# Csound for Portable Native Client

Victor Lazzarini

December 4, 2013

## 1 Introduction

Native Client (NaCl) is a sandboxing technology developed by Google that allows C/C++ modules to extend the support provided by HTML5. Portable Native Client (pNaCl) is one of the toolchains in the NaCl SDK (the others are newlib and glibc). The advantage of pNaCl over the other options is that it only requires a single module to be built for all supported architectures.

The other major advantage is that pNaCl is, as of Google Chrome 31, enabled by default in the browser. This means that users just need to load a page containing the pNaCl application and it will work. pNaCl modules are compiled to llvm bytecode that is translated to a native binary by the browser. To check whether your version of Chrome supports pNaCl, use the following address:

```
chrome://nacl
```

Porting Csound to pNaCl involved three steps, following the SDK installation:

1. Building libsndfile as a pNaCl library
2. Build Csound as a pNaCl library
3. Developing the pNaCl module to provide an interface to the Csound library

# 2 Building Csound for pNaCl

## 2.1 Building the libraries

With the NaCl SDK installed, and the NACL_SDK_ROOT set as per installation instructions and the libsndfile-nacl sources (`https://www.dropbox.com/s/ezfo9rmo5wtzptz/libsndfile-nacl.tar.gz`), you can use the make command to build libsndfile. To build the Csound library, run the build.sh script in the ./nacl subdirectory of the Csound 6 sources. When libraries are built, they are added to the SDK, and made readily available for applications to be built with them.

## 2.2 Building the pNaCl Csound module

Once the libraries are built, you can run make in the ./nacl/csound subdirectory of the Csound sources. This will build the nacl module in pnacl/Release. There is a package.sh that can be used to copy and package all the relevant files for HTML5 development. This package is self-contained, i.e. it does not have any dependencies, and it can be expanded elsewhere in your project application folders.

## 2.3 Running the example application

NaCl pages need to be served over http, which means they will not work when opened as local files. You need to start a local server, and this can be done with the python script httpd.py found in the $NACL_SDK_ROOT/tools directory. If you start this script in the top level directory of the pNaCl Csound package, then the example will be found at the `http://localhost:5103` address.

# 3 Csound pNaCl module reference

The interface to Csound is found in the csound.js javascript file. Csound is ready on module load, and can accept control messages from then on.

## 3.1 Control functions

The following control functions can be used to interact with Csound:

- `csound.Play()` - starts performance

- `csound.PlayCsd(s)` - starts performance from a CSD file s, which can be in ./http/ (ORIGIN server) or ./local/ (local sandbox).
- `csound.RenderCsd(s)` - renders a CSD file s, which can be in ./http/ (ORIGIN server) or ./local/ (local sandbox), with no RT audio output. The "finished render" message is issued on completion.
- `csound.Pause()` - pauses performance
- `csound.CompileOrc(s)` - compiles the Csound code in the string s
- `csound.ReadScore(s)` - reads the score in the string s (with preprocessing support)
- `csound.Event(s)` - sends in the line events contained in the string s (no preprocessing)
- `csound.SetChannel(name, value)` - sends the control channel *name* the value *value*, both arguments being strings.

### 3.2 Filesystem functions

In order to facilitate access to files, the following filesystem functions can be used:

- `csound.CopyToLocal(src, dest)` - copies the file *src* in the ORIGIN directory to the local file *dest*, which can be accessed at ./local/*dest*. The "Complete" message is issued on completion.
- `csound.CopyUrlToLocal(url,dest)` - copies the url *url* to the local file *dest*, which can be accessed at ./local/*dest*. Currently only ORIGIN and CORS urls are allowed remotely, but local files can also be passed if encoded as urls with the webkitURL.createObjectURL() javascript method. The "Complete" message is issued on completion.
- `csound.RequestFileFromLocal(src)` - requests the data from the local file *src*. The "Complete" message is issued on completion.
- `csound.GetFileData()` - returns the most recently requested file data as an ArrayObject.

### 3.3 Callbacks

The csound.js module will call the following window functions when it starts:

- `function moduleDidLoad()`: this is called as soon as the module is loaded
- `function handleMessage(message)`: called when there are messages from Csound (pnacl module). The string message.data contains the message.
- `function attachListeners()`: this is called when listeners for different events are to be attached.

You should implement these functions in your HTML page script, in order to use the Csound javascript interface. In addition to the above, Csound javascript module messages are always sent to the HTML element with id='console', which is normally of type  or <textarea>.

### 3.4 Example

Here is a minimal HTML example showing the use of Csound

```
   ,
<!DOCTYPE html>
<html>
<!--
 Csound pnacl minimal example
 Copyright (C) 2013 V Lazzarini
-->
<head>
 <title>Minimal Csound Example</title>
 <script type="text/javascript" src="csound.js"></script>
 <script type="text/javascript">
 // called by csound.js
 function moduleDidLoad() {
  csound.Play();
  csound.CompileOrc(
  "instr 1 \n" +
  "icps = 440+rnd(440) \n" +
  "chnset icps, \"freq\" \n" +
  "a1 oscili 0.1, icps\n" +
  "outs a1,a1 \n" +
  "endin");
  document.getElementById("tit").innerHTML = "Click on the page below to play";
 }
 function attachListeners() {
```

```
   document.getElementById("mess").
       addEventListener("click",Play);
 }
 function handleMessage(message) {
   var mess = message.data;
   if(mess.slice(0,11) == "::control::") {
   var messField = document.getElementById("console")
   messField.innerText = mess.slice(11);
   }
   else {
   var messField = document.getElementById("mess")
   messField.innerText += mess;
   csound.RequestChannel("freq");
  }
 }

 // click handler
 function Play() {
   csound.Event("i 1 0 5");
 }
 </script>
</head>
<body>
  <div id="console"></div>
   <h3 id="tit"> </h3>
  <div id="mess">

  </div>
  <!--pNaCl csound module-->
  <div id="engine"></div>
</body>
</html>
```

# 4 Limitations

The following limitations apply:

- no realtime audio input (not supported yet in Pepper/NaCl)
- no MIDI in the NaCl module. However, it might be possible to implement MIDI in javascript, and using the csound.js functions, send data to Csound, and respond to MIDI NOTE messages.
- no plugins, as pNaCl does not support dlopen() and friends. This means some opcodes are not available as they reside in plugin libraries.

It might be possible to add some of these opcodes statically to the Csound pNaCl library in the future.