel : E/S génériques.

kstatus, kchan, kdata1, kdata2 **midiin**  
  
**midiout** kstatus, kchan, kdata1, kdata2  
  
**midiout\_i** istatus, ichan, idata1, idata2

### MIDI en temps réel : extension d'évènements.

kflag **lastcycle**  
 kflag **release**  
**xtratim** iextradur

### **MIDI en temps réel : sortie de note.**

kMidiNoteNum, kCounter**midiarp** kRate[, kMode]  
**midion** kchn, knum, kvel  
**midion2** kchn, knum, kvel, ktrig  
**moscil** kchn, knum, kvel, kdur, kpause  
**noteoff** ichn, inum, ivel  
**noteon** ichn, inum, ivel  
**noteondur** ichn, inum, ivel, idur  
**noteondur2** ichn, inum, ivel, idur

### **MIDI en temps réel : interopérabilité MIDI/partition.**

**midichannelaftertouch** xchannelaftertouch [, ilow] [, ihigh]  
 ichn **midichn**  
**midicontrolchange** xcontroller, xcontrollervalue [, ilow] [, ihigh]  
**mididefault** xdefault, xvalue  
**midinoteoff** xkey, xvelocity  
**midinoteoncps** xcps, xvelocity  
**midinoteonkey** xkey, xvelocity  
**midinoteonoct** xoct, xvelocity  
**midinoteonpch** xpch, xvelocity  
**midipitchbend** xpitchbend [, ilow] [, ihigh]  
**midipolyaftertouch** xpolyaftertouch, xcontrollervalue [, ilow] [, ihigh]  
**midiprogramchange** xprogram

### **MIDI en temps réel : système temps réel.**

**mclock** ifreq  
**mrtmsg** imsgtype

## MIDI en temps réel : banques de réglettes.

```

i1,...,i16 sl6b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \
    initvalue1, ifn1,..., ictlno_msb16, ictlno_lsb16, imin16, imax16, initvalue16,
    ifn16
k1,...,k16 sl6b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \
    initvalue1, ifn1,..., ictlno_msb16, ictlno_lsb16, imin16, imax16, initvalue16,
    ifn16

i1,...,i32 s32b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \
    initvalue1, ifn1,..., ictlno_msb32, ictlno_lsb32, imin32, imax32, initvalue32,
    ifn32
k1,...,k32 s32b14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, \
    initvalue1, ifn1,..., ictlno_msb32, ictlno_lsb32, imin32, imax32, initvalue32,
    ifn32

i1,...,i16 slider16 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
    ictlnum16, imin16, imax16, init16, ifn16
k1,...,k16 slider16 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
    ictlnum16, imin16, imax16, init16, ifn16

k1,...,k16 slider16f ichan, ictlnum1, imin1, imax1, init1, ifn1, \
    icutoff1,..., ictlnum16, imin16, imax16, init16, ifn16, icutoff16

kflag slider16table ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
    init1, ifn1, .... , ictlnum16, imin16, imax16, init16, ifn16

kflag slider16tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
    init1, ifn1, icutoff1, .... , ictlnum16, imin16, imax16, init16, ifn16, icutoff16

i1,...,i32 slider32 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
    ictlnum32, imin32, imax32, init32, ifn32
k1,...,k32 slider32 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
    ictlnum32, imin32, imax32, init32, ifn32

k1,...,k32 slider32f ichan, ictlnum1, imin1, imax1, init1, ifn1, icutoff1, \
    ..., ictlnum32, imin32, imax32, init32, ifn32, icutoff32

kflag slider32table ichan, ioutTable, ioffset, ictlnum1, imin1, \
    imax1, init1, ifn1, .... , ictlnum32, imin32, imax32, init32, ifn32

kflag slider32tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
    init1, ifn1, icutoff1, .... , ictlnum32, imin32, imax32, init32, ifn32, icutoff32

i1,...,i64 slider64 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
    ictlnum64, imin64, imax64, init64, ifn64
k1,...,k64 slider64 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
    ictlnum64, imin64, imax64, init64, ifn64

k1,...,k64 slider64f ichan, ictlnum1, imin1, imax1, init1, ifn1, \
    icutoff1,..., ictlnum64, imin64, imax64, init64, ifn64, icutoff64

kflag slider64table ichan, ioutTable, ioffset, ictlnum1, imin1, \
    imax1, init1, ifn1, .... , ictlnum64, imin64, imax64, init64, ifn64

kflag slider64tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
    init1, ifn1, icutoff1, .... , ictlnum64, imin64, imax64, init64, ifn64, icutoff64

i1,...,i8 slider8 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
    ictlnum8, imin8, imax8, init8, ifn8
k1,...,k8 slider8 ichan, ictlnum1, imin1, imax1, init1, ifn1,..., \
    ictlnum8, imin8, imax8, init8, ifn8

k1,...,k8 slider8f ichan, ictlnum1, imin1, imax1, init1, ifn1, icutoff1, \

```

```

..., ictlnum8, imin8, imax8, init8, ifn8, icutoff8

kflag slider8table ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
    init1, ifn1,..., ictlnum8, imin8, imax8, init8, ifn8

kflag slider8tablef ichan, ioutTable, ioffset, ictlnum1, imin1, imax1, \
    init1, ifn1, icutoff1, .... , ictlnum8, imin8, imax8, init8, ifn8, icutoff8

k1, k2, ..., k16 sliderKawai imin1, imax1, init1, ifn1, \
    imin2, imax2, init2, ifn2, ..., imin16, imax16, init16, ifn16

```

## Graphe de fluence.

```

alwayson Tinstrument [p4, ..., pn]

connect Tsource1, Soutlet1, Tsink1, Sinlet1

ifno ftgenonce ipldummy, ip2dummy, isize, igen, iarga, iargb, ...

asignal inleta Sname

fsignal inletf Sname

ksignal inletk Sname

ksignal inletkid Sname, SinstanceID

array inletv Sname

outleta Sname, asignal

outletf Sname, fsignal

outletk Sname, ksignal

outletkid Sname, SinstanceID, ksignal

outletv Sname, array

```

## Traitement spectral : STFT.

```

ares pvadd ktmpnt, kfmod, ifilcod, ifn, ibins [, ibinoffset] \
    [, ibinincr] [, iextractmode] [, ifreqlim] [, igatefn]

pvbufread ktmpnt, ifile

ares pvcross ktmpnt, kfmod, ifile, kampscale1, kampscale2 [, ispecwp]

ares pvinterp ktmpnt, kfmod, ifile, kfreqscale1, kfreqscale2, \
    kampscale1, kampscale2, kfreqinterp, kampinterp

ares pvoc ktmpnt, kfmod, ifilcod [, ispecwp] [, iextractmode] \
    [, ifreqlim] [, igatefn]

kfreq, kamp pvread ktmpnt, ifile, ibin

tableseg ifn1, idur1, ifn2 [, idur2] [, ifn3] [...]

tablexseg ifn1, idur1, ifn2 [, idur2] [, ifn3] [...]

```

```
ares vpvoc ktmpnt, kfmmod, ifile [, ispecwp] [, ifn]
```

### Traitement spectral : LPC.

```
ares lpfreson asig, kfrqratio

lpinterp islot1, islot2, kmix

krmsr, krmso, kerr, kcps lpread ktmpnt, ifilcod [, inpoles] [, ifrmrate]

ares lpreson asig

lpslot islot
```

### Traitement spectral : non-standard.

```
wsig specaddm wsig1, wsig2 [, imul2]

wsig specdiff wsigin

specdisp wsig, iprd [, iwtflg]

wsig specfilt wsigin, ifhtim

wsig spechist wsigin

kocf, kamp specptrk wsig, kvar, ilo, ihi, istr, idbthresh, inptls, \
    irolloff [, ioddd] [, iconfs] [, interp] [, ifprd] [, iwtflg]

wsig specscal wsigin, ifscale, ifthresh

ksum specsum wsig [, interp]

wsig spectrum xsig, iprd, iocts, ifrqa [, iq] [, ihann] [, idbout] \
    [, idsprd] [, idsinrs]
```

### Traitement spectral : streaming.

```
fsg binit fin, isize

fsg cudanal ain, ifftsize, ioverlap, iwinsize, iwintype [, iformat] [, iinit]

asig cudasliding ain, amod, iwinsize

asig cudasynth kamp, kfreq, itab, iftab, iatab[, inum]
asig cudasynth fsg, kamp, kfreq[, inum]
asig cudasynth fsg

ftrks partials ffr, fphs, kthresh, kminpts, kmaxgap, imaxtracks

kframe pvs2tab tvar|kvar[], fsg
kframe pvs2tab kmags[], kfreqs[], fsg

ares pvsadsyn fsrsc, inoscs, kfmmod [, ibinoffset] [, ibinincr] [, iinit]

fsg pvsanal ain, ifftsize, ioverlap, iwinsize, iwintype [, iformat] [, iinit]
```

```

fsig pvsarp fsigin, kbin, kdepth, kgain

fsig pvsbandp fsigin, xlowcut, xlowfull, \
    xhighfull, xhighcut[, ktype]

fsig pvsbandr fsigin, xlowcut, xlowfull, \
    xhighfull, xhighcut[, ktype]

kamp, kfr pvsbin fsig, kbin

fsig pvsblur fsigin, kblurtime, imaxdel

ihandle, ktime pvsbuffer fsig, ilen

fsig pvsbufread ktime, khandle[, ilo, ihi, iclear]

fsig pvsbufread2 ktime, khandle, ift1, ift2

fsig pvscale fsigin, kscal[, kkeepform, kgain, kcoefs]

kcent pvscent fsig
acent pvscent fsig

keps[] pvsceps fsig[, icoefs]

fsig pvsccross fsrc, fdest, kamp1, kamp2

fsig pvsdemix fleft, fright, kpos, kwidth, ipoints

fsig pvsdiskin SFname, ktscal, kgain[, ioffset, ichan]

pvsdisp fsig[, ibins, iwtflg]

fsig pvsfilter fsigin, fsigfil, kdepth[, igain]

fsig pvsfread ktimpt, ifn [, ichan]

fsig pvsfreeze fsigin, kfreeza, kfreezf

pvsftr fsrc, ifna [, ifnf]

kflag pvsftw fsrc, ifna [, ifnf]

pvsfwrite fsig, ifile

fsig pvsgain fsigin, kgain

fsig pvsshift fsigin, kshift, klowest[, kkeepform, igain, kcoefs]

ffr, fphs pvsifd ain, ifftsize, ihopsize, iwintype[, iscal]

fsig pvsin kchan[, isize, iolap, iwinsize, iwintype, iformat]

ioverlap, inumbins, iwinsize, iformat pvsinfo fsrc

fsig pvsinit isize[, iolap, iwinsize, iwintype, iformat]

fsig pvslock fsigin, klock

fsig pvsmaska fsrc, ifn, kdepth

fsig pvsmix fsigin1, fsigin2

fsig pvssmooth fsigin, kacf, kcf

fsig pvsmorph fsig1, fsig2, kampint, kfrqint

```

```
fsig pvsosc kamp, kfreq, ktype, isize [,ioverlap] [, iwinsize] [, iwintype] [, iformat]
pvsout fsig, kchan
kfr, kamp pvspitch fsig, kthresh
fsig pvstanal ktimescal, kamp, kpitch, ktab, [kdetect, kwrap, ioffset, ifftsize, ihop,
idbthresh]
fsig pvstencil fsigin, kgain, klevel, iftable
fsig pvstrace fsigin, kn
fsig, kBins[] pvstrace fsigin, kn[, isort]
fsig pvsvoc famp, fexc, kdepth, kgain [,kcoefs]
fsig pvs warp fsigin, kscal, kshift[, klowest, kmeth, kgain, kcoefs]
ares pvsynth fsrc, [iinit]
asig resyn fin, kscal, kpitch, kmaxtracks, ifn
asig sinsyn fin, kscal, kmaxtracks, ifn
fsig tab2pvs tvar|karr[[],ihopsize, iwinsize, iwintype]
fsig tab2pvs kmags[], kfreqs[[],ihopsize, iwinsize, iwintype]
ffr,fphs tabifd ktimpt, kamp, kpitch, ifftsize, ihopsize, iwintype,ifn
asig tradsyn fin, kscal, kpitch, kmaxtracks, ifn
fsig trcross fin1, fin2, ksearch, kdepth [, kmode]
fsig trfilter fin, kamnt, ifn
fsig, kfr, kamp trhighest fin1, kscal
fsig, kfr, kamp trlowest fin1, kscal
fsig trmix fin1, fin2
fsig trscale fin, kpitch [, kgain]
fsig trshift fin, kpshift [, kgain]
fsiglow, fsighi trsplitt fin, ksplit [, kgainlow, kgainhigh]
```

### Traitement spectral : ATS.

```
ar ATSadd ktimepnt, kfmmod, iatsfile, ifn, ipartials[, ipartialoffset, \
ipartialincr, igatefn]
ar ATSaddnz ktimepnt, iatsfile, ibands[, ibandoffset, ibandincr]
ATSbufread ktimepnt, kfmmod, iatsfile, ipartials[, ipartialoffset, \
ipartialincr]
ar ATScross ktimepnt, kfmmod, iatsfile, ifn, kmylev, kbuflev, ipartials \
[, ipartialoffset, ipartialincr]
idata ATSinfo iatsfile, ilocation
```



```
kamp ATSinterpread kfreq

kfrq, kamp ATSpartialtap ipartialnum

kfreq, kamp ATSread ktimepnt, iatsfile, ipartial

kenergy ATSreadnz ktimepnt, iatsfile, iband

ar ATSSinnoi ktimepnt, ksinlev, knzlev, kfmmod, iatsfile, ipartials \
    [, ipartialoffset, ipartialincr]
```

### Traitement spectral : Loris.

```
lorismorph isrcidx, itgtidx, istoreidx, kfreqmorphenv, kampmorphenv, kbwmorphenv

ar lorisplay ireadidx, kfreqenv, kampenv, kbwenv

lorisread ktimepnt, ifilcod, istoreidx, kfreqenv, kampenv, kbwenv[, ifadetime]
```

### Traitement spectral : autres.

```
kcent centroid asig, ktrig, ifftsize

asig[, asig2] filescale ktimescal, kamp, kpitch, Sfile, klock [,ifftsize, iddecim,
    ithresh]

asig mincer atimpt, kamp, kpitch, ktab, klock[,ifftsize,iddecim]

asig, asig2, ktime mp3scal Sfile, ktimescal, kpitch, kamp [,iskip, ifftsize, iddecim,
    ilock]

asig paulstretch istretch, iwindowsize, ift
```

### Chaînes : définition.

```
Sdst strfromurl StringURL

Sdst strget indx

strset iarg, istring
```

### Chaînes : manipulation.

```
puts Sstr, ktrig[, inonl]

Sdst sprintf Sfmt, xarg1[, xarg2[, ... ]]

Sdst sprintfk Sfmt, xarg1[, xarg2[, ... ]]
```

```

Sdst strcat Ssrc1, Ssrc2

Sdst strcatk Ssrc1, Ssrc2

ires strcmp S1, S2

kres strcmpk S1, S2

Sdst strcpy Ssrc
Sdst = Ssrc

Sdst strcpyk Ssrc

ipos strindex S1, S2

kpos strindexk S1, S2

ilen strlen Sstr

klen strlenk Sstr

ipos strrindex S1, S2

kpos strrindexk S1, S2

Sout strstrip Sin [, Smodel]

Sdst strsub Ssrc[, istart[, iend]]

Sdst strsubk Ssrc, kstart, kend

```

### Chaînes : conversion.

```

ichr strchar Sstr[, ipos]

kchr strchark Sstr[, kpos]

Sdst strlower Ssrc

Sdst strlowerk Ssrc

ir strtod Sstr
ir strtod indx

kr strtodk Sstr
kr strtodk kndx

ir strtol Sstr
ir strtol indx

kr strtolk Sstr
kr strtolk kndx

Sdst strupper Ssrc

Sdst strupperk Ssrc

```

### Vectériel : tableaux.

```

vtaba andx, ifn, aout1 [, aout2, aout3, .... , aoutN ]

```

```

vtabi  indx, ifn, iout1 [, iout2, iout3, .... , ioutN ]
vtabk  kndx, ifn, kout1 [, kout2, kout3, .... , koutN ]
vtablek kfn, kout1 [, kout2, kout3, .... , koutN ]
vtablea andx, kfn, kinterp, ixmode, aout1 [, aout2, aout3, .... , aoutN ]
vtablei indx, ifn, interp, ixmode, iout1 [, iout2, iout3, .... , ioutN ]
vtablek kndx, kfn, kinterp, ixmode, kout1 [, kout2, kout3, .... , koutN ]
vtablewa andx, kfn, ixmode, ainarg1 [, ainarg2, ainarg3 , .... , ainargN ]
vtablewi indx, ifn, ixmode, inarg1 [, inarg2, inarg3 , .... , inargN ]
vtablewk kndx, kfn, ixmode, kinarg1 [, kinarg2, kinarg3 , .... , kinargN ]
vtabwa andx, ifn, ainarg1 [, ainarg2, ainarg3 , .... , ainargN ]
vtabwi indx, ifn, inarg1 [, inarg2, inarg3 , .... , inargN ]
vtabwk kndx, ifn, kinarg1 [, kinarg2, kinarg3 , .... , kinargN ]

```

### Vectoriel : opérations scalaires.

```

vadd  ifn, kval, kelements [, kdstoffset] [, kverbose]
vadd_i ifn, ival, ielements [, idstoffset]
vexp  ifn, kval, kelements [, kdstoffset] [, kverbose]
vexp_i ifn, ival, ielements[, idstoffset]
vmult ifn, kval, kelements [, kdstoffset] [, kverbose]
vmult_i ifn, ival, ielements [, idstoffset]
vpow  ifn, kval, kelements [, kdstoffset] [, kverbose]
vpow_i ifn, ival, ielements [, idstoffset]

```

### Vectoriel : opérations vectorielles.

```

vaddv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
vaddv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
vcopy ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [, kverbose]
vcopy_i ifn1, ifn2, ielements [,idstoffset, isrcoffset]
vdivv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
vdivv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]
vexpv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]
vexpv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]

```

```

vmap ifn1, ifn2, ielements [, idstoffset, isrcoffset]

vmultv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]

vmultv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]

vpowv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]

vpowv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]

vsubv ifn1, ifn2, kelements [, kdstoffset] [, ksrcoffset] [,kverbose]

vsubv_i ifn1, ifn2, ielements [, idstoffset] [, isrcoffset]

```

### Vectoriel : enveloppes.

```

vexpseg ifnout, ielements, ifn1, idur1, ifn2 [, idur2, ifn3 [...]]

vlinseg ifnout, ielements, ifn1, idur1, ifn2 [, idur2, ifn3 [...]]

```

### Vectoriel : limitation et enroulement.

```

vlimit ifn, kmin, kmax, ielements

vmirror ifn, kmin, kmax, ielements

vwrap ifn, kmin, kmax, ielements

```

### Vectoriel : chemins de retard.

```

kout vdelayk ksig, kdel, imaxdel [, iskip, imode]

vecdelay ifn, ifnIn, ifnDel, ielements, imaxdel [, iskip]

vport ifn, khtime, ielements [, ifnInit]

```

### Vectoriel : aléatoire.

```

vrandh ifn, krange, kcps, ielements [, idstoffset] [, iseed] \
    [, isize] [, ioffset]

vrandi ifn, krange, kcps, ielements [, idstoffset] [, iseed] \
    [, isize] [, ioffset]

```

### Vectoriel : automates cellulaires.

```

cell ktrig, kreinit, ioutFunc, initStateFunc, iRuleFunc, ielements

```

```
vcella ktrig, kreinit, ioutFunc, initStateFunc, \
        iRuleFunc, ielements, irulelen [, iradius]
```

### Système de patch zak.

```
zACL kfirst [, klast]

zakinit isizea, isizek

ares zamod asig, kzamod

ares zar kndx

ares zarg kndx, kgain

zaw asig, kndx

zawm asig, kndx [, imix]

ir zir indx

ziw isig, indx

ziwm isig, indx [, imix]

zkcl kfirst, klast

kres zkmod ksig, kzkmmod

kres zkr kndx

zkw kval, kndx

zkwm ksig, kndx [, imix]
```

### Accueil de greffon : DSSI et LADSPA.

```
dssiactivate ihandle, ktoggle

[aout1, aout2, ..., aout9] dssiaudio ihandle, [ain1, ain2, ..., ain9]

dssictls ihandle, iport, kvalue, ktrigger

ihandle dssiinit ilibraryname, igreffondex [, iverbose]

dssilist
```

### OSC.

```
OSCbundle kwhen, ihost, iport, \
        Sdest[], Stype[], kArgs[][][], isize]

kans OSCcount

ihandle OSCinit iport
```

```

ihandle OSCinitM Sgroup, iport

kans OSClisten ihandle, idest, itype [, xdata1, xdata2, ...]
kans, kdata[] OSClisten ihandle, idest, itype

Smess[],klen OSCraw iport

OSCsend kwhen, ihost, iport, idestination[, itype , xdata1, xdata2, ...]

```

### Opcodes Faust.

```

ihandle, a1 [,a2, ...] faustaudio ifac [,ain1, ...]

ihandle faustcompile Scode, Sargs[, iasync, istacksize]

faustctl idsp, Scontrol, kval

ihandle faustdsp ifac

ihandle, a1 [,a2, ...] faustgen SCode [,ain1, ...]

a1[, a2,...] faustplay ihandle[, ain1,...]

```

### Réseau.

```

remoteport iportnum

asig sockrecv iport, ilength
ksig sockrecv iport, ilength
asigl, asigr sockrecvs iport, ilength
String sockrecv iport, ilength
asig[,kstate] strecv Sipaddr, iport

socksend asig, Sipaddr, iport, ilength
socksend ksig, Sipaddr, iport, ilength
socksends asigl, asigr, Sipaddr, iport,
    ilength
stsend asig, Sipaddr, iport

```

### Opcodes pour le traitement à distance.

```

insglobal isource, instrnum [,instrnum...]

insremot idestination, isource, instrnum [,instrnum...]

midglobal isource, instrnum [,instrnum...]

midremot idestination, isource, instrnum [,instrnum...]

```

### Opcodes Mixer.

```

MixerClear

```

```
kgain MixerGetLevel isend, ibuss

asignal MixerReceive ibuss, ichannel

MixerSend asignal, isend, ibuss, ichannel

MixerSetLevel isend, ibuss, kgain

MixerSetLevel_i isend, ibuss, igain
```

## Opcodes Ableton Link.

```
link_beat_force i_peer, k_beat [, k_at_time_seconds [, k_quantum ]]

k_beat_number, k_phase, k_current_time_seconds link_beat_get i_peer [, k_quantum]

link_beat_request i_peer, k_beat [, k_at_time_seconds [, k_quantum ]]

i_peer link_create [i_bpm]

ableton_link_enable i_peer [, k_enable]

k_is_enabled link_is_enabled i_peer

k_trigger, k_beat, k_phase, k_current_time_seconds link_metro i_peer [, k_quantum]

k_count link_peers i_peer

k_bpm link_tempo_get i_peer

link_tempo_set i_peer, k_bpm [, k_at_time_seconds]
```

## Opcodes Python.

```
pyassign "variable", kvalue
pyassigni "variable", ivalue
pylassign "variable", kvalue
pylassigni "variable", ivalue
pyassignt ktrigger, "variable", kvalue
pylassignt ktrigger, "variable", kvalue

pycall "callable", karg1, ...
pycall1 "callable", karg1, ...
pycall12 "callable", karg1, ...
pycall13 "callable", karg1, ...
pycall14 "callable", karg1, ...
pycall15 "callable", karg1, ...
pycall16 "callable", karg1, ...
pycall17 "callable", karg1, ...
pycall18 "callable", karg1, ...
pycall1t ktrigger, "callable", karg1, ...
pycall11t ktrigger, "callable", karg1, ...
pycall12t ktrigger, "callable", karg1, ...
pycall13t ktrigger, "callable", karg1, ...
pycall14t ktrigger, "callable", karg1, ...
pycall15t ktrigger, "callable", karg1, ...
pycall16t ktrigger, "callable", karg1, ...
pycall17t ktrigger, "callable", karg1, ...
pycall18t ktrigger, "callable", karg1, ...
pycalli "callable", karg1, ...
```

```

iresult      pycall11i "callable", iarg1, ...
iresultt1, iresult2  pycall12i "callable", iarg1, ...
ir1, ir2, ir3      pycall13i "callable", iarg1, ...
ir1, ir2, ir3, ir4  pycall14i "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5  pycall15i "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6  pycall16i "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6, ir7  pycall17i "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6, ir7, ir8  pycall18i "callable", iarg1, ...
pycalln      "callable", nresults, kresult1, ..., kresultn, karg1, ...
pycallni     "callable", nresults, iresult1, ..., iresultn, iarg1, ...

kresult      pycall1      "callable", karg1, ...
kresultt1, kresult2  pycall11 "callable", karg1, ...
kr1, kr2, kr3      pycall12 "callable", karg1, ...
kr1, kr2, kr3, kr4  pycall13 "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5  pycall14 "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6  pycall15 "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7  pycall16 "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8  pycall17 "callable", karg1, ...
kresult      pycall18 "callable", karg1, ...
kresultt1, kresult2  pycall1t ktrigger, "callable", karg1, ...
kr1, kr2, kr3      pycall11t ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4  pycall12t ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5  pycall13t ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6  pycall14t ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7  pycall15t ktrigger, "callable", karg1, ...
kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8  pycall16t ktrigger, "callable", karg1, ...
iresult      pycall17t ktrigger, "callable", karg1, ...
iresultt1, iresult2  pycall18t ktrigger, "callable", karg1, ...
ir1, ir2, ir3      pycalli      "callable", karg1, ...
ir1, ir2, ir3, ir4  pycall11i "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5  pycall12i "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6  pycall13i "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6, ir7  pycall14i "callable", iarg1, ...
ir1, ir2, ir3, ir4, ir5, ir6, ir7, ir8  pycall15i "callable", iarg1, ...
pycalln      pycall16i "callable", iarg1, ...
pycallni     pycall17i "callable", iarg1, ...

kresult pyeval "expression"
iresult pyevali "expression"
kresult pyleval "expression"
iresult pylevali "expression"
kresult pyevalt ktrigger, "expression"
kresult pylevalt ktrigger, "expression"

pyexec "filename"
pyexeci "filename"
pylexec "filename"
pylexeci "filename"
pyexec t ktrigger, "filename"
pylexec t ktrigger, "filename"

pyinit

pyrun "statement"
pyruni "statement"
pylrn "statement"
pylrni "statement"
pyrnt ktrigger, "statement"
pylrnt ktrigger, "statement"

```

Opcodes pour le traitement d'image.



```

iimagenum imagecreate iwidth, iheight

imagefree iimagenum

ared, agreen, ablue imagegetpixel iimagenum, ax, ay
kred, kgreen, kblue imagegetpixel iimagenum, kx, ky

iimagenum imageload filename

imagesave iimagenum, filename

imagesetpixel iimagenum, ax, ay, ared, agreen, ablue
imagesetpixel iimagenum, kx, ky, kred, kgreen, kblue

iwidth, iheight imagesize iimagenum

```

### Opcodes de tableaux.

```

copya2ftab kArray[], ktab [, koffset]
copya2ftab iArray[], itab [, ioffset]

copyf2array tab, kftbl

i/kout[] getcolii/kin[], i/kcol

i/kout[] getrowii/kin[],i/krow

kmax [,kindx] maxarray karray

kmin [,kindx] minarray karray

printarray ix[] [, Smft, Slabel ]
printarray kx[] [, ktrig, Sfmt, Slabel ]

kres/iresproduct karr[]/iarr[] (k- or i-arrays )

reshapearray array[], isize0 [, isize1 ]

scalearray tarray, kmin, kmax[, kleft, kright]

i/kout[] setcolii/kin[],i/kcol

i/kout[] setrowii/kin[],i/krow

kout[] shiftin asig

asig shiftoutkIn[][, ioff]

ksum sumarray karray

kout[] tab2array ifn [, kstart, kend, kstep ]
iout[] tab2array ifn [, istart, iend, istep ]

trim_i iarray, ilen
trim xarray, klen

```

### Opérations de tableau : transformée de Fourier rapide.

```

kout[] fft kin[]

```

```
kout[] fftnv kin[]
kout[] rfft kin[]
kout[] rifft kin[]
kout[] unwrap kin[]
kout[] window kin[],koff, itype]
```

### Opérations de tableau : nombres complexes.

```
kout[] c2r kin[]
kout[] cmplxprod kin1[], kin2[]
kout[] mags kin[]
kout[] phs kin[]
kout[] pol2rect kin[]
kout[] pol2rect kmags[], kphs[]
kout[] r2c kin[]
kout[] rect2pol kin[]
```

### Opérations de tableau : transformée cosinus discrète.

```
kout[] dct kin[]
iout[] dct iin[]
kout[] dctinv kin[]
iout[] dctinv iin[]
```

### Opérations de tableau : produit scalaire.

```
kres/iresdot karr1[]/iarr1[], karr2[]/iarr2[] (k- or i-arrays )
```

### Opérations de tableau : banc de filtres à échelle de Mel.

```
kout[] mfb kin[], klow, khigh, ibands
```

### Opérations de tableau : tri.

```
k/i[]sorta k/i[] (k- or i-arrays )
k/i[]sortd k/i[] (k- or i-arrays )
```

### Opérations de tableau : cepstre.

```

kaps[] ceps kmags[][ , icoefs]

kenv cepsinv kaps[]

```

### Opérations de tableau : obsolète.

```

ktableseg ifn1, idur1, ifn2 [ , idur2] [ , ifn3] [...]

tableiw isig, indx, ifn [ , ixmode] [ , ixoff] [ , iwgmodes]

```

### Divers.

```

SFiles[] directory SDirectory[ , SExtention]

kfl fareylen kfn

ifl fareyleni ifn

kout[] framebuffer ain, isize
aout framebuffer kin, isize

modmatrix iresfn, isrcmodfn, isrcparmf, imodscale, inum_mod, \
inum_parm, kupdate

idacc, iadcc nchnls_hw

aout olabuffer kin, ioverlap

Sres pwd

aout select a1, a2, aless, aequal, amore

ires system_i itrig, Scmd, [inowait]
kres system ktrig, Scmd, [knowait]

tableshuffle ktalenum
tableshufflei italenum

```

### Utilitaires.

```

csound -U atsa [options] nomfichier_entree nomfichier_sortie

cs [-OPTIONS] <nom> [OPTIONS DE CSOUND ... ]

csb64enc [OPTIONS ... ] fichier1 [ fichier2 [ ... ] ]

csound -U cvanal [options] nomfichier_entree nomfichier_sortie
cvanal [options] nomfichier_entree nomfichier_sortie

dnoise [options] -i ficref_bruit -o ficson_sortie ficson_entree

envext [-options] fichierson
csound -U envext [-options] fichierson

```

```

extractor [OPTIONS ... ] fichierentree

het_export fichier_het fichier_textecsv
csound -U het_export fichier_het fichier_textecsv

het_import fichier_textecsv fichier_het
csound -U het_import fichier_textecsv fichier_het

csound -U hetro [options] nomfichier_entree nomfichier_sortie
hetro [options] nomfichier_entree nomfichier_sortie

csound -U lpanal [options] nomfichier_entree nomfichier_sortie
lpanal [options] nomfichier_entree nomfichier_sortie

makecsd [OPTIONS ... ] fichier1 [ fichier2 [ ... ]]

mixer [OPTIONS ... ] fichier [[OPTIONS... ] fichier] ...

mkdb [-m] [base_directory]

pv_export fichier_pv fichier_texte_csv
csound -U pv_export fichier_pv fichier_texte_csv

pv_import fichier_texte_csv fichier_pv
csound -U pv_import fichier_texte_csv fichier_pv

csound -U pvanal [options] nomfic_entree nomfic_sortie
pvanal [options] nomfic_entree nomfic_sortie

csound -U pvlook [options] fichier_entree
pvlook [options] fichier_entree

scale [OPTIONS ... ] fichier

sdif2ad [options] fichier_entree fichier_sortie

csound -U sndinfo [options] fichierson ...
sndinfo [options] fichierson ...

src_conv [options] fichier_entree

srconv [options] fichier_entree

```

---

# Annexe A. Liste des exemples

## **Syntaxe de l'orchestre : en-tête.**

*0dbfs.csd* [examples/0dbfs.csd]

*nchnls.csd* [examples/nchnls.csd]

*nchnls\_i.csd* [examples/nchnls\_i.csd]

*sr.csd* [examples/sr.csd]

## **Syntaxe de l'orchestre : bloc d'instructions.**

*endin.csd* [examples/endin.csd]

*endop.csd* [examples/endop.csd]

*instr.csd* [examples/instr.csd]

*opcode\_example.csd* [examples/opcode\_example.csd]

## **Syntaxe de l'orchestre : macros.**

*define.csd* [examples/define.csd]

*define\_args.csd* [examples/define\_args.csd]

*define.csd* [examples/define.csd]

*define\_args.csd* [examples/define\_args.csd]

*include.csd* [examples/include.csd]

## **Générateurs de signal : synthèse/resynthèse additive.**

*adsyn.csd* [examples/adsyn.csd]

*adsynt.csd* [examples/adsynt.csd]

*adsynt2.csd* [examples/adsynt2.csd]

*beadsynt.csd* [examples/beadsynt.csd]

*beosc.csd* [examples/beosc.csd]

*hsboscil.csd* [examples/hsboscil.csd]

*hsboscil\_midi.csd* [examples/hsboscil\_midi.csd]

## **Générateurs de signal : oscillateurs élémentaires.**

*lfo.csd* [examples/lfo.csd]

*oscbnk.csd* [examples/oscbnk.csd]

*oscil.csd* [examples/oscil.csd]

*oscil3.csd* [examples/oscil3.csd]

*oscili.csd* [examples/oscili.csd]

*oscilikt.csd* [examples/oscilikt.csd]

*osciliktp.csd* [examples/osciliktp.csd]

*oscilikts.csd* [examples/oscilikts.csd]

*osciln.csd* [examples/osciln.csd]

*oscils.csd* [examples/oscils.csd]

*poscil.csd* [examples/poscil.csd]

*poscil3.csd* [examples/poscil3.csd]

*poscil3-file.csd* [examples/poscil3-file.csd]

*vibr.csd* [examples/vibr.csd]

*vibrato.csd* [examples/vibrato.csd]

### **Générateurs de signal : oscillateurs à spectre dynamique.**

*buzz.csd* [examples/buzz.csd]

*gbuzz.csd* [examples/gbuzz.csd]

*mpulse.csd* [examples/mpulse.csd]

*squnewave.csd* [examples/squnewave.csd]

*vco.csd* [examples/vco.csd]

*vco2.csd* [examples/vco2.csd]

*vco2ift.csd* [examples/vco2ift.csd]

*vco2init.csd* [examples/vco2init.csd]

### **Générateurs de signal : synthèse MF.**

*crossfm.csd* [examples/crossfm.csd]

*fmb3.csd* [examples/fmb3.csd]

*fmbell.csd* [examples/fmbell.csd]

*fmmetal.csd* [examples/fmmetal.csd]

*fmpercfl.csd* [examples/fmpercfl.csd]

*fmrhode.csd* [examples/fmrhode.csd]

*fmvoice.csd* [examples/fmvoice.csd]

*fmwurlie.csd* [examples/fmwurlie.csd]

*foscil.csd* [examples/foscil.csd]

*foscili.csd* [examples/foscili.csd]

**Générateurs de signal : synthèse granulaire.**

*diskgrain.csd* [examples/diskgrain.csd]

*fof.csd* [examples/fof.csd]

*fof2.csd* [examples/fof2.csd]

*fof2-2.csd* [examples/fof2-2.csd]

*fog.csd* [examples/fog.csd]

*grain.csd* [examples/grain.csd]

*grain2.csd* [examples/grain2.csd]

*grain3.csd* [examples/grain3.csd]

*granule.csd* [examples/granule.csd]

*partikkel.csd* [examples/partikkel.csd]

*partikkel-2.csd* [examples/partikkel-2.csd]

*partikkel-2.csd* [examples/partikkel-2.csd]

*partikkelgetset.csd* [examples/partikkelgetset.csd]

*partikkelgetset.csd* [examples/partikkelgetset.csd]

*partikkelsync.csd* [examples/partikkelsync.csd]

*sndwarp.csd* [examples/sndwarp.csd]

*sndwarpst.csd* [examples/sndwarpst.csd]

*syncgrain.csd* [examples/syncgrain.csd]

*syncloop.csd* [examples/syncloop.csd]

*vosim.csd* [examples/vosim.csd]

**Générateurs de signal : synthèse hyper vectorielle.**

*hvs1.csd* [examples/hvs1.csd]

*hvs2.csd* [examples/hvs2.csd]

*hvs2-2.csd* [examples/hvs2-2.csd]

*hvs3.csd* [examples/hvs3.csd]

**Générateurs de signal : générateurs linéaires et exponentiels.**

*bpf.csd* [examples/bpf.csd]

*bpfcos.csd* [examples/bpfcos.csd]

*cosseg.csd* [examples/cosseg.csd]

*cossegb.csd* [examples/cossegb.csd]

*cossegr.csd* [examples/cossegr.csd]

*expcurve.csd* [examples/expcurve.csd]

*expon.csd* [examples/expon.csd]

*expseg.csd* [examples/expseg.csd]

*expsega.csd* [examples/expsega.csd]

*expsegb.csd* [examples/expsegb.csd]

*expsegba.csd* [examples/expsegba.csd]

*expsegr.csd* [examples/expsegr.csd]

*gainslider.csd* [examples/gainslider.csd]

*lincos.csd* [examples/lincos.csd]

*line.csd* [examples/line.csd]

*linlin.csd* [examples/linlin.csd]

*linseg.csd* [examples/linseg.csd]

*linsegb.csd* [examples/linsegb.csd]

*linsegr.csd* [examples/linsegr.csd]

*logcurve.csd* [examples/logcurve.csd]

*loopseg.csd* [examples/loopseg.csd]

*loopsegp.csd* [examples/loopsegp.csd]

*looptseg.csd* [examples/looptseg.csd]

*loopxseg.csd* [examples/loopxseg.csd]

*lpshold.csd* [examples/lpshold.csd]

*scale.csd* [examples/scale.csd]



*transeg.csd* [examples/transeg.csd]

*transegb.csd* [examples/transegb.csd]

*transegr.csd* [examples/transegr.csd]

*xyscale.csd* [examples/xyscale.csd]

**Générateurs de signal : générateurs d'enveloppe.**

*adsr.csd* [examples/adsr.csd]

*envlpx.csd* [examples/envlpx.csd]

*envlpxr.csd* [examples/envlpxr.csd]

*linen.csd* [examples/linen.csd]

*linenr.csd* [examples/linenr.csd]

*madsr.csd* [examples/madsr.csd]

*madsr-2.csd* [examples/madsr-2.csd]

*mxadsr.csd* [examples/mxadsr.csd]

*xadsr.csd* [examples/xadsr.csd]

**Générateurs de signal : modèles et émulations.**

*bamboo.csd* [examples/bamboo.csd]

*barmodel.csd* [examples/barmodel.csd]

*cabasa.csd* [examples/cabasa.csd]

*chuap.csd* [examples/chuap.csd]

*crunch.csd* [examples/crunch.csd]

*dripwater.csd* [examples/dripwater.csd]

*gendy.csd* [examples/gendy.csd]

*gendy-2.csd* [examples/gendy-2.csd]

*gendyc.csd* [examples/gendyc.csd]

*gendyx.csd* [examples/gendyx.csd]

*gendyx-2.csd* [examples/gendyx-2.csd]

*gogobel.csd* [examples/gogobel.csd]

*guiro.csd* [examples/guiro.csd]

*lorenz.csd* [examples/lorenz.csd]

*mandel.csd* [examples/mandel.csd]

*mandol.csd* [examples/mandol.csd]

*marimba.csd* [examples/marimba.csd]

*moog.csd* [examples/moog.csd]

*planet.csd* [examples/planet.csd]

*prepiano.csd* [examples/prepiano.csd]

*sandpaper.csd* [examples/sandpaper.csd]

*sekere.csd* [examples/sekere.csd]

*shaker.csd* [examples/shaker.csd]

*sleighbells.csd* [examples/sleighbells.csd]

*stix.csd* [examples/stix.csd]

*tambourine.csd* [examples/tambourine.csd]

*vibes.csd* [examples/vibes.csd]

*voice.csd* [examples/voice.csd]

#### **Générateurs de signal : phaseurs.**

*ephasor.csd* [examples/ephasor.csd]

*phasor.csd* [examples/phasor.csd]

*phasorbnk.csd* [examples/phasorbnk.csd]

*sc\_phasor.csd* [examples/sc\_phasor.csd]

*syncphasor.csd* [examples/syncphasor.csd]

*syncphasor-CZresonance.csd* [examples/syncphasor-CZresonance.csd]

#### **Générateurs de signal : générateurs de nombres aléatoires (de bruit).**

*betarand.csd* [examples/betarand.csd]

*bexprnd.csd* [examples/bexprnd.csd]

*cauchy.csd* [examples/cauchy.csd]

*cauchy1.csd* [examples/cauchy1.csd]

*cusernd.csd* [examples/cusernd.csd]

*dusernd.csd* [examples/dusernd.csd]

*dust.csd* [examples/dust.csd]

*dust2.csd* [examples/dust2.csd]

*exprand.csd* [examples/exprand.csd]

*exprandi.csd* [examples/exprandi.csd]

*fractalnoise.csd* [examples/fractalnoise.csd]

*gauss.csd* [examples/gauss.csd]

*gaussi.csd* [examples/gaussi.csd]

*gausstrig.csd* [examples/gausstrig.csd]

*gausstrig-2.csd* [examples/gausstrig-2.csd]

*jitter.csd* [examples/jitter.csd]

*jitter2.csd* [examples/jitter2.csd]

*jspline.csd* [examples/jspline.csd]

*linrand.csd* [examples/linrand.csd]

*noise.csd* [examples/noise.csd]

*noise-2.csd* [examples/noise-2.csd]

*pcauchy.csd* [examples/pcauchy.csd]

*pinker.csd* [examples/pinker.csd]

*pinkish.csd* [examples/pinkish.csd]

*poisson.csd* [examples/poisson.csd]

*rand.csd* [examples/rand.csd]

*randh.csd* [examples/randh.csd]

*randi.csd* [examples/randi.csd]

*random.csd* [examples/random.csd]

*randomh.csd* [examples/randomh.csd]

*randomi.csd* [examples/randomi.csd]

*rnd31.csd* [examples/rnd31.csd]

*rnd31\_krate.csd* [examples/rnd31\_krate.csd]

*rnd31\_seed7.csd* [examples/rnd31\_seed7.csd]

*rnd31\_time.csd* [examples/rnd31\_time.csd]

*rspline.csd* [examples/rspline.csd]

*seed.csd* [examples/seed.csd]

*trandom.csd* [examples/trandom.csd]

*trirand.csd* [examples/trirand.csd]

*unirand.csd* [examples/unirand.csd]

*urandom.csd* [examples/urandom.csd]

*urandom\_krate.csd* [examples/urandom\_krate.csd]

*urd.csd* [examples/urd.csd]

*weibull.csd* [examples/weibull.csd]

**Générateurs de signal : reproduction de sons échantillonnés.**

*bbcutm.csd* [examples/bbcutm.csd]

*bbcuts.csd* [examples/bbcuts.csd]

*flooper.csd* [examples/flooper.csd]

*flooper2.csd* [examples/flooper2.csd]

*fluidAllOut.csd* [examples/fluidAllOut.csd]

*fluidCCi.csd* [examples/fluidCCi.csd]

*fluidCCk.csd* [examples/fluidCCk.csd]

*fluidcomplex.csd* [examples/fluidcomplex.csd]

*fluidEngine.csd* [examples/fluidEngine.csd]

*fluidInfo.csd* [examples/fluidInfo.csd]

*fluidLoad.csd* [examples/fluidLoad.csd]

*fluidNote.csd* [examples/fluidNote.csd]

*fluidOut.csd* [examples/fluidOut.csd]

*fluidProgramSelect.csd* [examples/fluidProgramSelect.csd]

*fluidcomplex.csd* [examples/fluidcomplex.csd]

*fluidSetInterpMethod.csd* [examples/fluidSetInterpMethod.csd]

*loscil.csd* [examples/loscil.csd]

*loscil3.csd* [examples/loscil3.csd]

*loscilx.csd* [examples/loscilx.csd]

*lphasor.csd* [examples/lphasor.csd]

*lposcil.csd* [examples/lposcil.csd]

*lposcil3.csd* [examples/lposcil3.csd]  
*lposcila.csd* [examples/lposcila.csd]  
*lposcilsa.csd* [examples/lposcilsa.csd]  
*lposcilsa2.csd* [examples/lposcilsa2.csd]  
*sfilist.csd* [examples/sfilist.csd]  
*sfinstr.csd* [examples/sfinstr.csd]  
*sfinstr3.csd* [examples/sfinstr3.csd]  
*sfinstr3m.csd* [examples/sfinstr3m.csd]  
*sfload.csd* [examples/sfload.csd]  
*sflooper.csd* [examples/sflooper.csd]  
*sfpassign.csd* [examples/sfpassign.csd]  
*sfplay3.csd* [examples/sfplay3.csd]  
*sfplay3m.csd* [examples/sfplay3m.csd]  
*sfplaym.csd* [examples/sfplaym.csd]  
*sfplist.csd* [examples/sfplist.csd]  
*sfpreset.csd* [examples/sfpreset.csd]  
*sndloop.csd* [examples/sndloop.csd]  
*waveset.csd* [examples/waveset.csd]

**Générateurs de signal : synthèse par balayage.**

*scans.csd* [examples/scans.csd]  
*scans-2.csd* [examples/scans-2.csd]  
*scantable.csd* [examples/scantable.csd]  
*xscanmap.csd* [examples/xscanmap.csd]  
*xs cans.csd* [examples/xs cans.csd]  
*xscanu.csd* [examples/xscanu.csd]

**Générateurs de signal : opcodes STK.**

*STKBandedWG.csd* [examples/STKBandedWG.csd]  
*STKBeeThree.csd* [examples/STKBeeThree.csd]  
*STKBlowBotl.csd* [examples/STKBlowBotl.csd]

*STKBlowHole.csd* [examples/STKBlowHole.csd]

*STKBowed.csd* [examples/STKBowed.csd]

*STKBrass.csd* [examples/STKBrass.csd]

*STKClarinet.csd* [examples/STKClarinet.csd]

*STKDrummer.csd* [examples/STKDrummer.csd]

*STKFMVoices.csd* [examples/STKFMVoices.csd]

*STKFlute.csd* [examples/STKFlute.csd]

*STKHevyMetl.csd* [examples/STKHevyMetl.csd]

*STKMandolin.csd* [examples/STKMandolin.csd]

*STKModalBar.csd* [examples/STKModalBar.csd]

*STKMoog.csd* [examples/STKMoog.csd]

*STKPercFlut.csd* [examples/STKPercFlut.csd]

*STKPlucked.csd* [examples/STKPlucked.csd]

*STKResonate.csd* [examples/STKResonate.csd]

*STKRhodey.csd* [examples/STKRhodey.csd]

*STKSaxofony.csd* [examples/STKSaxofony.csd]

*STKShakers.csd* [examples/STKShakers.csd]

*STKSimple.csd* [examples/STKSimple.csd]

*STKSitar.csd* [examples/STKSitar.csd]

*STKStifKarp.csd* [examples/STKStifKarp.csd]

*STKTubeBell.csd* [examples/STKTubeBell.csd]

*STKVoicForm.csd* [examples/STKVoicForm.csd]

*STKWhistle.csd* [examples/STKWhistle.csd]

*STKWurley.csd* [examples/STKWurley.csd]

**Générateurs de signal : accès aux tables.**

*oscil1i.csd* [examples/oscil1i.csd]

*oscil1i.csd* [examples/oscil1i.csd]

*ptable.csd* [examples/ptable.csd]

*ptablei.csd* [examples/ptablei.csd]

*tab.csd* [examples/tab.csd]

*table.csd* [examples/table.csd]

*tablei.csd* [examples/tablei.csd]

**Générateurs de signal : synthèse par terrain d'ondes.**

*wterrain.csd* [examples/wterrain.csd]

**Générateurs de signal : modèles physiques par guide d'onde.**

*pluck.csd* [examples/pluck.csd]

*repluck.csd* [examples/repluck.csd]

*streson.csd* [examples/streson.csd]

*wgbow.csd* [examples/wgbow.csd]

*wgbowedbar.csd* [examples/wgbowedbar.csd]

*wgbrass.csd* [examples/wgbrass.csd]

*wgclar.csd* [examples/wgclar.csd]

*wgflute.csd* [examples/wgflute.csd]

*wgpluck.csd* [examples/wgpluck.csd]

*wgpluck\_brighter.csd* [examples/wgpluck\_brighter.csd]

*wgpluck2.csd* [examples/wgpluck2.csd]

**E/S de signal : E/S fichier.**

*dumpk.csd* [examples/dumpk.csd]

*dumpk-2.csd* [examples/dumpk-2.csd]

*dumpk2.csd* [examples/dumpk2.csd]

*dumpk3.csd* [examples/dumpk3.csd]

*dumpk4.csd* [examples/dumpk4.csd]

*ficlose.csd* [examples/ficlose.csd]

*fin.csd* [examples/fin.csd]

*fini.csd* [examples/fini.csd]

*fink.csd* [examples/fink.csd]

*fiopen.csd* [examples/fiopen.csd]

*fout.csd* [examples/fout.csd]  
*fout\_ftable.csd* [examples/fout\_ftable.csd]  
*fouti.csd* [examples/fouti.csd]  
*foutir.csd* [examples/foutir.csd]  
*fprintks.csd* [examples/fprintks.csd]  
*fprintks-2.csd* [examples/fprintks-2.csd]  
*scogen.csd* [examples/scogen.csd]  
*fprints.csd* [examples/fprints.csd]  
*hdf5read.csd* [examples/hdf5read.csd]  
*hdf5write.csd* [examples/hdf5write.csd]  
*readf.csd* [examples/readf.csd]  
*readfi.csd* [examples/readfi.csd]  
*readk.csd* [examples/readk.csd]  
*readk-2.csd* [examples/readk-2.csd]  
*readk2.csd* [examples/readk2.csd]  
*readk3.csd* [examples/readk3.csd]  
*readk4.csd* [examples/readk4.csd]  
*websocket.csd* [examples/websocket.csd]

**E/S de signal : entrée de signal.**

*diskin.csd* [examples/diskin.csd]  
*diskin2.csd* [examples/diskin2.csd]  
*in.csd* [examples/in.csd]  
*inch.csd* [examples/inch.csd]  
*inq.csd* [examples/inq.csd]  
*inrg.csd* [examples/inrg.csd]  
*ins.csd* [examples/ins.csd]  
*invalue.csd* [examples/invalue.csd]  
*mp3in.csd* [examples/mp3in.csd]  
*soundin.csd* [examples/soundin.csd]



**E/S de signal : sortie de signal.**

*mdelay.csd* [examples/mdelay.csd]  
*monitor.csd* [examples/monitor.csd]  
*out.csd* [examples/out.csd]  
*outc.csd* [examples/outc.csd]  
*outch.csd* [examples/outch.csd]  
*outch-2.csd* [examples/outch-2.csd]  
*outq.csd* [examples/outq.csd]  
*outq1.csd* [examples/outq1.csd]  
*outq2.csd* [examples/outq2.csd]  
*outq3.csd* [examples/outq3.csd]  
*outq4.csd* [examples/outq4.csd]  
*outrg.csd* [examples/outrg.csd]  
*outs.csd* [examples/outs.csd]  
*outs1.csd* [examples/outs1.csd]  
*outs2.csd* [examples/outs2.csd]  
*outvalue.csd* [examples/outvalue.csd]

**E/S de signal : bus logiciel.**

*chnclear.csd* [examples/chnclear.csd]  
*chnget.csd* [examples/chnget.csd]  
*chnmix.csd* [examples/chnmix.csd]  
*chnset.csd* [examples/chnset.csd]

**E/S de signal : impression et affichage.**

*dispfft.csd* [examples/disppfft.csd]  
*display.csd* [examples/display.csd]  
*flashtxt.csd* [examples/flashtxt.csd]  
*print.csd* [examples/print.csd]  
*printf.csd* [examples/printf.csd]  
*printk.csd* [examples/printk.csd]

*printk2.csd* [examples/printk2.csd]

*printks.csd* [examples/printks.csd]

*printks2.csd* [examples/printks2.csd]

*prints.csd* [examples/prints.csd]

**E/S de signal : requêtes sur les fichiers sons.**

*filebit.csd* [examples/filebit.csd]

*filelen.csd* [examples/filelen.csd]

*filenchnls.csd* [examples/filenchnls.csd]

*filepeak.csd* [examples/filepeak.csd]

*filesr.csd* [examples/filesr.csd]

*filevalid.csd* [examples/filevalid.csd]

*mp3len.csd* [examples/mp3len.csd]

**Modificateurs de signal : modificateurs d'amplitude.**

*balance.csd* [examples/balance.csd]

*balance2.csd* [examples/balance2.csd]

*clip.csd* [examples/clip.csd]

*compress.csd* [examples/compress.csd]

*compress2.csd* [examples/compress2.csd]

*dam.csd* [examples/dam.csd]

*dam\_expanded.csd* [examples/dam\_expanded.csd]

*gain.csd* [examples/gain.csd]

**Modificateurs de signal : convolution et morphing.**

*convolve.csd* [examples/convolve.csd]

*cross2.csd* [examples/cross2.csd]

*dconv.csd* [examples/dconv.csd]

*ftconv.csd* [examples/ftconv.csd]

*ftmorf.csd* [examples/ftmorf.csd]

*liveconv.csd* [examples/liveconv.csd]

*pconvolve.csd* [examples/pconvolve.csd]

*tvconv.csd* [examples/tvconv.csd]

**Modificateurs de signal : retard.**

*delay.csd* [examples/delay.csd]

*delay1.csd* [examples/delay1.csd]

*delayk.csd* [examples/delayk.csd]

*delayr.csd* [examples/delayr.csd]

*delayw.csd* [examples/delayw.csd]

*deltap.csd* [examples/deltap.csd]

*deltap3.csd* [examples/deltap3.csd]

*deltapi.csd* [examples/deltapi.csd]

*deltapn.csd* [examples/deltapn.csd]

*deltapx.csd* [examples/deltapx.csd]

*deltapxw.csd* [examples/deltapxw.csd]

*multitap.csd* [examples/multitap.csd]

*vdelay.csd* [examples/vdelay.csd]

*vdelay3.csd* [examples/vdelay3.csd]

*vdelayx.csd* [examples/vdelayx.csd]

*vdelayxq.csd* [examples/vdelayxq.csd]

*vdelayxs.csd* [examples/vdelayxs.csd]

*vdelayxw.csd* [examples/vdelayxw.csd]

*vdelayxwq.csd* [examples/vdelayxwq.csd]

*vdelayxws.csd* [examples/vdelayxws.csd]

**Modificateurs de signal : panoramique et spatialisation.**

*bformenc.csd* [examples/bformenc.csd]

*bformenc1.csd* [examples/bformenc1.csd]

*bformenc.csd* [examples/bformenc.csd]

*bformenc1.csd* [examples/bformenc1.csd]

*hrtfearly.csd* [examples/hrtfearly.csd]

*hrtfer.csd* [examples/hrtfer.csd]

*hrtfmove.csd* [examples/hrtfmove.csd]  
*hrtfmove2.csd* [examples/hrtfmove2.csd]  
*hrtfearly.csd* [examples/hrtfearly.csd]  
*hrtfstat.csd* [examples/hrtfstat.csd]  
*hrtfstat-2.csd* [examples/hrtfstat-2.csd]  
*locsend\_stereo.csd* [examples/locsend\_stereo.csd]  
*locsig\_quad.csd* [examples/locsig\_quad.csd]  
*pan.csd* [examples/pan.csd]  
*pan2.csd* [examples/pan2.csd]  
*space\_quad.csd* [examples/space\_quad.csd]  
*space\_stereo.csd* [examples/space\_stereo.csd]  
*spat3d\_stereo.csd* [examples/spat3d\_stereo.csd]  
*spat3d\_UHJ.csd* [examples/spat3d\_UHJ.csd]  
*spat3d\_quad.csd* [examples/spat3d\_quad.csd]  
*spat3dt.csd* [examples/spat3dt.csd]  
*spdist.csd* [examples/spdist.csd]  
*spsend.csd* [examples/spsend.csd]  
*vbap.csd* [examples/vbap.csd]  
*vbap4.csd* [examples/vbap4.csd]  
*vbap4move.csd* [examples/vbap4move.csd]  
*vbap8.csd* [examples/vbap8.csd]  
*vbap8move.csd* [examples/vbap8move.csd]  
*vbapg.csd* [examples/vbapg.csd]  
*vbapgmove.csd* [examples/vbapgmove.csd]  
*vbaplsinit.csd* [examples/vbaplsinit.csd]  
*vbapmove.csd* [examples/vbapmove.csd]

**Modificateurs de signal : réverbération.**

*alpass.csd* [examples/alpass.csd]  
*babo.csd* [examples/babo.csd]

*babo\_expert.csd* [examples/babo\_expert.csd]  
*comb.csd* [examples/comb.csd]  
*combinv.csd* [examples/combinv.csd]  
*freeverb.csd* [examples/freeverb.csd]  
*nestedap.csd* [examples/nestedap.csd]  
*nreverb.csd* [examples/nreverb.csd]  
*nreverb\_ftable.csd* [examples/nreverb\_ftable.csd]  
*plate.csd* [examples/plate.csd]  
*reverb.csd* [examples/reverb.csd]  
*reverbsc.csd* [examples/reverbsc.csd]  
*valpass.csd* [examples/valpass.csd]  
*valpass-2.csd* [examples/valpass-2.csd]  
*vcomb.csd* [examples/vcomb.csd]

**Modificateurs de signal : opérateurs du niveau échantillon.**

*denorm.csd* [examples/denorm.csd]  
*diff.csd* [examples/diff.csd]  
*downsamp.csd* [examples/downsamp.csd]  
*fold.csd* [examples/fold.csd]  
*integ.csd* [examples/integ.csd]  
*interp.csd* [examples/interp.csd]  
*ntrpol.csd* [examples/ntrpol.csd]  
*opa.csd* [examples/opa.csd]  
*samphold.csd* [examples/samphold.csd]  
*upsamp.csd* [examples/upsamp.csd]  
*vaget.csd* [examples/vaget.csd]  
*vaset.csd* [examples/vaset.csd]

**Modificateurs de signal : limiteurs de signal.**

*limit.csd* [examples/limit.csd]  
*mirror.csd* [examples/mirror.csd]

*wrap.csd* [examples/wrap.csd]

**Modificateurs de signal : effets spéciaux.**

*distort.csd* [examples/distort.csd]

*distort1.csd* [examples/distort1.csd]

*flanger.csd* [examples/flanger.csd]

*harmon.csd* [examples/harmon.csd]

*harmon3.csd* [examples/harmon3.csd]

*phaser1.csd* [examples/phaser1.csd]

*phaser2.csd* [examples/phaser2.csd]

**Modificateurs de signal : filtres standard.**

*atone.csd* [examples/atone.csd]

*atonex.csd* [examples/atonex.csd]

*biquad.csd* [examples/biquad.csd]

*biquad-2.csd* [examples/biquad-2.csd]

*biquada.csd* [examples/biquada.csd]

*butterbp.csd* [examples/butterbp.csd]

*butterbr.csd* [examples/butterbr.csd]

*butterhp.csd* [examples/butterhp.csd]

*butterlp.csd* [examples/butterlp.csd]

*clfilt\_lowpass.csd* [examples/clfilt\_lowpass.csd]

*clfilt\_highpass.csd* [examples/clfilt\_highpass.csd]

*diode\_ladder.csd* [examples/diode\_ladder.csd]

*doppler.csd* [examples/doppler.csd]

*k35.csd* [examples/k35.csd]

*k35.csd* [examples/k35.csd]

*median.csd* [examples/median.csd]

*mediank.csd* [examples/mediank.csd]

*mode.csd* [examples/mode.csd]

*tone.csd* [examples/tone.csd]

*tonex.csd* [examples/tonex.csd]

*zdf\_1pole.csd* [examples/zdf\_1pole.csd]

*zdf\_1pole\_mode.csd* [examples/zdf\_1pole\_mode.csd]

*zdf\_2pole.csd* [examples/zdf\_2pole.csd]

*zdf\_2pole\_mode.csd* [examples/zdf\_2pole\_mode.csd]

*zdf\_ladder.csd* [examples/zdf\_ladder.csd]

**Modificateurs de signal : filtres standard : résonants.**

*areson.csd* [examples/areson.csd]

*bqrez.csd* [examples/bqrez.csd]

*lowpass2.csd* [examples/lowpass2.csd]

*lowres.csd* [examples/lowres.csd]

*lowresx.csd* [examples/lowresx.csd]

*lpf18.csd* [examples/lpf18.csd]

*moogladder.csd* [examples/moogladder.csd]

*moogladder2.csd* [examples/moogladder2.csd]

*moogvcf.csd* [examples/moogvcf.csd]

*moogvcf2.csd* [examples/moogvcf2.csd]

*mvchpf.csd* [examples/mvchpf.csd]

*mvclpf1.csd* [examples/mvclpf1.csd]

*mvclpf2.csd* [examples/mvclpf2.csd]

*mvclpf3.csd* [examples/mvclpf3.csd]

*mvclpf4.csd* [examples/mvclpf4.csd]

*reson.csd* [examples/reson.csd]

*resonr.csd* [examples/resonr.csd]

*resonx.csd* [examples/resonx.csd]

*resony.csd* [examples/resony.csd]

*resonr.csd* [examples/resonr.csd]

*rezzy.csd* [examples/rezzy.csd]

*statevar.csd* [examples/statevar.csd]

*svfilter.csd* [examples/svfilter.csd]

*tbvcf.csd* [examples/tbvcf.csd]

*vlowres.csd* [examples/vlowres.csd]

**Modificateurs de signal : filtres standard : contrôle.**

*aresonk.csd* [examples/aresonk.csd]

*atonek.csd* [examples/atonek.csd]

*lineto.csd* [examples/lineto.csd]

*port.csd* [examples/port.csd]

*portk.csd* [examples/portk.csd]

*resonk.csd* [examples/resonk.csd]

*resonxk.csd* [examples/resonxk.csd]

*sc\_lag.csd* [examples/sc\_lag.csd]

*sc\_lagud.csd* [examples/sc\_lagud.csd]

*sc\_trig.csd* [examples/sc\_trig.csd]

*tlineto.csd* [examples/tlineto.csd]

*tonek.csd* [examples/tonek.csd]

**Modificateurs de signal : filtres spécialisés.**

*dcblock.csd* [examples/dcblock.csd]

*dcblock2.csd* [examples/dcblock2.csd]

*eqfil.csd* [examples/eqfil.csd]

*exciter.csd* [examples/exciter.csd]

*filter2.csd* [examples/filter2.csd]

*fmanal.csd* [examples/fmanal.csd]

*fofilter.csd* [examples/fofilter.csd]

*gtf.csd* [examples/gtf.csd]

*hilbert.csd* [examples/hilbert.csd]

*hilbert\_barberpole.csd* [examples/hilbert\_barberpole.csd]

*hilbert2.csd* [examples/hilbert2.csd]

*nlfilt.csd* [examples/nlfilt.csd]



*nlfilt2.csd* [examples/nlfilt2.csd]

*pareq.csd* [examples/pareq.csd]

*rbjeq.csd* [examples/rbjeq.csd]

**Modificateurs de signal : guides d'onde.**

*wguide1.csd* [examples/wguide1.csd]

*wguide2.csd* [examples/wguide2.csd]

**Modificateurs de signal : distorsion non-linéaire.**

*chebyshevpoly.csd* [examples/chebyshevpoly.csd]

*pdclip.csd* [examples/pdclip.csd]

*pdhalf.csd* [examples/pdhalf.csd]

*pdhalfy.csd* [examples/pdhalfy.csd]

*powershape.csd* [examples/powershape.csd]

**Modificateurs de signal : comparateurs et accumulateurs.**

*cmp.csd* [examples/cmp.csd]

*max.csd* [examples/max.csd]

*max\_k.csd* [examples/max\_k.csd]

*maxabs.csd* [examples/maxabs.csd]

*maxabsaccum.csd* [examples/maxabsaccum.csd]

*maxaccum.csd* [examples/maxaccum.csd]

*min.csd* [examples/min.csd]

*minabs.csd* [examples/minabs.csd]

*minabsaccum.csd* [examples/minabsaccum.csd]

*minaccum.csd* [examples/minaccum.csd]

**Contrôle d'instrument : contrôle d'horloge.**

*clockoff.csd* [examples/clockoff.csd]

*clockon.csd* [examples/clockon.csd]

**Contrôle d'instrument : valeurs conditionnelles.**

*equals.csd* [examples>equals.csd]

*greaterqual.csd* [examples/greaterqual.csd]

*greaterthan.csd* [examples/greaterthan.csd]

*lessequal.csd* [examples/lessequal.csd]

*lessthan.csd* [examples/lessthan.csd]

*notequal.csd* [examples/notequal.csd]

**Contrôle d'instrument : compilation.**

*compilecsd.csd* [examples/compilecsd.csd]

*compileorc.csd* [examples/compileorc.csd]

*compilestr.csd* [examples/compilestr.csd]

**Contrôle d'instrument : contrôle de durée.**

*ihold.csd* [examples/ihold.csd]

*turnoff.csd* [examples/turnoff.csd]

*turnoff2.csd* [examples/turnoff2.csd]

**Contrôle d'instrument : appel d'instrument.**

*event.csd* [examples/event.csd]

*event\_named.csd* [examples/event\_named.csd]

*event\_i.csd* [examples/event\_i.csd]

*mute.csd* [examples/mute.csd]

*nstance.csd* [examples/nstance.csd]

*readscore.csd* [examples/readscore.csd]

*schedkwhen.csd* [examples/schedkwhen.csd]

*schedkwhennamed.csd* [examples/schedkwhennamed.csd]

*schedule.csd* [examples/schedule.csd]

*schedule.csd* [examples/schedule.csd]

*schedwhen.csd* [examples/schedwhen.csd]

*scoreline.csd* [examples/scoreline.csd]

*scoreline\_i.csd* [examples/scoreline\_i.csd]

**Contrôle d'instrument : contrôle séquentiel d'un programme.**

*cggoto.csd* [examples/cggoto.csd]  
*cigoto.csd* [examples/cigoto.csd]  
*ckgoto.csd* [examples/ckgoto.csd]  
*cngoto.csd* [examples/cngoto.csd]  
*else.csd* [examples/else.csd]  
*elseif.csd* [examples/elseif.csd]  
*endif.csd* [examples/endif.csd]  
*goto.csd* [examples/goto.csd]  
*igoto.csd* [examples/igoto.csd]  
*kgoto.csd* [examples/kgoto.csd]  
*ifthen.csd* [examples/ifthen.csd]  
*igoto.csd* [examples/igoto.csd]  
*kgoto.csd* [examples/kgoto.csd]  
*loop\_le.csd* [examples/loop\_le.csd]  
*loop\_lt.csd* [examples/loop\_lt.csd]  
*tigoto.csd* [examples/tigoto.csd]  
*timeout.csd* [examples/timeout.csd]  
*until.csd* [examples/until.csd]  
*while.csd* [examples/while.csd]

**Contrôle d'instrument : controle de l'exécution en temps réel.**

*active.csd* [examples/active.csd]  
*active\_k.csd* [examples/active\_k.csd]  
*active\_scale.csd* [examples/active\_scale.csd]  
*cpumeter.csd* [examples/cpumeter.csd]  
*cpuprc.csd* [examples/cpuprc.csd]  
*exitnow.csd* [examples/exitnow.csd]  
*jacktransport.csd* [examples/jacktransport.csd]  
*maxalloc.csd* [examples/maxalloc.csd]  
*prealloc.csd* [examples/prealloc.csd]

**Contrôle d'instrument : initialisation et réinitialisation.**

*assign.csd* [examples/assign.csd]

*init.csd* [examples/init.csd]

*p.csd* [examples/p.csd]

*reverb.csd* [examples/reverb.csd]

*pset.csd* [examples/pset.csd]

*pset-midi.csd* [examples/pset-midi.csd]

*reinit.csd* [examples/reinit.csd]

*reinit.csd* [examples/reinit.csd]

*tival.csd* [examples/tival.csd]

**Contrôle d'instrument : détection et contrôle.**

*checkbox.csd* [examples/checkbox.csd]

*changed.csd* [examples/changed.csd]

*changed2.csd* [examples/changed2.csd]

*changed2a.csd* [examples/changed2a.csd]

*checkbox.csd* [examples/checkbox.csd]

*follow.csd* [examples/follow.csd]

*follow2.csd* [examples/follow2.csd]

*getcfd.csd* [examples/getcfd.csd]

*joystick.csd* [examples/joystick.csd]

*joystick-2.csd* [examples/joystick-2.csd]

*metro.csd* [examples/metro.csd]

*metro-2.csd* [examples/metro-2.csd]

*metro2.csd* [examples/metro2.csd]

*miditempo.csd* [examples/miditempo.csd]

*p5g.csd* [examples/p5g.csd]

*pcount.csd* [examples/pcount.csd]

*peak.csd* [examples/peak.csd]

*pindex.csd* [examples/pindex.csd]

*pindex-2.csd* [examples/pindex-2.csd]  
*pitch.csd* [examples/pitch.csd]  
*pitchamdf.csd* [examples/pitchamdf.csd]  
*plltrack.csd* [examples/plltrack.csd]  
*ptrack.csd* [examples/ptrack.csd]  
*readscratch.csd* [examples/readscratch.csd]  
*rewindscore.csd* [examples/rewindscore.csd]  
*rms.csd* [examples/rms.csd]  
*sensekey.csd* [examples/sensekey.csd]  
*FLpanel-sensekey.csd* [examples/FLpanel-sensekey.csd]  
*FLpanel-sensekey2.csd* [examples/FLpanel-sensekey2.csd]  
*seqtime.csd* [examples/seqtime.csd]  
*seqtime2.csd* [examples/seqtime2.csd]  
*setctrl.csd* [examples/setctrl.csd]  
*setscorepos.csd* [examples/setscorepos.csd]  
*tempest.csd* [examples/tempest.csd]  
*tempo.csd* [examples/tempo.csd]  
*tempoval.csd* [examples/tempoval.csd]  
*timedseq.csd* [examples/timedseq.csd]  
*trigger.csd* [examples/trigger.csd]  
*trigseq.csd* [examples/trigseq.csd]  
*vactrol.csd* [examples/vactrol.csd]  
*wii.csd* [examples/wii.csd]  
*readscratch.csd* [examples/readscratch.csd]  
*xyin.csd* [examples/xyin.csd]

**Contrôle d'instrument : piles.**

*pop.csd* [examples/pop.csd]  
*push.csd* [examples/push.csd]  
*stack.csd* [examples/stack.csd]

**Contrôle d'instrument : contrôle de sous-instrument.**

*subinstr.csd* [examples/subinstr.csd]

*subinstr\_named.csd* [examples/subinstr\_named.csd]

**Contrôle d'instrument : lecture du temps.**

*date.csd* [examples/date.csd]

*dates.csd* [examples/dates.csd]

*readclock.csd* [examples/readclock.csd]

*rtclock.csd* [examples/rtclock.csd]

*timeinstk.csd* [examples/timeinstk.csd]

*timeinsts.csd* [examples/timeinsts.csd]

*timek.csd* [examples/timek.csd]

*times\_complex.csd* [examples/times\_complex.csd]

**Opcodes jacko.**

*JackoInfo.csd* [examples/JackoInfo.csd]

*JackoInit.csd* [examples/JackoInit.csd]

**Contrôle des tables de fonction.**

*ftfree.csd* [examples/ftfree.csd]

*ftgen.csd* [examples/ftgen.csd]

*ftgen-2.csd* [examples/ftgen-2.csd]

*ftgentmp.csd* [examples/ftgentmp.csd]

*getftargs.csd* [examples/getftargs.csd]

**Contrôle des tables de fonction : requêtes sur une table.**

*array.csd* [examples/array.csd]

*fillarray.csd* [examples/fillarray.csd]

*ftchnls.csd* [examples/ftchnls.csd]

*ftcps.csd* [examples/ftcps.csd]

*ftexists.csd* [examples/ftexists.csd]

*flen.csd* [examples/flen.csd]

*ftlptim.csd* [examples/ftlptim.csd]

*ftsr.csd* [examples/ftsr.csd]

*genarray.csd* [examples/genarray.csd]

*genarray\_i.csd* [examples/genarray\_i.csd]

*lenarray.csd* [examples/lenarray.csd]

*maparray.csd* [examples/maparray.csd]

*nsamp.csd* [examples/nsamp.csd]

*slicearray.csd* [examples/slicearray.csd]

*tableng.csd* [examples/tableng.csd]

*tabsum.csd* [examples/tabsum.csd]

**Contrôle des tables de fonction : sélection dynamique.**

*tableikt.csd* [examples/tableikt.csd]

*tablekt.csd* [examples/tablekt.csd]

*tablexkt.csd* [examples/tablexkt.csd]

**Contrôle des tables de fonction : opérations de lecture/écriture.**

*ftaudio.csd* [examples/ftaudio.csd]

*ftprint.csd* [examples/ftprint.csd]

*ftsamplbank.csd* [examples/ftsamplbank.csd]

*ftsav.csd* [examples/ftsav.csd]

*tablecopy.csd* [examples/tablecopy.csd]

*tablefilter.csd* [examples/tablefilter.csd]

*tablefilter.csd* [examples/tablefilter.csd]

*tableimix.csd* [examples/tableimix.csd]

*tablemix.csd* [examples/tablemix.csd]

*tabmorph.csd* [examples/tabmorph.csd]

*tabmorpha.csd* [examples/tabmorpha.csd]

*tabmorphak.csd* [examples/tabmorphak.csd]

*tabmorphi.csd* [examples/tabmorphi.csd]

*tabrec.csd* [examples/tabrec.csd]

**FLTK : conteneurs.**

*FLpanel.csd* [examples/FLpanel.csd]

*FLscroll.csd* [examples/FLscroll.csd]

*FLtabs.csd* [examples/FLtabs.csd]

**FLTK : valueurs.**

*FLcount.csd* [examples/FLcount.csd]

*FLjoy.csd* [examples/FLjoy.csd]

*FLknob.csd* [examples/FLknob.csd]

*FLknob-2.csd* [examples/FLknob-2.csd]

*FLroller.csd* [examples/FLroller.csd]

*FLslider.csd* [examples/FLslider.csd]

*FLslider-2.csd* [examples/FLslider-2.csd]

*FLtext.csd* [examples/FLtext.csd]

**FLTK : autres.**

*FLbox.csd* [examples/FLbox.csd]

*FLbutBank.csd* [examples/FLbutBank.csd]

*FLbutton.csd* [examples/FLbutton.csd]

*FLexecButton.csd* [examples/FLexecButton.csd]

*FLhvsBox.csd* [examples/FLhvsBox.csd]

*FLhvsBoxSetValue.csd* [examples/FLhvsBoxSetValue.csd]

*FLkeyIn.csd* [examples/FLkeyIn.csd]

*FLmouse.csd* [examples/FLmouse.csd]

*FLsavesnap\_simple.csd* [examples/FLsavesnap\_simple.csd]

*FLsavesnap.csd* [examples/FLsavesnap.csd]

*FLslidBnk.csd* [examples/FLslidBnk.csd]

*FLslidBnk2.csd* [examples/FLslidBnk2.csd]

*FLslidBnk2Set.csd* [examples/FLslidBnk2Set.csd]

*FLslidBnk2Setk.csd* [examples/FLslidBnk2Setk.csd]



*FLslidBnkGetHandle.csd* [examples/FLslidBnkGetHandle.csd]

*FLslidBnkSet.csd* [examples/FLslidBnkSet.csd]

*FLslidBnkSetk.csd* [examples/FLslidBnkSetk.csd]

*FLvalue.csd* [examples/FLvalue.csd]

*FLvslidBnk.csd* [examples/FLvslidBnk.csd]

*FLvslidBnk2.csd* [examples/FLvslidBnk2.csd]

*FLxyin.csd* [examples/FLxyin.csd]

*FLxyin-2.csd* [examples/FLxyin-2.csd]

*vphaseseg.csd* [examples/vphaseseg.csd]

### **FLTK : apparence.**

*FLsetcolor.csd* [examples/FLsetcolor.csd]

*FLsetText.csd* [examples/FLsetText.csd]

### **Opérations mathématiques : opérations arithmétiques et logiques.**

*adds.csd* [examples/adds.csd]

*divides.csd* [examples/divides.csd]

*modulus.csd* [examples/modulus.csd]

*multiplies.csd* [examples/multiplies.csd]

*opand.csd* [examples/opand.csd]

*bitwise.csd* [examples/bitwise.csd]

*bitshift.csd* [examples/bitshift.csd]

*opnot.csd* [examples/opnot.csd]

*logicOR.csd* [examples/logicOR.csd]

*raises.csd* [examples/raises.csd]

*subtracts.csd* [examples/subtracts.csd]

### **Opérations mathématiques : comparateurs et accumulateurs.**

*clear.csd* [examples/clear.csd]

*vincr.csd* [examples/vincr.csd]

*vincr-complex.csd* [examples/vincr-complex.csd]

**Opérations mathématiques : fonctions mathématiques.**

*abs.csd* [examples/abs.csd]

*ceil.csd* [examples/ceil.csd]

*exp.csd* [examples/exp.csd]

*floor.csd* [examples/floor.csd]

*frac.csd* [examples/frac.csd]

*int.csd* [examples/int.csd]

*log.csd* [examples/log.csd]

*log10.csd* [examples/log10.csd]

*log2.csd* [examples/log2.csd]

*logbtwo.csd* [examples/logbtwo.csd]

*powoftwo.csd* [examples/powoftwo.csd]

*qinf.csd* [examples/qinf.csd]

*qnan.csd* [examples/qnan.csd]

*round.csd* [examples/round.csd]

*sqrt.csd* [examples/sqrt.csd]

**Opérations mathématiques : fonctions trigonométriques.**

*cos.csd* [examples/cos.csd]

*cosh.csd* [examples/cosh.csd]

*cosinv.csd* [examples/cosinv.csd]

*signum.csd* [examples/signum.csd]

*sin.csd* [examples/sin.csd]

*sinh.csd* [examples/sinh.csd]

*sininv.csd* [examples/sininv.csd]

*tan.csd* [examples/tan.csd]

*tanh.csd* [examples/tanh.csd]

*taninv.csd* [examples/taninv.csd]

**Opérations mathématiques : fonctions d'amplitude.**

*ampdb.csd* [examples/ampdb.csd]

*ampdbfs.csd* [examples/ampdbfs.csd]

*db.csd* [examples/db.csd]

*dbamp.csd* [examples/dbamp.csd]

*dbfsamp.csd* [examples/dbfsamp.csd]

**Opérations mathématiques : fonctions aléatoires.**

*birnd.csd* [examples/birnd.csd]

*rnd.csd* [examples/rnd.csd]

**Opérations mathématiques : opcodes équivalents à des fonctions.**

*divz.csd* [examples/divz.csd]

*mac.csd* [examples/mac.csd]

*maca.csd* [examples/maca.csd]

*polynomial.csd* [examples/polynomial.csd]

*pow.csd* [examples/pow.csd]

*product.csd* [examples/product.csd]

*sum.csd* [examples/sum.csd]

*taninv2.csd* [examples/taninv2.csd]

**Conversion des hauteurs : fonctions.**

*cent.csd* [examples/cent.csd]

*cpsmidinn.csd* [examples/cpsmidinn.csd]

*cpsmidinn2.csd* [examples/cpsmidinn2.csd]

*cpsoct.csd* [examples/cpsoct.csd]

*cpspch.csd* [examples/cpspch.csd]

*mtof-ftom.csd* [examples/mtof-ftom.csd]

*mtof-ftom.csd* [examples/mtof-ftom.csd]

*mton-ntom.csd* [examples/mton-ntom.csd]

*ntof.csd* [examples/ntof.csd]

*mton-ntom.csd* [examples/mton-ntom.csd]

*octave.csd* [examples/octave.csd]

*octcps.csd* [examples/octcps.csd]

*cpsmidinn.csd* [examples/cpsmidinn.csd]

*octpch.csd* [examples/octpch.csd]

*cpsmidinn.csd* [examples/cpsmidinn.csd]

*pchoct.csd* [examples/pchoct.csd]

*phtom.csd* [examples/phtom.csd]

*semitone.csd* [examples/semitone.csd]

**Conversion des hauteurs : opcodes de hauteurs.**

*cps2pch.csd* [examples/cps2pch.csd]

*cps2pch\_ftable.csd* [examples/cps2pch\_ftable.csd]

*cps2pch\_19et.csd* [examples/cps2pch\_19et.csd]

*cpstun.csd* [examples/cpstun.csd]

*cpstuni.csd* [examples/cpstuni.csd]

*cpsxpch.csd* [examples/cpsxpch.csd]

*cpsxpch\_105et.csd* [examples/cpsxpch\_105et.csd]

*cpsxpch\_pierce.csd* [examples/cpsxpch\_pierce.csd]

**MIDI en temps réel : entrée.**

*aftouch.csd* [examples/aftouch.csd]

*chanctrl.csd* [examples/chanctrl.csd]

*ctrl14.csd* [examples/ctrl14.csd]

*ctrl21.csd* [examples/ctrl21.csd]

*ctrl7.csd* [examples/ctrl7.csd]

*initc7.csd* [examples/initc7.csd]

*massign.csd* [examples/massign.csd]

*midic7.csd* [examples/midic7.csd]

*midictrl.csd* [examples/midictrl.csd]

*notnum.csd* [examples/notnum.csd]

*pchbend.csd* [examples/pchbend.csd]

*pgmassign.csd* [examples/pgmassign.csd]

*pgmassign\_ignore.csd* [examples/pgmassign\_ignore.csd]

*pgmassign\_advanced.csd* [examples/pgmassign\_advanced.csd]

*polyaft.csd* [examples/polyaft.csd]

*veloc.csd* [examples/veloc.csd]

**MIDI en temps réel : sortie.**

*nrpn.csd* [examples/nrpn.csd]

*outiat.csd* [examples/outiat.csd]

*outic.csd* [examples/outic.csd]

*outipb.csd* [examples/outipb.csd]

*outipc.csd* [examples/outipc.csd]

*outkat.csd* [examples/outkat.csd]

*outkc.csd* [examples/outkc.csd]

*outkpb.csd* [examples/outkpb.csd]

*outkpc.csd* [examples/outkpc.csd]

*outkpc\_fltk.csd* [examples/outkpc\_fltk.csd]

**MIDI en temps réel : convertisseurs.**

*ampmidi.csd* [examples/ampmidi.csd]

*ampmidicurve.csd* [examples/ampmidicurve.csd]

*ampmidid.csd* [examples/ampmidid.csd]

*cpsmidi.csd* [examples/cpsmidi.csd]

*cpsmidib.csd* [examples/cpsmidib.csd]

*cpstmid.csd* [examples/cpstmid.csd]

*octmidi.csd* [examples/octmidi.csd]

*octmidib.csd* [examples/octmidib.csd]

*pchmidi.csd* [examples/pchmidi.csd]

*pchmidib.csd* [examples/pchmidib.csd]

**MIDI en temps réel : E/S génériques.**

*midiiin.csd* [examples/midiin.csd]

*midiout.csd* [examples/midiout.csd]

*midiout\_i.csd* [examples/midiout\_i.csd]

**MIDI en temps réel : extension d'évènements.**

*lastcycle.csd* [examples/lastcycle.csd]

*xtratim.csd* [examples/xtratim.csd]

*xtratim-2.csd* [examples/xtratim-2.csd]

**MIDI en temps réel : sortie de note.**

*midiarp.csd* [examples/midiarp.csd]

*midion\_simple.csd* [examples/midion\_simple.csd]

*midion\_scale.csd* [examples/midion\_scale.csd]

*midion2.csd* [examples/midion2.csd]

*moscil.csd* [examples/moscil.csd]

*noteondur.csd* [examples/noteondur.csd]

*noteondur2.csd* [examples/noteondur2.csd]

**MIDI en temps réel : interopérabilité MIDI/partition.**

*midichannelaftertouch.csd* [examples/midichannelaftertouch.csd]

*midichn.csd* [examples/midichn.csd]

*midichn\_advanced.csd* [examples/midichn\_advanced.csd]

*midicontrolchange.csd* [examples/midicontrolchange.csd]

*mididefault.csd* [examples/mididefault.csd]

*midinoteoff.csd* [examples/midinoteoff.csd]

*midinoteoncps.csd* [examples/midinoteoncps.csd]

*midinoteonkey.csd* [examples/midinoteonkey.csd]

*midinoteonoct.csd* [examples/midinoteonoct.csd]

*midinoteonpch.csd* [examples/midinoteonpch.csd]

*midipitchbend.csd* [examples/midipitchbend.csd]

*midipolyaftertouch.csd* [examples/midipolyaftertouch.csd]

*midiprogramchange.csd* [examples/midiprogramchange.csd]

**MIDI en temps réel : système temps réel.**

*mclock.csd* [examples/mclock.csd]

**Graphe de fluence.**

*alwayson.csd* [examples/alwayson.csd]

*connect.csd* [examples/connect.csd]

*ftgenonce.csd* [examples/ftgenonce.csd]

*inleta.csd* [examples/inleta.csd]

*inletk.csd* [examples/inletk.csd]

*outleta.csd* [examples/outleta.csd]

*outletk.csd* [examples/outletk.csd]

**Traitement spectral : STFT.**

*pvadd.csd* [examples/pvadd.csd]

*pdbufread.csd* [examples/pdbufread.csd]

*pvcross.csd* [examples/pvcross.csd]

*pvinterp.csd* [examples/pvinterp.csd]

*pvoc.csd* [examples/pvoc.csd]

*pvread.csd* [examples/pvread.csd]

*tableseg.csd* [examples/tableseg.csd]

*tablexseg.csd* [examples/tablexseg.csd]

*vpvoc.csd* [examples/vpvoc.csd]

**Traitement spectral : LPC.**

*lpfreson.csd* [examples/lpfreson.csd]

*lpread.csd* [examples/lpread.csd]

*lpreson.csd* [examples/lpreson.csd]

*lpreson-2.csd* [examples/lpreson-2.csd]

**Traitement spectral : streaming.**

*binit.csd* [examples/binit.csd]

*cudanal.csd* [examples/cudanal.csd]

*cuDasliding.csd* [examples/cuDasliding.csd]  
*cuDasynth.csd* [examples/cuDasynth.csd]  
*partials.csd* [examples/partials.csd]  
*pvsadsyn.csd* [examples/pvsadsyn.csd]  
*pvsanal.csd* [examples/pvsanal.csd]  
*pvsarp.csd* [examples/pvsarp.csd]  
*pvsarp2.csd* [examples/pvsarp2.csd]  
*pvsbandp.csd* [examples/pvsbandp.csd]  
*pvsbandr.csd* [examples/pvsbandr.csd]  
*pvsbin.csd* [examples/pvsbin.csd]  
*pvsblur.csd* [examples/pvsblur.csd]  
*pvsbufread.csd* [examples/pvsbufread.csd]  
*pvsbufread2.csd* [examples/pvsbufread2.csd]  
*pvscale.csd* [examples/pvscale.csd]  
*pvscent.csd* [examples/pvscent.csd]  
*pvsceps.csd* [examples/pvsceps.csd]  
*pvscompress.csd* [examples/pvscompress.csd]  
*pvsdiskin.csd* [examples/pvsdiskin.csd]  
*pvsdisp.csd* [examples/pvsdisp.csd]  
*pvsfilter.csd* [examples/pvsfilter.csd]  
*pvsfread.csd* [examples/pvsfread.csd]  
*pvsfreeze.csd* [examples/pvsfreeze.csd]  
*pvsftr.csd* [examples/pvsftr.csd]  
*pvsftw.csd* [examples/pvsftw.csd]  
*pvsfwrite.csd* [examples/pvsfwrite.csd]  
*pvsgain.csd* [examples/pvsgain.csd]  
*pvsift.csd* [examples/pvsift.csd]  
*pvsinfo.csd* [examples/pvsinfo.csd]  
*pvslock.csd* [examples/pvslock.csd]



*pvsmaska.csd* [examples/pvsmaska.csd]  
*pvmix.csd* [examples/pvmix.csd]  
*pvsMOOTH.csd* [examples/pvsMOOTH.csd]  
*pvsMORPH.csd* [examples/pvsMORPH.csd]  
*pvsMORPH2.csd* [examples/pvsMORPH2.csd]  
*pvsOSC.csd* [examples/pvsOSC.csd]  
*pvsPITCH.csd* [examples/pvsPITCH.csd]  
*pvsTANAL.csd* [examples/pvsTANAL.csd]  
*pvsTRACE.csd* [examples/pvsTRACE.csd]  
*pvsVOC.csd* [examples/pvsVOC.csd]  
*pvsWARP.csd* [examples/pvsWARP.csd]  
*pvsYNTH.csd* [examples/pvsYNTH.csd]  
*resyn.csd* [examples/resyn.csd]  
*sinsyn.csd* [examples/sinsyn.csd]  
*tabifd.csd* [examples/tabifd.csd]  
*tradsyn.csd* [examples/tradsyn.csd]  
*trcross.csd* [examples/trcross.csd]  
*trfilter.csd* [examples/trfilter.csd]  
*trhighest.csd* [examples/trhighest.csd]  
*trlowest.csd* [examples/trlowest.csd]  
*trmix.csd* [examples/trmix.csd]  
*trscale.csd* [examples/trscale.csd]  
*trshift.csd* [examples/trshift.csd]  
*trsPLIT.csd* [examples/trsPLIT.csd]

**Traitement spectral : ATS.**

*ATSadd.csd* [examples/ATSadd.csd]  
*ATSadd-2.csd* [examples/ATSadd-2.csd]  
*ATSaddnz.csd* [examples/ATSaddnz.csd]  
*ATSaddnz-2.csd* [examples/ATSaddnz-2.csd]

*ATSbufread.csd* [examples/ATSbufread.csd]

*ATScross.csd* [examples/ATScross.csd]

*ATSinfo.csd* [examples/ATSinfo.csd]

*ATSinfo-2.csd* [examples/ATSinfo-2.csd]

*ATSinterpread.csd* [examples/ATSinterpread.csd]

*ATSpartialtap.csd* [examples/ATSpartialtap.csd]

*ATSread.csd* [examples/ATSread.csd]

*ATSread-2.csd* [examples/ATSread-2.csd]

*ATSreadnz.csd* [examples/ATSreadnz.csd]

*ATSinnoi.csd* [examples/ATSinnoi.csd]

*ATSinnoi-2.csd* [examples/ATSinnoi-2.csd]

#### **Traitement spectral : Loris.**

*lorismorph.csd* [examples/lorismorph.csd]

*lorisplay.csd* [examples/lorisplay.csd]

*lorisread.csd* [examples/lorisread.csd]

#### **Traitement spectral : autres.**

*centroid.csd* [examples/centroid.csd]

*filescale.csd* [examples/filescale.csd]

*mincer.csd* [examples/mincer.csd]

*mp3scale.csd* [examples/mp3scale.csd]

*paulstretch.csd* [examples/paulstretch.csd]

#### **Chaînes : définition.**

*strfromurl.csd* [examples/strfromurl.csd]

*strget.csd* [examples/strget.csd]

*strset.csd* [examples/strset.csd]

#### **Chaînes : manipulation.**

*puts.csd* [examples/puts.csd]

*sprintf.csd* [examples/sprintf.csd]

*sprintfk.csd* [examples/sprintfk.csd]

*strcat.csd* [examples/strcat.csd]

*strcatk.csd* [examples/strcatk.csd]

*strcmp.csd* [examples/strcmp.csd]

*strcpyk.csd* [examples/strcpyk.csd]

*strindexk.csd* [examples/strindexk.csd]

*strrindex.csd* [examples/strrindex.csd]

*rstrip.csd* [examples/rstrip.csd]

*strsub.csd* [examples/strsub.csd]

### **Chaînes : conversion.**

*strchar.csd* [examples/strchar.csd]

*strlower.csd* [examples/strlower.csd]

*strtod.csd* [examples/strtod.csd]

*strtodk.csd* [examples/strtodk.csd]

*strtol.csd* [examples/strtol.csd]

*strtolk.csd* [examples/strtolk.csd]

### **Vectorel : tableaux.**

*vtable1k.csd* [examples/vtable1k.csd]

*vtablei.csd* [examples/vtablei.csd]

*vtablek.csd* [examples/vtablek.csd]

*vtablewa.csd* [examples/vtablewa.csd]

*vtablewk.csd* [examples/vtablewk.csd]

### **Vectorel : opérations scalaires.**

*vadd.csd* [examples/vadd.csd]

*vadd\_i.csd* [examples/vadd\_i.csd]

*vexp.csd* [examples/vexp.csd]

*vexp\_i.csd* [examples/vexp\_i.csd]

*vmult-2.csd* [examples/vmult-2.csd]

*vmult.csd* [examples/vmult.csd]

*vmult\_i.csd* [examples/vmult\_i.csd]

*vpow.csd* [examples/vpow.csd]

*vpow-2.csd* [examples/vpow-2.csd]

*vpow\_i.csd* [examples/vpow\_i.csd]

*vpow\_i-2.csd* [examples/vpow\_i-2.csd]

**Vectoriel : opérations vectorielles.**

*vaddv.csd* [examples/vaddv.csd]

*vcopy.csd* [examples/vcopy.csd]

*vdivv.csd* [examples/vdivv.csd]

*vexpv.csd* [examples/vexpv.csd]

*vmap.csd* [examples/vmap.csd]

*vmultv.csd* [examples/vmultv.csd]

*vpowv.csd* [examples/vpowv.csd]

*vsubv.csd* [examples/vsubv.csd]

**Vectoriel : enveloppes.**

*vexpseg.csd* [examples/vexpseg.csd]

*vlinseg.csd* [examples/vlinseg.csd]

**Vectoriel : aléatoire.**

*vrandh.csd* [examples/vrandh.csd]

*vrandi.csd* [examples/vrandi.csd]

**Vectoriel : automates cellulaires.**

*cell.csd* [examples/cell.csd]

*vcella.csd* [examples/vcella.csd]

**Système de patch zak.**

*zacl.csd* [examples/zacl.csd]

*zakinit.csd* [examples/zakinit.csd]

*zamod.csd* [examples/zamod.csd]

*zar.csd* [examples/zar.csd]

*zarg.csd* [examples/zarg.csd]

*zaw.csd* [examples/zaw.csd]

*zawm.csd* [examples/zawm.csd]

*zir.csd* [examples/zir.csd]

*ziw.csd* [examples/ziw.csd]

*ziwm.csd* [examples/ziwm.csd]

*zkcl.csd* [examples/zkcl.csd]

*zkmod.csd* [examples/zkmod.csd]

*zkr.csd* [examples/zkr.csd]

*zkw.csd* [examples/zkw.csd]

*zkwm.csd* [examples/zkwm.csd]

#### **Accueil de greffon : DSSI et LADSPA.**

*dssiactivate.csd* [examples/dssiactivate.csd]

*dssiaudio.csd* [examples/dssiaudio.csd]

*dssictls.csd* [examples/dssictls.csd]

*dssiinit.csd* [examples/dssiinit.csd]

*dssilist.csd* [examples/dssilist.csd]

#### **OSC.**

*oscbundle.csd* [examples/oscbundle.csd]

*OSCmidisend.csd* [examples/OSCmidisend.csd]

*OSCmidircv.csd* [examples/OSCmidircv.csd]

*OSCrw.csd* [examples/OSCrw.csd]

#### **Opcodes Faust.**

*faustaudio.csd* [examples/faustaudio.csd]

*faustcompile.csd* [examples/faustcompile.csd]

*faustctl.csd* [examples/faustctl.csd]

*faustdsp.csd* [examples/faustdsp.csd]

*faustgen.csd* [examples/faustgen.csd]

*faustplay.csd* [examples/faustplay.csd]

**Opcodes pour le traitement à distance.**

*insremot.csd* [examples/insremot.csd]

*insremotM.csd* [examples/insremotM.csd]

*midremot.csd* [examples/midremot.csd]

**Opcodes Mixer.**

*Mixer.csd* [examples/Mixer.csd]

*Mixer.csd* [examples/Mixer.csd]

*Mixer.csd* [examples/Mixer.csd]

*Mixer.csd* [examples/Mixer.csd]

*Mixer.csd* [examples/Mixer.csd]

**Opcodes Ableton Link.**

*ableton\_link.csd* [examples/ableton\_link.csd]

*ableton\_link.csd* [examples/ableton\_link.csd]

*ableton\_link.csd* [examples/ableton\_link.csd]

*ableton\_link.csd* [examples/ableton\_link.csd]

*ableton\_link.csd* [examples/ableton\_link.csd]

*ableton\_link.csd* [examples/ableton\_link.csd]

*ableton\_link.csd* [examples/ableton\_link.csd]

**Opcodes pour le traitement d'image.**

*imageopcodes.csd* [examples/imageopcodes.csd]

*imageopcodes.csd* [examples/imageopcodes.csd]

*imageopcodesdemo2.csd* [examples/imageopcodesdemo2.csd]

*imageopcodes.csd* [examples/imageopcodes.csd]

*imageopcodes.csd* [examples/imageopcodes.csd]

*imageopcodes.csd* [examples/imageopcodes.csd]

*imageopcodes.csd* [examples/imageopcodes.csd]

**Opcodes de tableaux.**

*copya2ftab.csd* [examples/copya2ftab.csd]

*copyf2array.csd* [examples/copyf2array.csd]

*getcol.csd* [examples/getcol.csd]

*rfft.csd* [examples/rfft.csd]

*maxarray.csd* [examples/maxarray.csd]

*minarray.csd* [examples/minarray.csd]

*printarray.csd* [examples/printarray.csd]

*productarray.csd* [examples/productarray.csd]

*reshapearray.csd* [examples/reshapearray.csd]

*scalearray.csd* [examples/scalearray.csd]

*setcol.csd* [examples/setcol.csd]

*rfft.csd* [examples/rfft.csd]

*shiftin.csd* [examples/shiftin.csd]

*shiftout.csd* [examples/shiftout.csd]

*sumarray.csd* [examples/sumarray.csd]

*tab2array.csd* [examples/tab2array.csd]

*trim.csd* [examples/trim.csd]

**Opérations de tableau : transformée de Fourier rapide.**

*fft.csd* [examples/fft.csd]

*ifft.csd* [examples/ifft.csd]

*rfft.csd* [examples/rfft.csd]

*irfft.csd* [examples/irfft.csd]

*unwrap.csd* [examples/unwrap.csd]

*window.csd* [examples/window.csd]

**Opérations de tableau : nombres complexes.**

*c2r.csd* [examples/c2r.csd]

*cmplxprod.csd* [examples/cmplxprod.csd]

*mags.csd* [examples/mags.csd]

*phs.csd* [examples/phs.csd]

*pol2rect.csd* [examples/pol2rect.csd]

*r2c.csd* [examples/r2c.csd]

*rect2pol.csd* [examples/rect2pol.csd]

**Opérations de tableau : transformée cosinus discrète.**

*dct.csd* [examples/dct.csd]

*dctinv.csd* [examples/dctinv.csd]

**Opérations de tableau : produit scalaire.**

*dot.csd* [examples/dot.csd]

**Opérations de tableau : banc de filtres à échelle de Mel.**

*mfb.csd* [examples/mfb.csd]

**Opérations de tableau : tri.**

*sorta.csd* [examples/sorta.csd]

*sortd.csd* [examples/sortd.csd]

**Opérations de tableau : cepstre.**

*ceps.csd* [examples/ceps.csd]

*cepsinv.csd* [examples/cepsinv.csd]

**Opérations de tableau : obsolète.**

*tableiw.csd* [examples/tableiw.csd]

**Divers.**

*directory.csd* [examples/directory.csd]

*fareyleni.csd* [examples/fareyleni.csd]

*framebuffer.csd* [examples/framebuffer.csd]

*modmatrix.csd* [examples/modmatrix.csd]



*framebuffer.csd* [examples/framebuffer.csd]

*pwd.csd* [examples/pwd.csd]

*select.csd* [examples/select.csd]

*system.csd* [examples/system.csd]

*farey7shuffled.csd* [examples/farey7shuffled.csd]

---

# Annexe B. Conversion de hauteur

**Tableau B.1. Conversion de hauteur**

Note (anglais)	Note (français)	Hz	cpspch	MIDI
C-1	do-2	8.176	3.00	0
C#-1	do#-2	8.662	3.01	1
D-1	ré-2	9.177	3.02	2
D#-1	ré#-2	9.723	3.03	3
E-1	mi-2	10.301	3.04	4
F-1	fa-2	10.913	3.05	5
F#-1	fa#-2	11.562	3.06	6
G-1	sol-2	12.250	3.07	7
G#-1	sol#-2	12.978	3.08	8
A-1	la-2	13.750	3.09	9
A#-1	la#-2	14.568	3.10	10
B-1	si-2	15.434	3.11	11
C0	do-1	16.352	4.00	12
C#0	do#-1	17.324	4.01	13
D0	ré-1	18.354	4.02	14
D#0	ré#-1	19.445	4.03	15
E0	mi-1	20.602	4.04	16
F0	fa-1	21.827	4.05	17
F#0	fa#-1	23.125	4.06	18
G0	sol-1	24.500	4.07	19
G#0	sol#-1	25.957	4.08	20
A0	la-1	27.500	4.09	21
A#0	la#-1	29.135	4.10	22
B0	si-1	30.868	4.11	23
C1	do0	32.703	5.00	24
C#1	do#0	34.648	5.01	25
D1	ré0	36.708	5.02	26
D#1	ré#0	38.891	5.03	27
E1	mi0	41.203	5.04	28
F1	fa0	43.654	5.05	29
F#1	fa#0	46.249	5.06	30
G1	sol0	48.999	5.07	31
G#1	sol#0	51.913	5.08	32
A1	la0	55.000	5.09	33

Note (anglais)	Note (français)	Hz	cpspch	MIDI
A#1	la#0	58.270	5.10	34
B1	si0	61.735	5.11	35
C2	do1	65.406	6.00	36
C#2	do#1	69.296	6.01	37
D2	ré1	73.416	6.02	38
D#2	ré#1	77.782	6.03	39
E2	mi1	82.407	6.04	40
F2	fa1	87.307	6.05	41
F#2	fa#1	92.499	6.06	42
G2	sol1	97.999	6.07	43
G#2	sol#1	103.826	6.08	44
A2	la1	110.000	6.09	45
A#2	la#1	116.541	6.10	46
B2	si1	123.471	6.11	47
C3	do2	130.813	7.00	48
C#3	do#2	138.591	7.01	49
D3	ré2	146.832	7.02	50
D#3	ré#2	155.563	7.03	51
E3	mi2	164.814	7.04	52
F3	fa2	174.614	7.05	53
F#3	fa#2	184.997	7.06	54
G3	sol2	195.998	7.07	55
G#3	sol#2	207.652	7.08	56
A3	la2	220.000	7.09	57
A#3	la#2	233.082	7.10	58
B3	si2	246.942	7.11	59
C4	do3	261.626	8.00	60
C#4	do#3	277.183	8.01	61
D4	ré3	293.665	8.02	62
D#4	ré#3	311.127	8.03	63
E4	mi3	329.628	8.04	64
F4	fa3	349.228	8.05	65
F#4	fa#3	369.994	8.06	66
G4	sol3	391.995	8.07	67
G#4	sol#3	415.305	8.08	68
A4	la3	440.000	8.09	69
A#4	la#3	466.164	8.10	70
B4	si3	493.883	8.11	71

Note (anglais)	Note (français)	Hz	cpspch	MIDI
C5	do4	523.251	9.00	72
C#5	do#4	554.365	9.01	73
D5	ré4	587.330	9.02	74
D#5	ré#4	622.254	9.03	75
E5	mi4	659.255	9.04	76
F5	fa4	698.456	9.05	77
F#5	fa#4	739.989	9.06	78
G5	sol4	783.991	9.07	79
G#5	sol#4	830.609	9.08	80
A5	la4	880.000	9.09	81
A#5	la#4	932.328	9.10	82
B5	si4	987.767	9.11	83
C6	do5	1046.502	10.00	84
C#6	do#5	1108.731	10.01	85
D6	ré5	1174.659	10.02	86
D#6	ré#5	1244.508	10.03	87
E6	mi5	1318.510	10.04	88
F6	fa5	1396.913	10.05	89
F#6	fa#5	1479.978	10.06	90
G6	sol5	1567.982	10.07	91
G#6	sol#5	1661.219	10.08	92
A6	la5	1760.000	10.09	93
A#6	la#5	1864.655	10.10	94
B6	si5	1975.533	10.11	95
C7	do6	2093.005	11.00	96
C#7	do#6	2217.461	11.01	97
D7	ré6	2349.318	11.02	98
D#7	ré#6	2489.016	11.03	99
E7	mi6	2637.020	11.04	100
F7	fa6	2793.826	11.05	101
F#7	fa#6	2959.955	11.06	102
G7	sol6	3135.963	11.07	103
G#7	sol#6	3322.438	11.08	104
A7	la6	3520.000	11.09	105
A#7	la#6	3729.310	11.10	106
B7	si6	3951.066	11.11	107
C8	do7	4186.009	12.00	108
C#8	do#7	4434.922	12.01	109

Note (anglais)	Note (français)	Hz	cpspch	MIDI
D8	ré7	4698.636	12.02	110
D#8	ré#7	4978.032	12.03	111
E8	mi7	5274.041	12.04	112
F8	fa7	5587.652	12.05	113
F#8	fa#7	5919.911	12.06	114
G8	sol7	6271.927	12.07	115
G#8	sol#7	6644.875	12.08	116
A8	la7	7040.000	12.09	117
A#8	la#7	7458.620	12.10	118
B8	si7	7902.133	12.11	119
C9	do8	8372.018	13.00	120
C#9	do#8	8869.844	13.01	121
D9	ré8	9397.273	13.02	122
D#9	ré#8	9956.063	13.03	123
E9	mi8	10548.08	13.04	124
F9	fa8	11175.30	13.05	125
F#9	fa#8	11839.82	13.06	126
G9	sol8	12543.85	13.07	127

---

# Annexe C. Valeurs d'intensité du son

**Tableau C.1. Valeurs d'intensité du son (pour un ton pur à 1000 Hz)**

Dynamiques	Intensité (W/m ^ 2)	Niveau (dB)
douleur	1	120
fff	$10^{-2}$	100
f	$10^{-4}$	80
p	$10^{-6}$	60
ppp	$10^{-8}$	40
seuil d'audibilité	$10^{-12}$	0

---

# Annexe D. Valeurs de formant

**Tableau D.1. alto « a »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	800	1150	2800	3500	4950
amp (dB)	0	-4	-20	-36	-60
larg. bande (Hz)	80	90	120	130	140

**Tableau D.2. alto « e »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	400	1600	2700	3300	4950
amp (dB)	0	-24	-30	-35	-60
larg. bande (Hz)	60	80	120	150	200

**Tableau D.3. alto « i »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	350	1700	2700	3700	4950
amp (dB)	0	-20	-30	-36	-60
larg. bande (Hz)	50	100	120	150	200

**Tableau D.4. alto « o »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	450	800	2830	3500	4950
amp (dB)	0	-9	-16	-28	-55
larg. bande (Hz)	70	80	100	130	135

**Tableau D.5. alto « u »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	325	700	2530	3500	4950
amp (dB)	0	-12	-30	-40	-64
larg. bande (Hz)	50	60	170	180	200

**Tableau D.6. basse « a »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	600	1040	2250	2450	2750
amp (dB)	0	-7	-9	-9	-20
larg. bande (Hz)	60	70	110	120	130

**Tableau D.7. basse « e »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	400	1620	2400	2800	3100
amp (dB)	0	-12	-9	-12	-18
larg. bande (Hz)	40	80	100	120	120

**Tableau D.8. basse « i »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	250	1750	2600	3050	3340
amp (dB)	0	-30	-16	-22	-28
larg. bande (Hz)	60	90	100	120	120

**Tableau D.9. basse « o »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	400	750	2400	2600	2900
amp (dB)	0	-11	-21	-20	-40
larg. bande (Hz)	40	80	100	120	120

**Tableau D.10. basse « u »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	350	600	2400	2675	2950
amp (dB)	0	-20	-32	-28	-36
larg. bande (Hz)	40	80	100	120	120

**Tableau D.11. haute-contre « a »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	660	1120	2750	3000	3350
amp (dB)	0	-6	-23	-24	-38
larg. bande (Hz)	80	90	120	130	140

**Tableau D.12. haute-contre « e »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	440	1800	2700	3000	3300
amp (dB)	0	-14	-18	-20	-20
larg. bande (Hz)	70	80	100	120	120

**Tableau D.13. haute-contre « i »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	270	1850	2900	3350	3590



Valeurs	f1	f2	f3	f4	f5
amp (dB)	0	-24	-24	-36	-36
larg. bande (Hz)	40	90	100	120	120

**Tableau D.14. haute-contre « o »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	430	820	2700	3000	3300
amp (dB)	0	-10	-26	-22	-34
larg. bande (Hz)	40	80	100	120	120

**Tableau D.15. haute-contre « u »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	370	630	2750	3000	3400
amp (dB)	0	-20	-23	-30	-34
larg. bande (Hz)	40	60	100	120	120

**Tableau D.16. soprano « a »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	800	1150	2900	3900	4950
amp (dB)	0	-6	-32	-20	-50
larg. bande (Hz)	80	90	120	130	140

**Tableau D.17. soprano « e »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	350	2000	2800	3600	4950
amp (dB)	0	-20	-15	-40	-56
larg. bande (Hz)	60	100	120	150	200

**Tableau D.18. soprano « i »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	270	2140	2950	3900	4950
amp (dB)	0	-12	-26	-26	-44
larg. bande (Hz)	60	90	100	120	120

**Tableau D.19. soprano « o »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	450	800	2830	3800	4950
amp (dB)	0	-11	-22	-22	-50
larg. bande (Hz)	40	80	100	120	120

**Tableau D.20. soprano « u »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	325	700	2700	3800	4950
amp (dB)	0	-16	-35	-40	-60
larg. bande (Hz)	50	60	170	180	200

**Tableau D.21. ténor « a »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	650	1080	2650	2900	3250
amp (dB)	0	-6	-7	-8	-22
larg. bande (Hz)	80	90	120	130	140

**Tableau D.22. ténor « e »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	400	1700	2600	3200	3580
amp (dB)	0	-14	-12	-14	-20
larg. bande (Hz)	70	80	100	120	120

**Tableau D.23. ténor « i »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	290	1870	2800	3250	3540
amp (dB)	0	-15	-18	-20	-30
larg. bande (Hz)	40	90	100	120	120

**Tableau D.24. ténor « o »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	400	800	2600	2800	3000
amp (dB)	0	-10	-12	-12	-26
larg. bande (Hz)	70	80	100	130	135

**Tableau D.25. ténor « u »**

Valeurs	f1	f2	f3	f4	f5
fréq (Hz)	350	600	2700	2900	3300
amp (dB)	0	-20	-17	-14	-26
larg. bande (Hz)	40	60	100	120	120

---

# Annexe E. Rapports de fréquence modale

## Contribution de Scott Lindroth

John Bower, un étudiant de Scott Lindroth, a dressé cette liste de fréquences modales pour différents objets et matériaux. Certains modes fonctionnent mieux que d'autres, et la plupart ne donnent des résultats plausibles que dans un intervalle de fréquences particulier. Caveat emptor.

En général, les objets en bois ne sonneront pas "bois" à moins qu'un composant aléatoire ne soit présent dans le son (essayez les guides d'onde en bandes). Néanmoins, certains des objets en bois font aussi de merveilleux instruments métalliques.

Ces rapports peuvent être utiles avec des opcodes comme *mode* ou *streson*.

**Tableau E.1. Rapports de Fréquence Modale**

Instrument	Rapports de fréquence modale
Dahina (tabla)	[1, 2.89, 4.95, 6.99, 8.01, 9.02]
Bayan (tabla)	[1, 2.0, 3.01, 4.01, 4.69, 5.63]
Plaque en bois de Cèdre Rouge	[1, 1.47, 2.09, 2.56]
Plaque en bois de Séquoia	[1, 1.47, 2.11, 2.57]
Plaque en bois de Sapin de Douglas	[1, 1.42, 2.11, 2.47]
Barre uniforme en bois	[1, 2.572, 4.644, 6.984, 9.723, 12]
Barre uniforme en aluminium	[1, 2.756, 5.423, 8.988, 13.448, 18.680]
Xylophone	[1, 3.932, 9.538, 16.688, 24.566, 31.147]
Vibraphone 1	[1, 3.984, 10.668, 17.979, 23.679, 33.642]
Vibraphone 2	[1, 3.997, 9.469, 15.566, 20.863, 29.440]
Plaques de Chladni	([62, 107, 360, 460, 863] Hz +-2Hz) [1, 1.72581, 5.80645, 7.41935, 13.91935] rapports
Bol tibétain (180mm)	( [221, 614, 1145, 1804, 2577, 3456, 4419] Hz) 934g, 180mm [1, 2.77828, 5.18099, 8.16289, 11.66063, 15.63801, 19.99] rapports
Bol tibétain (152 mm)	([314, 836, 1519, 2360, 3341, 4462, 5696] Hz) 563g, 152mm [1, 2.66242, 4.83757, 7.51592, 10.64012, 14.21019, 18.14027] rapports
Bol tibétain (140 mm)	([528, 1460, 2704, 4122, 5694] Hz) 557g, 140mm [1, 2.76515, 5.12121, 7.80681, 10.78409] rapports
Ver de vin	[1, 2.32, 4.25, 6.63, 9.38]

Instrument	Rapports de fréquence modale
Petite cloche à main	<p>([1312.0, 1314.5, 2353.3, 2362.9, 3306.5, 3309.4, 3923.8, 3928.2, 4966.6, 4993.7, 5994.4, 6003.0, 6598.9, 6619.7, 7971.7, 7753.2, 8413.1, 8453.3, 9292.4, 9305.2, 9602.3, 9912.4] Hz)</p> <p>[ 1, 1.0019054878049, 1.7936737804878, 1.8009908536585, 2.5201981707317, 2.5224085365854, 2.9907012195122, 2.9940548780488, 3.7855182926829, 3.8061737804878, 4.5689024390244, 4.5754573170732, 5.0296493902439, 5.0455030487805, 6.0759908536585, 5.9094512195122, 6.4124237804878, 6.4430640243902, 7.0826219512195, 7.0923780487805, 7.3188262195122, 7.5551829268293 ] rapports</p>
Sphère en spinelle de diamètre 3.6675mm	<p>([977.25, 1003.16, 1390.13, 1414.93, 1432.84, 1465.34, 1748.48, 1834.20, 1919.90, 1933.64, 1987.20, 2096.48, 2107.10, 2202.08, 2238.40, 2280.10, 0 /*2290.53 calculated*/, 2400.88, 2435.85, 2507.80, 2546.30, 2608.55, 2652.35, 2691.70, 2708.00] Hz)</p> <p>[ 1, 1.026513174725, 1.4224916858532, 1.4478690202098, 1.4661959580455, 1.499452545408, 1.7891839345101, 1.8768994627782, 1.9645945254541, 1.9786543873113, 2.0334612432847, 2.1452852391916, 2.1561524686621, 2.2533435661294, 2.2905090816065, 2.3331798413917, 0, 2.4567715528268, 2.4925556408289, 2.5661806088514, 2.6055768738808, 2.6692760296751, 2.7140956766436, 2.7543617293425, 2.7710411870043 ] rapports</p>
Couvercle de pot	[ 1, 3.2, 6.23, 6.27, 9.92, 14.15] rapports

---

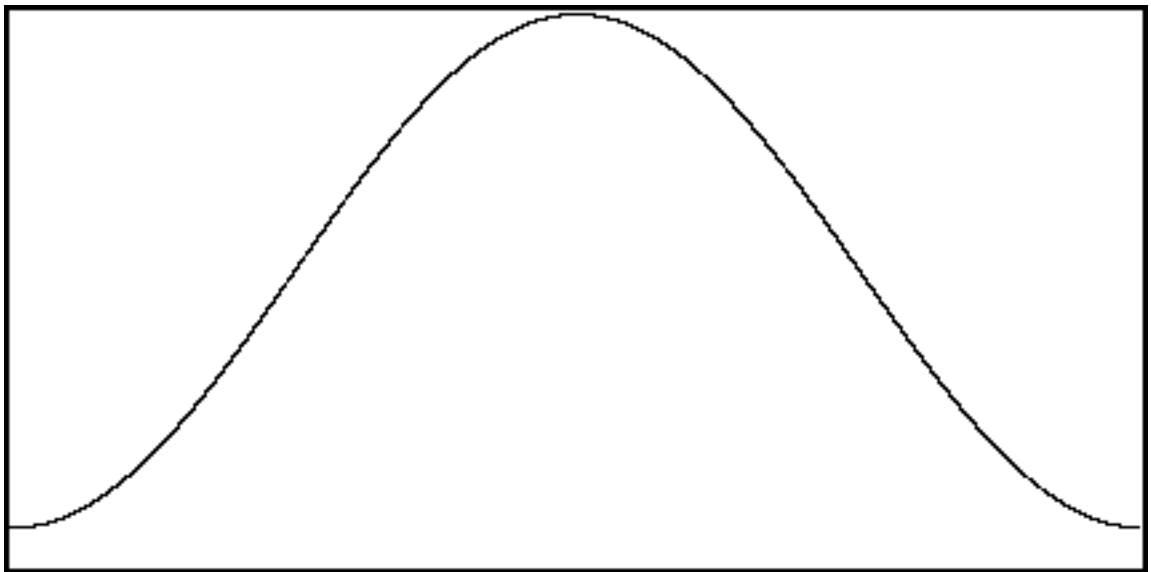
# Annexe F. Fonctions fenêtres

Les fonctions fenêtres sont utilisées pour l'analyse, et comme enveloppes de forme d'onde, particulièrement dans la synthèse granulaire. Les fonctions fenêtre sont intégrées à certains opcodes, mais d'autres opcodes nécessitent une table de fonction pour générer la fenêtre. *GEN20* est utilisé à cet effet. Le diagramme de chaque fenêtre ci-dessous est accompagné de l'instruction *f* utilisée pour la générer.

## Hamming.

### Exemple F.1. Instruction pour la fonction fenêtre de Hamming

```
f81 0 8192 20 1 1
```

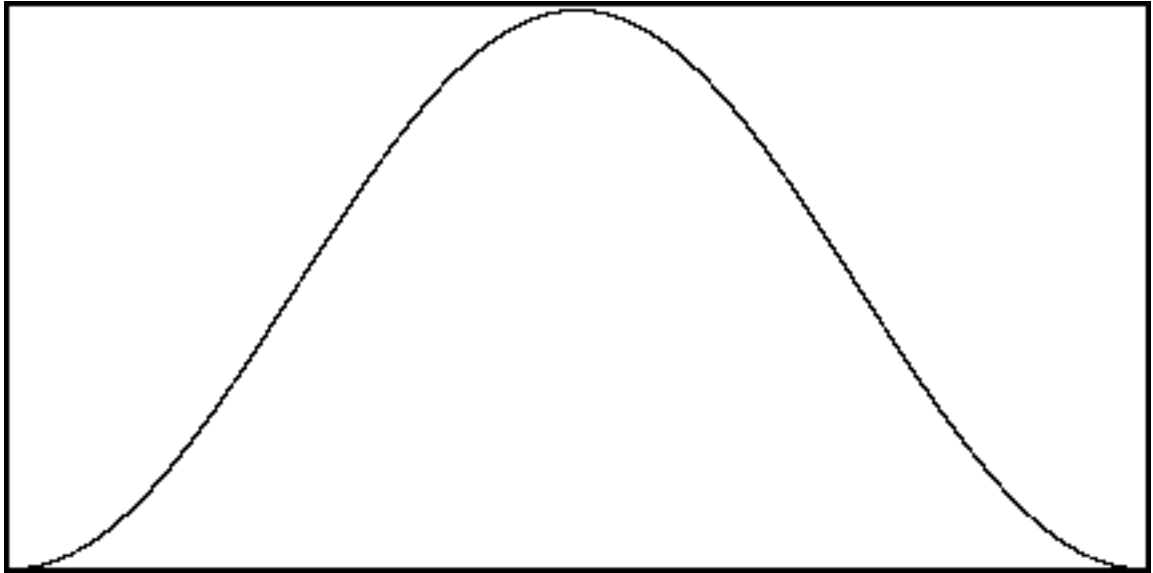


Fonction Fenêtre de Hamming.

## Hanning.

### Exemple F.2. Instruction pour la fonction fenêtre de Hanning

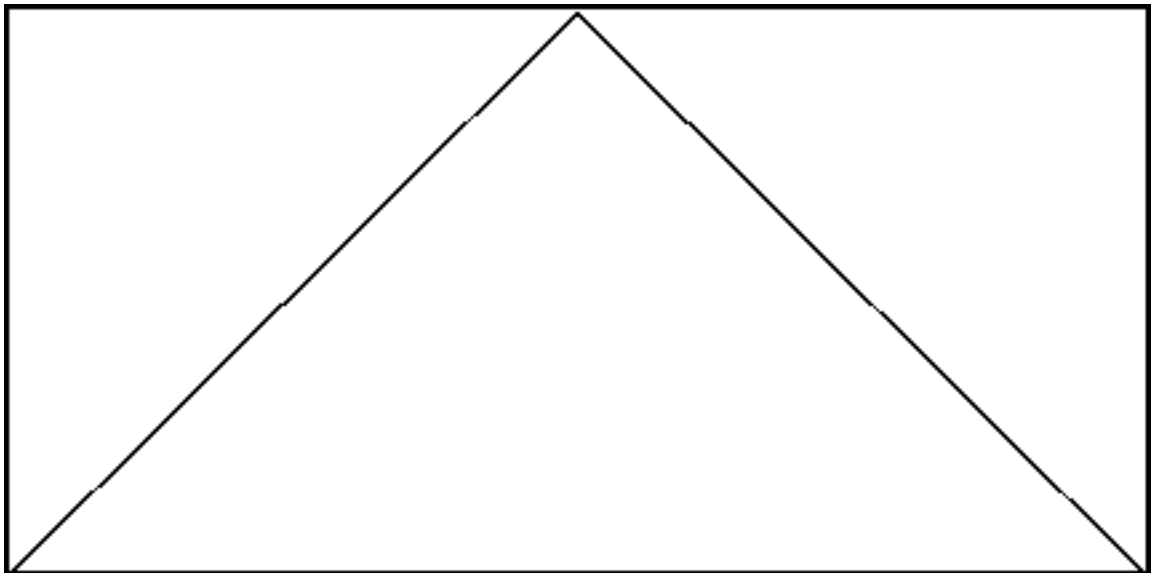
```
f82 0 8192 20 2 1
```



Fonction Fenêtre de Hanning

**Bartlett.****Exemple F.3. Instruction pour la fonction fenêtre de Bartlett**

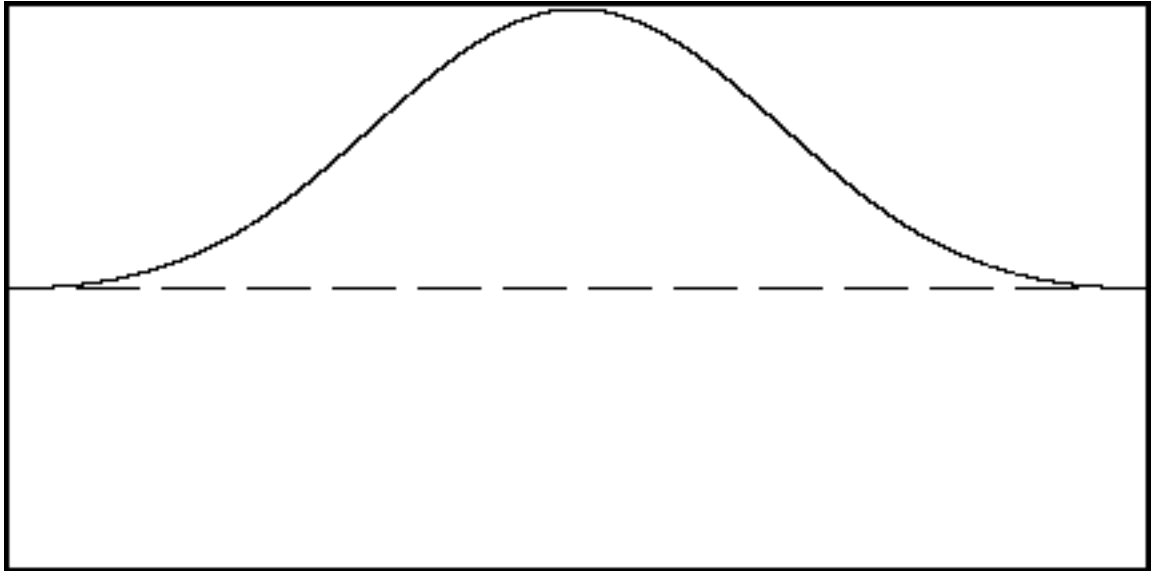
```
f83    0    8192    20    3    1
```



Fonction Fenêtre de Bartlett

**Blackman.****Exemple F.4. Instruction pour la fonction fenêtre de Blackman**

```
f84    0    8192    20    4    1
```

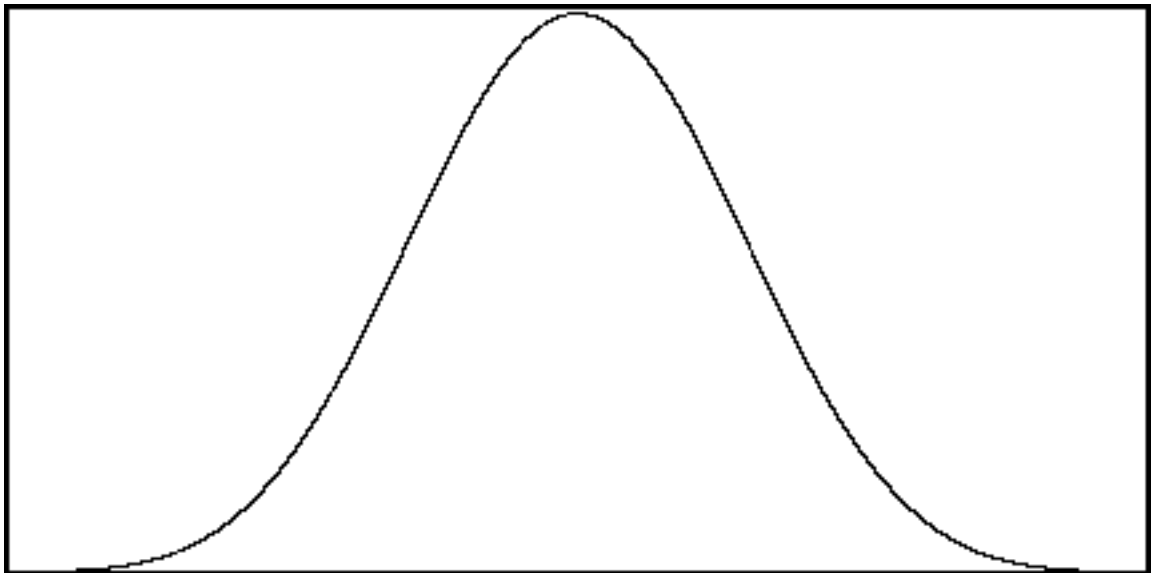


Fonction Fenêtre de Blackman

**Blackman-Harris.**

**Exemple F.5. Instruction pour la fonction fenêtre de Blackman-Harris**

f85 0 8192 20 5 1

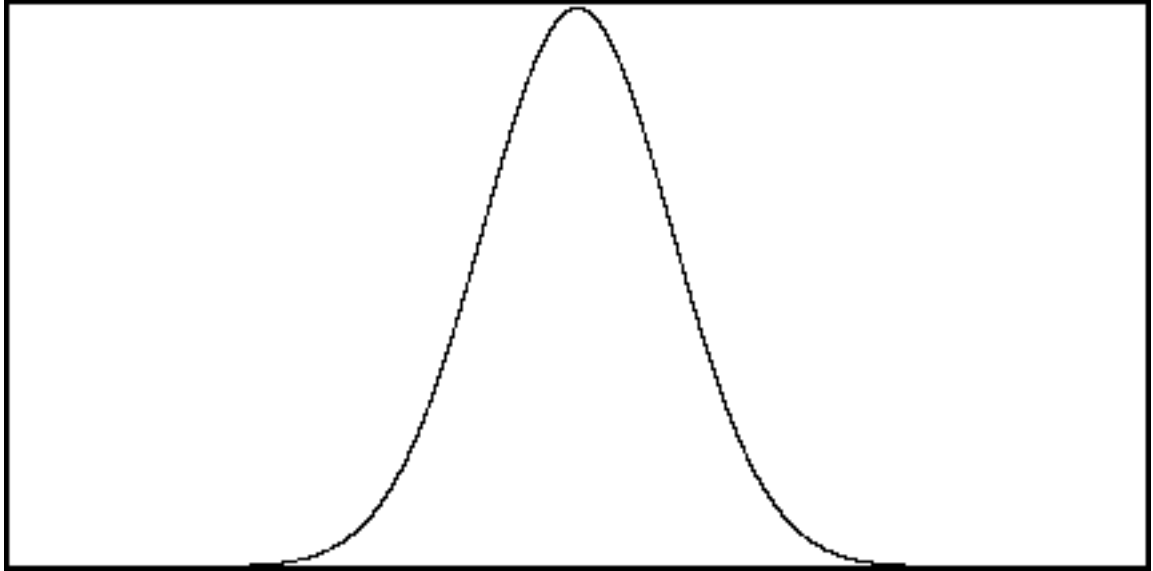


Fonction Fenêtre de Blackman-Harris

**Gaussienne.**

**Exemple F.6. Instruction pour la fonction fenêtre Gaussienne**

f86 0 8192 20 6 1

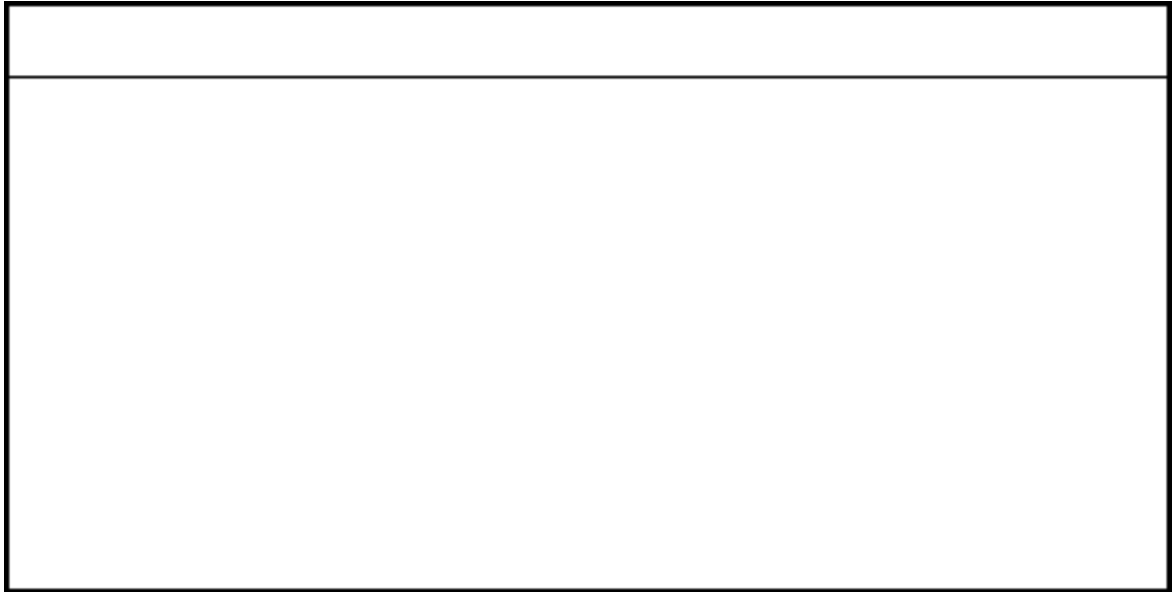


Fonction Fenêtre Gaussienne

**Rectangle.****Exemple F.7. Instruction pour la fonction fenêtre Rectangle**

```
f88  0  8192  -20  8  .1
```

*Note* : l'échelle verticale est exagérée dans ce diagramme.

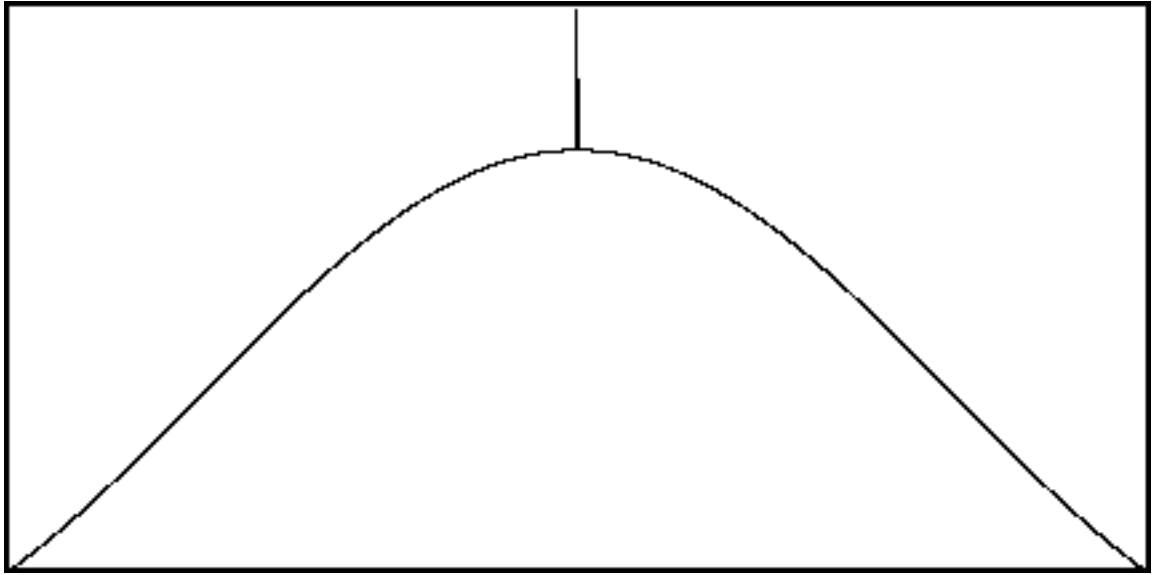


Fonction Fenêtre Rectangle

**Sync.****Exemple F.8. Instruction pour la fonction fenêtre Sync**

```
f89  0  4096  -20  9  .75
```





Fonction Fenêtre Sync

---

# Annexe G. Format de fichier SoundFont2

A partir de la version 4.07 de Csound, *Csound* supporte le format de fichier de sons échantillonnés *SoundFont2*. SoundFont2 (ou SF2) est un standard répandu qui permet l'encodage de banques de sons basés sur des tables d'onde dans un fichier binaire. Afin de comprendre l'usage de ces opcodes, l'utilisateur doit avoir une certaine connaissance du format SF2, c'est pourquoi une brève description de ce format suit.

Le format SF2 comprend des objets générateurs et modulateurs. Tous les opcodes actuels de Csound concernant SF2 ne supportent que la fonction générateur.

Il y a plusieurs niveaux de générateurs ayant une structure hiérarchique. Le type de générateur le plus élémentaire est le « sample » (son échantillonné). Les samples peuvent être bouclés ou non, et sont associés avec un numéro de note MIDI, appelé la touche de base. Quand un sample est associé à un intervalle de numéros de notes MIDI, un intervalle de vélocités, une transposition (accord grossier et fin), un accord d'échelle, un facteur de pondération de niveau, le sample et ses associations constituent un « split » (division). Un ensemble de splits, avec un nom, constituent un « instrument ». Quand un instrument est associé avec un intervalle de touches, un intervalle de vélocités, un facteur de pondération de niveau, et une transposition, l'instrument et ses associations constituent un « layer » (couche). Un ensemble de layers, avec un nom, constituent un « preset ». Les presets sont normalement les structures de génération sonore finales prêtes pour l'utilisateur. Ils génèrent le son selon les réglages de leurs composants des niveaux inférieurs.

Les données des sons échantillonnés et les données de structure sont incorporées dans le même fichier binaire SF2. Un fichier SF2 unique peut contenir au maximum 128 banques de 128 programmes de preset, soit un total de 16384 presets dans un fichier SF2. Le nombre maximum de layers, instruments, splits et samples n'est probablement limité que par la mémoire de l'ordinateur.

---

# Glossaire

## G

### Point de garde

Un point de garde est la dernière position d'une table de fonction. Si la longueur est, disons 1024, la table aura 1024+1 (1025) points : le point supplémentaire est le point de garde.

Dans tous les cas, pour une table de 1024 points, le premier point aura l'index 0 et le dernier l'index 1023 (l'index 1024 n'est pas réellement utilisé).

Il y a un point de garde car certains opcodes lisent les valeurs de la table par interpolation ; dans ce cas, si l'index de lecture est par exemple 1023,5, nous aurons besoin de la position 1024 pour l'interpolation.

Il y a deux manières de remplir ce point (écrire sa valeur) :

1. La manière par défaut : en copiant la valeur du 1er point de la table
2. Le point de garde étendu : en prolongeant le contour de la table (en continuant le calcul pour un point supplémentaire)

En général le premier mode est utilisé pour les applications cycliques, comme un oscillateur (qui lit la table en boucle continue). Le second usage est pour les lectures à passage unique, comme les enveloppes, où il faut interpoler le dernier point correctement en suivant le contour de la table (on ne boucle pas sur le début de la table).