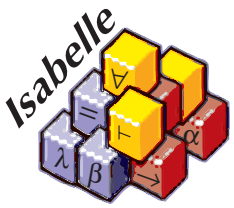

The Isabelle System Manual

Makarius Wenzel and *Stefan Berghofer*
TU München

22 May 2012

Contents

Chapter 1

The Isabelle system environment

This manual describes Isabelle together with related tools and user interfaces as seen from a system oriented view. See also the *Isabelle/Isar Reference Manual* [4] for the actual Isabelle input language and related concepts, and *The Isabelle/Isar Implementation Manual* [3] for the main concepts of the underlying implementation in Isabelle/ML.

The Isabelle system environment provides the following basic infrastructure to integrate tools smoothly.

1. The *Isabelle settings* mechanism provides process environment variables to all Isabelle executables (including tools and user interfaces).
2. The raw *Isabelle process* (`isabelle-process`) runs logic sessions either interactively or in batch mode. In particular, this view abstracts over the invocation of the actual ML system to be used. Regular users rarely need to care about the low-level process.
3. The main *Isabelle tools wrapper* (`isabelle`) provides a generic startup environment Isabelle related utilities, user interfaces etc. Such tools automatically benefit from the settings mechanism.

1.1 Isabelle settings

The Isabelle system heavily depends on the *settings mechanism*. Essentially, this is a statically scoped collection of environment variables, such as `ISABELLE_HOME`, `ML_SYSTEM`, `ML_HOME`. These variables are *not* intended to be set directly from the shell, though. Isabelle employs a somewhat more sophisticated scheme of *settings files* — one for site-wide defaults, another for additional user-specific modifications. With all configuration variables

in clearly defined places, this scheme is more maintainable and user-friendly than global shell environment variables.

In particular, we avoid the typical situation where prospective users of a software package are told to put several things into their shell startup scripts, before being able to actually run the program. Isabelle requires none such administrative chores of its end-users — the executables can be invoked straight away. Occasionally, users would still want to put the `$ISABELLE_HOME/bin` directory into their shell's search path, but this is not required.

1.1.1 Bootstrapping the environment

Isabelle executables need to be run within a proper settings environment. This is bootstrapped as described below, on the first invocation of one of the outer wrapper scripts (such as `isabelle`). This happens only once for each process tree, i.e. the environment is passed to subprocesses according to regular Unix conventions.

1. The special variable `ISABELLE_HOME` is determined automatically from the location of the binary that has been run.

   You should not try to set `ISABELLE_HOME` manually. Also note that the Isabelle executables either have to be run from their original location in the distribution directory, or via the executable objects created by the `install` utility. Symbolic links are admissible, but a plain copy of the `$ISABELLE_HOME/bin` files will not work!

2. The file `$ISABELLE_HOME/etc/settings` is run as a `bash` shell script with the auto-export option for variables enabled.

   This file holds a rather long list of shell variable assigments, thus providing the site-wide default settings. The Isabelle distribution already contains a global settings file with sensible defaults for most variables. When installing the system, only a few of these may have to be adapted (probably `ML_SYSTEM` etc.).

3. The file `$ISABELLE_HOME_USER/etc/settings` (if it exists) is run in the same way as the site default settings. Note that the variable `ISABELLE_HOME_USER` has already been set before — usually to something like `$USER_HOME/.isabelle/IsabelleXXXX`.

   Thus individual users may override the site-wide defaults. See also file `$ISABELLE_HOME/etc/user-settings.sample` in the distribution. Typically, a user settings file would contain only a few lines, just the

assigments that are really changed. One should definitely *not* start with a full copy the basic `$ISABELLE_HOME/etc/settings`. This could cause very annoying maintainance problems later, when the Isabelle installation is updated or changed otherwise.

Since settings files are regular GNU `bash` scripts, one may use complex shell commands, such as `if` or `case` statements to set variables depending on the system architecture or other environment variables. Such advanced features should be added only with great care, though. In particular, external environment references should be kept at a minimum.

A few variables are somewhat special:

- `ISABELLE_PROCESS` and `ISABELLE_TOOL` are set automatically to the absolute path names of the `isabelle-process` and `isabelle` executables, respectively.
- `ISABELLE_OUTPUT` will have the identifiers of the Isabelle distribution (cf. `ISABELLE_IDENTIFIER`) and the ML system (cf. `ML_IDENTIFIER`) appended automatically to its value.

Note that the settings environment may be inspected with the Isabelle tool `getenv`. This might help to figure out the effect of complex settings scripts.

1.1.2 Common variables

This is a reference of common Isabelle settings variables. Note that the list is somewhat open-ended. Third-party utilities or interfaces may add their own selection. Variables that are special in some sense are marked with *.

`USER_HOME`* Is the cross-platform user home directory. On Unix systems this is usually the same as `HOME`, but on Windows it is the regular home directory of the user, not the one of within the Cygwin root file-system.[1]

`ISABELLE_HOME`* is the location of the top-level Isabelle distribution directory. This is automatically determined from the Isabelle executable that has been invoked. Do not attempt to set `ISABELLE_HOME` yourself from the shell!

[1]Cygwin itself offers another choice whether its HOME should point to the `/home` directory tree or the Windows user home.

ISABELLE_HOME_USER is the user-specific counterpart of ISABELLE_HOME. The default value is relative to $USER_HOME/.isabelle, under rare circumstances this may be changed in the global setting file. Typically, the ISABELLE_HOME_USER directory mimics ISABELLE_HOME to some extend. In particular, site-wide defaults may be overridden by a private $ISABELLE_HOME_USER/etc/settings.

ISABELLE_PLATFORM* is automatically set to a symbolic identifier for the underlying hardware and operating system. The Isabelle platform identification always refers to the 32 bit variant, even this is a 64 bit machine. Note that the ML or Java runtime may have a different idea, depending on which binaries are actually run.

ISABELLE_PLATFORM64* is similar to ISABELLE_PLATFORM but refers to the proper 64 bit variant on a platform that supports this; the value is empty for 32 bit. Note that the following bash expression (including the quotes) prefers the 64 bit platform, if that is available:

```
"${ISABELLE_PLATFORM64:-$ISABELLE_PLATFORM}"
```

ISABELLE_PROCESS*, ISABELLE_TOOL* are automatically set to the full path names of the isabelle-process and isabelle executables, respectively. Thus other tools and scripts need not assume that the $ISABELLE_HOME/bin directory is on the current search path of the shell.

ISABELLE_IDENTIFIER* refers to the name of this Isabelle distribution, e.g. "Isabelle2012".

ML_SYSTEM, ML_HOME, ML_OPTIONS, ML_PLATFORM, ML_IDENTIFIER* specify the underlying ML system to be used for Isabelle. There is only a fixed set of admissable ML_SYSTEM names (see the $ISABELLE_HOME/etc/settings file of the distribution).

The actual compiler binary will be run from the directory ML_HOME, with ML_OPTIONS as first arguments on the command line. The optional ML_PLATFORM may specify the binary format of ML heap images, which is useful for cross-platform installations. The value of ML_IDENTIFIER is automatically obtained by composing the values of ML_SYSTEM, ML_PLATFORM and the Isabelle version values.

ISABELLE_JDK_HOME needs to point to a full JDK (Java Development Kit) installation with javac and jar executables. This is essential for Isabelle/Scala and other JVM-based tools to work properly. Note that

conventional `JAVA_HOME` usually points to the JRE (Java Runtime Environment), not JDK.

`ISABELLE_PATH` is a list of directories (separated by colons) where Isabelle logic images may reside. When looking up heaps files, the value of `ML_IDENTIFIER` is appended to each component internally.

`ISABELLE_OUTPUT`* is a directory where output heap files should be stored by default. The ML system and Isabelle version identifier is appended here, too.

`ISABELLE_BROWSER_INFO` is the directory where theory browser information (HTML text, graph data, and printable documents) is stored (see also §3.1). The default value is `$ISABELLE_HOME_USER/browser_info`.

`ISABELLE_LOGIC` specifies the default logic to load if none is given explicitely by the user. The default value is `HOL`.

`ISABELLE_LINE_EDITOR` specifies the default line editor for the `tty` interface.

`ISABELLE_USEDIR_OPTIONS` is implicitly prefixed to the command line of any `usedir` invocation. This typically contains compilation options for object-logics — `usedir` is the basic utility for managing logic sessions (cf. the `IsaMakefile`s in the distribution).

`ISABELLE_LATEX`, `ISABELLE_PDFLATEX`, `ISABELLE_BIBTEX`, `ISABELLE_DVIPS` refer to LaTeX related tools for Isabelle document preparation (see also §3.6).

`ISABELLE_TOOLS` is a colon separated list of directories that are scanned by `isabelle` for external utility programs (see also §1.3).

`ISABELLE_DOCS` is a colon separated list of directories with documentation files.

`ISABELLE_DOC_FORMAT` specifies the preferred document format, typically `dvi` or `pdf`.

`DVI_VIEWER` specifies the command to be used for displaying `dvi` files.

`PDF_VIEWER` specifies the command to be used for displaying `pdf` files.

`PRINT_COMMAND` specifies the standard printer spool command, which is expected to accept `ps` files.

`ISABELLE_TMP_PREFIX`* is the prefix from which any running `isabelle-process` derives an individual directory for temporary files. The default is somewhere in `/tmp`.

1.1.3 Additional components

Any directory may be registered as an explicit *Isabelle component*. The general layout conventions are that of the main Isabelle distribution itself, and the following two files (both optional) have a special meaning:

- `etc/settings` holds additional settings that are initialized when bootstrapping the overall Isabelle environment, cf. §1.1.1. As usual, the content is interpreted as a `bash` script. It may refer to the component's enclosing directory via the `COMPONENT` shell variable.

  For example, the following setting allows to refer to files within the component later on, without having to hardwire absolute paths:

  ```
  MY_COMPONENT_HOME="$COMPONENT"
  ```

  Components can also add to existing Isabelle settings such as `ISABELLE_TOOLS`, in order to provide component-specific tools that can be invoked by end-users. For example:

  ```
  ISABELLE_TOOLS="$ISABELLE_TOOLS:$COMPONENT/lib/Tools"
  ```

- `etc/components` holds a list of further sub-components of the same structure. The directory specifications given here can be either absolute (with leading `/`) or relative to the component's main directory.

The root of component initialization is `ISABELLE_HOME` itself. After initializing all of its sub-components recursively, `ISABELLE_HOME_USER` is included in the same manner (if that directory exists). This allows to install private components via `$ISABELLE_HOME_USER/etc/components`, although it is often more convenient to do that programmatically via the `init_component` shell function in the `etc/settings` script of `$ISABELLE_HOME_USER` (or any other component directory). For example:

```
if [ -d "$HOME/screwdriver-2.0" ]
then
  init_component "$HOME/screwdriver-2.0"
else
```

1.2 The raw Isabelle process

The `isabelle-process` executable runs bare-bones Isabelle logic sessions — either interactively or in batch mode. It provides an abstraction over the underlying ML system, and over the actual heap file locations. Its usage is:

```
Usage: isabelle-process [OPTIONS] [INPUT] [OUTPUT]

  Options are:
    -I           startup Isar interaction mode
    -P           startup Proof General interaction mode
    -S           secure mode -- disallow critical operations
    -T ADDR      startup process wrapper, with socket address
    -W IN:OUT    startup process wrapper, with input/output fifos
    -X           startup PGIP interaction mode
    -e MLTEXT    pass MLTEXT to the ML session
    -f           pass 'Session.finish();' to the ML session
    -m MODE      add print mode for output
    -q           non-interactive session
    -r           open heap file read-only
    -u           pass 'use"ROOT.ML";' to the ML session
    -w           reset write permissions on OUTPUT

  INPUT (default "$ISABELLE_LOGIC") and OUTPUT specify in/out heaps.
  These are either names to be searched in the Isabelle path, or
  actual file names (containing at least one /).
  If INPUT is "RAW_ML_SYSTEM", just start the bare bones ML system.
```

Input files without path specifications are looked up in the `ISABELLE_PATH` setting, which may consist of multiple components separated by colons — these are tried in the given order with the value of `ML_IDENTIFIER` appended internally. In a similar way, base names are relative to the directory specified by `ISABELLE_OUTPUT`. In any case, actual file locations may also be given by including at least one slash (`/`) in the name (hint: use `./` to refer to the current directory).

Options

If the input heap file does not have write permission bits set, or the `-r` option is given explicitely, then the session started will be read-only. That is, the ML world cannot be committed back into the image file. Otherwise, a writable session enables commits into either the input file, or into another output heap file (if that is given as the second argument on the command line).

The read-write state of sessions is determined at startup only, it cannot be changed intermediately. Also note that heap images may require considerable amounts of disk space (approximately 50–200 MB). Users are responsible for themselves to dispose their heap files when they are no longer needed.

The `-w` option makes the output heap file read-only after terminating. Thus subsequent invocations cause the logic image to be read-only automatically.

Using the `-e` option, arbitrary ML code may be passed to the Isabelle session from the command line. Multiple `-e`'s are evaluated in the given order. Strange things may happen when errorneous ML code is provided. Also make sure that the ML commands are terminated properly by semicolon.

The `-u` option is a shortcut for `-e` passing "`use "ROOT.ML";`" to the ML session. The `-f` option passes "`Session.finish();`", which is intended mainly for administrative purposes.

The `-m` option adds identifiers of print modes to be made active for this session. Typically, this is used by some user interface, e.g. to enable output of proper mathematical symbols.

Isabelle normally enters an interactive top-level loop (after processing the `-e` texts). The `-q` option inhibits interaction, thus providing a pure batch mode facility.

The `-I` option makes Isabelle enter Isar interaction mode on startup, instead of the primitive ML top-level. The `-P` option configures the top-level loop for interaction with the Proof General user interface, and the `-X` option enables XML-based PGIP communication.

The `-T` or `-W` option makes Isabelle enter a special process wrapper for interaction via the Isabelle/Scala layer, see also `~~/src/Pure/System/isabelle_process.scala`. The protocol between the ML and JVM process is private to the implementation.

The `-S` option makes the Isabelle process more secure by disabling some critical operations, notably runtime compilation and evaluation of ML source code.

Examples

Run an interactive session of the default object-logic (as specified by the `ISABELLE_LOGIC` setting) like this:

```
isabelle-process
```

Usually `ISABELLE_LOGIC` refers to one of the standard logic images, which are read-only by default. A writable session — based on `HOL`, but output to `Test` (in the directory specified by the `ISABELLE_OUTPUT` setting) — may be invoked as follows:

```
isabelle-process HOL Test
```

Ending this session normally (e.g. by typing control-D) dumps the whole ML system state into `Test` (be prepared for more than 100 MB):
The `Test` session may be continued later (still in writable state) by:

```
isabelle-process Test
```

A read-only `Test` session may be started by:

```
isabelle-process -r Test
```

Note that manual session management like this does *not* provide proper setup for theory presentation. This would require the `usedir` utility.

The next example demonstrates batch execution of Isabelle. We retrieve the `Main` theory value from the theory loader within ML (observe the delicate quoting rules for the Bash shell vs. ML):

```
isabelle-process -e 'Thy_Info.get_theory "Main";' -q -r HOL
```

Note that the output text will be interspersed with additional junk messages by the ML runtime environment. The `-W` option allows to communicate with the Isabelle process via an external program in a more robust fashion.

1.3 The Isabelle tools wrapper

All Isabelle related tools and interfaces are called via a common wrapper — `isabelle`:

```
Usage: isabelle TOOL [ARGS ...]

  Start Isabelle tool NAME with ARGS; pass "-?" for tool specific help.

  Available tools are:

    browser - Isabelle graph browser
    ...
```

In principle, Isabelle tools are ordinary executable scripts that are run within the Isabelle settings environment, see §1.1. The set of available tools is collected by `isabelle` from the directories listed in the `ISABELLE_TOOLS` setting. Do not try to call the scripts directly from the shell. Neither should you add the tool directories to your shell's search path!

Examples

Show the list of available documentation of the current Isabelle installation like this:

```
isabelle doc
```

View a certain document as follows:

```
isabelle doc system
```

Create an Isabelle session derived from HOL (see also §3.3 and §5.8):

```
isabelle mkdir HOL Test && isabelle make
```

Note that `isabelle mkdir` is usually only invoked once; existing sessions (including document output etc.) are then updated by `isabelle make` alone.

Chapter 2

User interfaces

2.1 Plain TTY interaction

The `tty` tool runs the Isabelle process interactively within a plain terminal session:

```
Usage: tty [OPTIONS]

  Options are:
    -l NAME      logic image name (default ISABELLE_LOGIC)
    -m MODE      add print mode for output
    -p NAME      line editor program name (default ISABELLE_LINE_EDITOR)

  Run Isabelle process with plain tty interaction, and optional line editor.
```

The `-l` option specifies the logic image. The `-m` option specifies additional print modes. The `-p` option specifies an alternative line editor (such as the `rlwrap` wrapper for GNU readline); the fall-back is to use raw standard input.

Regular interaction is via the standard Isabelle/Isar toplevel loop. The Isar command **exit** drops out into the raw ML system, which is occasionally useful for low-level debugging. Invoking `Isar.loop ();` in ML will return to the Isar toplevel.

2.2 Proof General / Emacs

The `emacs` tool invokes a version of Emacs and Proof General within the regular Isabelle settings environment (§1.1). This is more robust than starting Emacs separately, loading the Proof General lisp files, and then attempting to start Isabelle with dynamic `PATH` lookup etc.

The actual interface script is part of the Proof General distribution [1]; its usage depends on the particular version. There are some options available, such as `-l` for passing the logic image to be used by default, or `-m` to tune the standard print mode. The following Isabelle settings are particularly important for Proof General:

`PROOFGENERAL_HOME` points to the main installation directory of the Proof General distribution. The default settings try to locate this in a number of standard places, notably `$ISABELLE_HOME/contrib/ProofGeneral`.

`PROOFGENERAL_OPTIONS` is implicitly prefixed to the command line of any invocation of the Proof General `interface` script.

`XSYMBOL_INSTALLFONTS` may contain a small shell script to install the X11 fonts required for the old X-Symbols mode of Proof General. This is only relevant if the X11 display server runs on a different machine than the Emacs application, with a different file-system view on the Proof General installation. Under most circumstances Proof General 3.x is able to refer to the font files that are part of its distribution, and Proof General 4.x finds its fonts by different means.

2.3 Isabelle/jEdit Prover IDE

The `jedit` tool invokes a version of jEdit that has been augmented with some components to provide a fully-featured Prover IDE (based on Isabelle/Scala):

```
Usage: isabelle jedit [OPTIONS] [FILES ...]

  Options are:
    -J OPTION    add JVM runtime option (default JEDIT_JAVA_OPTIONS)
    -b           build only
    -d           enable debugger
    -f           fresh build
    -j OPTION    add jEdit runtime option (default JEDIT_OPTIONS)
    -l NAME      logic image name (default ISABELLE_LOGIC)
    -m MODE      add print mode for output

  Start jEdit with Isabelle plugin setup and opens theory FILES
  (default Scratch.thy).
```

The `-l` option specifies the logic image. The `-m` option specifies additional print modes.

The `-J` and `-j` options allow to pass additional low-level options to the JVM or jEdit, respectively. The defaults are provided by the Isabelle settings environment.

The `-d` option allows to connect to the runtime debugger of the JVM. Note that the Scala Console of Isabelle/jEdit is more convenient in most practical situations.

The `-b` and `-f` options control the self-build mechanism of Isabelle/jEdit. This is only relevant for building from sources, which also requires an auxiliary `jedit_build` component. Official Isabelle releases already include a version of Isabelle/jEdit that is built properly.

Chapter 3

Presenting theories

Isabelle provides several ways to present the outcome of formal developments, including WWW-based browsable libraries or actual printable documents. Presentation is centered around the concept of *logic sessions*. The global session structure is that of a tree, with Isabelle Pure at its root, further object-logics derived (e.g. HOLCF from HOL, and HOL from Pure), and application sessions in leaf positions (usually without a separate image).

The Isabelle tools `mkdir` and `make` provide the primary means for managing Isabelle sessions, including proper setup for presentation. Here the `usedir` tool takes care to let `isabelle-process` process run any additional stages required for document preparation, notably the tools `document` and `latex`. The complete tool chain for managing batch-mode Isabelle sessions is illustrated in figure 3.1.

`isabelle mkdir`	invoked once by the user to create the initial source setup (common `IsaMakefile` plus a single session directory);
`isabelle make`	invoked repeatedly by the user to keep session output up-to-date (HTML, documents etc.);
`isabelle usedir`	part of the standard `IsaMakefile` entry of a session;
`isabelle-process`	run through `isabelle usedir`;
`isabelle document`	run by the Isabelle process if document preparation is enabled;
`isabelle latex`	universal LaTeX tool wrapper invoked multiple times by `isabelle document`; also useful for manual experiments;

Figure 3.1: The tool chain of Isabelle session presentation

3.1 Generating theory browser information

As a side-effect of running a logic sessions, Isabelle is able to generate theory browsing information, including HTML documents that show a theory's definition, the theorems proved in its ML file and the relationship with its ancestors and descendants. Besides the HTML file that is generated for every theory, Isabelle stores links to all theories in an index file. These indexes are linked with other indexes to represent the overall tree structure of logic sessions.

Isabelle also generates graph files that represent the theory hierarchy of a logic. There is a graph browser Java applet embedded in the generated HTML pages, and also a stand-alone application that allows browsing theory graphs without having to start a WWW client first. The latter version also includes features such as generating Postscript files, which are not available in the applet version. See §3.2 for further information.

The easiest way to let Isabelle generate theory browsing information for existing sessions is to append "`-i true`" to the `ISABELLE_USEDIR_OPTIONS` before invoking `isabelle make` (or `$ISABELLE_HOME/build`). For example, add something like this to your Isabelle settings file

```
ISABELLE_USEDIR_OPTIONS="-i true"
```

and then change into the `~~/src/FOL` directory and run `isabelle make`, or even `isabelle make all`. The presentation output will appear in `ISABELLE_BROWSER_INFO/FOL`, which usually refers to something like `~/.isabelle/IsabelleXXXX/browser_info/FOL`. Note that option `-v true` will make the internal runs of `usedir` more explicit about such details.

Many standard Isabelle sessions (such as `~~/src/HOL/ex`) also provide actual printable documents. These are prepared automatically as well if enabled like this, using the `-d` option

```
ISABELLE_USEDIR_OPTIONS="-i true -d dvi"
```

Enabling options `-i` and `-d` simultaneously as shown above causes an appropriate "document" link to be included in the HTML index. Documents (or raw document sources) may be generated independently of browser information as well, see §3.5 for further details.

The theory browsing information is stored in a sub-directory directory determined by the `ISABELLE_BROWSER_INFO` setting plus a prefix corresponding to the session identifier (according to the tree structure of sub-sessions by

default). A complete WWW view of all standard object-logics and examples of the Isabelle distribution is available at the usual Isabelle sites:

http://isabelle.in.tum.de/dist/library/
http://www.cl.cam.ac.uk/research/hvg/Isabelle/dist/library/
http://mirror.cse.unsw.edu.au/pub/isabelle/dist/library/

In order to present your own theories on the web, simply copy the corresponding subdirectory from `ISABELLE_BROWSER_INFO` to your WWW server, having generated browser info like this:

```
isabelle usedir -i true HOL Foo
```

This assumes that directory `Foo` contains some `ROOT.ML` file to load all your theories, and HOL is your parent logic image (`isabelle mkdir` assists in setting up Isabelle session directories. Theory browser information for HOL should have been generated already beforehand. Alternatively, one may specify an external link to an existing body of HTML data by giving `usedir` a `-P` option like this:

```
isabelle usedir -i true -P http://isabelle.in.tum.de/library/ HOL Foo
```

For production use, the `usedir` tool is usually invoked in an appropriate `IsaMakefile`, via the Isabelle `make` tool. There is a separate `mkdir` tool to provide easy setup of all this, with only minimal manual editing required.

```
isabelle mkdir HOL Foo && isabelle make
```

See §3.3 for more information on preparing Isabelle session directories, including the setup for documents.

3.2 Browsing theory graphs

The Isabelle graph browser is a general tool for visualizing dependency graphs. Certain nodes of the graph (i.e. theories) can be grouped together in "directories", whose contents may be hidden, thus enabling the user to collapse irrelevant portions of information. The browser is written in Java, it can be used both as a stand-alone application and as an applet. Note that the option `-g` of `isabelle usedir` creates graph presentations in batch mode for inclusion in session documents.

3.2.1 Invoking the graph browser

The stand-alone version of the graph browser is wrapped up as an Isabelle tool called `browser`:

```
Usage: browser [OPTIONS] [GRAPHFILE]

  Options are:
    -b           Admin/build only
    -c           cleanup -- remove GRAPHFILE after use
    -o FILE      output to FILE (ps, eps, pdf)
```

When no filename is specified, the browser automatically changes to the directory `ISABELLE_BROWSER_INFO`.

The `-b` option indicates that this is for administrative build only, i.e. no browser popup if no files are given.

The `-c` option causes the input file to be removed after use.

The `-o` option indicates batch-mode operation, with the output written to the indicated file; note that `pdf` produces an `eps` copy as well.

The applet version of the browser is part of the standard WWW theory presentation, see the link "theory dependencies" within each session index.

3.2.2 Using the graph browser

The browser's main window, which is shown in figure 3.2, consists of two sub-windows. In the left sub-window, the directory tree is displayed. The graph itself is displayed in the right sub-window.

The directory tree window

We describe the usage of the directory browser and the meaning of the different items in the browser window.

- A red arrow before a directory name indicates that the directory is currently "folded", i.e. the nodes in this directory are collapsed to one single node. In the right sub-window, the names of nodes corresponding to folded directories are enclosed in square brackets and displayed in red color.

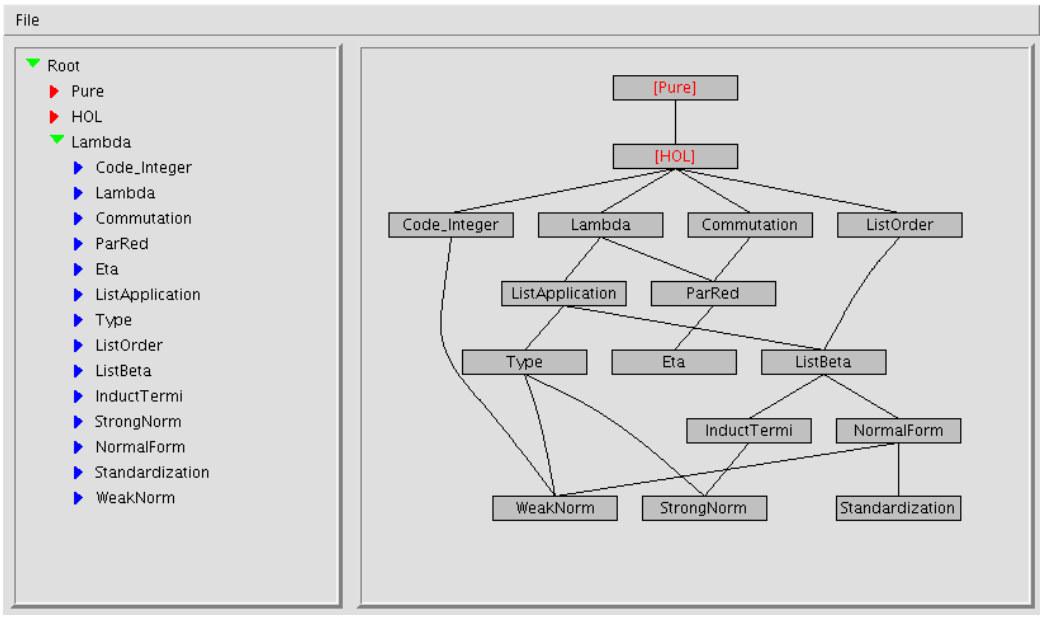

Figure 3.2: Browser main window

- A green downward arrow before a directory name indicates that the directory is currently "unfolded". It can be folded by clicking on the directory name. Clicking on the name for a second time unfolds the directory again. Alternatively, a directory can also be unfolded by clicking on the corresponding node in the right sub-window.
- Blue arrows stand before ordinary node names. When clicking on such a name (i.e. that of a theory), the graph display window focuses to the corresponding node. Double clicking invokes a text viewer window in which the contents of the theory file are displayed.

The graph display window

When pointing on an ordinary node, an upward and a downward arrow is shown. Initially, both of these arrows are green. Clicking on the upward or downward arrow collapses all predecessor or successor nodes, respectively. The arrow's color then changes to red, indicating that the predecessor or successor nodes are currently collapsed. The node corresponding to the collapsed nodes has the name "`[....]`". To uncollapse the nodes again, simply click on the red arrow or on the node with the name "`[....]`". Similar to the directory browser, the contents of theory files can be displayed by double clicking on the corresponding node.

The "File" menu

Due to Java Applet security restrictions this menu is only available in the full application version. The meaning of the menu items is as follows:

Open ... Open a new graph file.

Export to PostScript Outputs the current graph in Postscript format, appropriately scaled to fit on one single sheet of A4 paper. The resulting file can be printed directly.

Export to EPS Outputs the current graph in Encapsulated Postscript format. The resulting file can be included in other documents.

Quit Quit the graph browser.

3.2.3 Syntax of graph definition files

A graph definition file has the following syntax:

graph = { *vertex* ; }+
vertex = *vertex-name vertex-ID dir-name* [+] *path* [< | >] { *vertex-ID* }∗

The meaning of the items in a vertex description is as follows:

vertex-name The name of the vertex.

vertex-ID The vertex identifier. Note that there may be several vertices with equal names, whereas identifiers must be unique.

dir-name The name of the "directory" the vertex should be placed in. A "+" sign after *dir-name* indicates that the nodes in the directory are initially visible. Directories are initially invisible by default.

path The path of the corresponding theory file. This is specified relatively to the path of the graph definition file.

List of successor/predecessor nodes A "<" sign before the list means that successor nodes are listed, a ">" sign means that predecessor nodes are listed. If neither "<" nor ">" is found, the browser assumes that successor nodes are listed.

3.3 Creating Isabelle session directories

The `mkdir` utility prepares Isabelle session source directories, including a sensible default setup of `IsaMakefile`, `ROOT.ML`, and a `document` directory with a minimal `root.tex` that is sufficient to print all theories of the session (in the order of appearance); see §3.5 for further information on Isabelle document preparation. The usage of `isabelle mkdir` is:

```
Usage: mkdir [OPTIONS] [LOGIC] NAME

  Options are:
    -I FILE      alternative IsaMakefile output
    -P           include parent logic target
    -b           setup build mode (session outputs heap image)
    -q           quiet mode

  Prepare session directory, including IsaMakefile and document source,
  with parent LOGIC (default ISABELLE_LOGIC=$ISABELLE_LOGIC)
```

The `mkdir` tool is conservative in the sense that any existing `IsaMakefile` etc. is left unchanged. Thus it is safe to invoke it multiple times, although later runs may not have the desired effect.

Note that `mkdir` is unable to change `IsaMakefile` incrementally — manual changes are required for multiple sub-sessions. On order to get an initial working session, the only editing needed is to add appropriate `use_thy` calls to the generated `ROOT.ML` file.

Options

The `-I` option specifies an alternative to `IsaMakefile` for dependencies. Note that "`-`" refers to *stdout*, i.e. "`-I-`" provides an easy way to peek at `mkdir`'s idea of `make` setup required for some particular of Isabelle session.

The `-P` option includes a target for the parent `LOGIC` session in the generated `IsaMakefile`. The corresponding sources are assumed to be located within the Isabelle distribution.

The `-b` option sets up the current directory as the base for a new session that provides an actual logic image, as opposed to one that only runs several theories based on an existing image. Note that in the latter case, everything except `IsaMakefile` would be placed into a separate directory `NAME`, rather than the current one. See §3.4 for further information on *build mode* vs. *example mode* of the `usedir` utility.

The `-q` option enables quiet mode, suppressing further notes on how to proceed.

Examples

The standard setup of a single "example session" based on the default logic, with proper document generation is generated like this:

```
isabelle mkdir Foo && isabelle make
```

The theory sources should be put into the `Foo` directory, and its `ROOT.ML` should be edited to load all required theories. Invoking `isabelle make` again would run the whole session, generating browser information and the document automatically. The `IsaMakefile` is typically tuned manually later, e.g. adding source dependencies, or changing the options passed to `usedir`.

Large projects may demand further sessions, potentially with separate logic images being created. This usually requires manual editing of the generated `IsaMakefile`, which is meant to cover all of the sub-session directories at the same time (this is the deeper reasong why `IsaMakefile` is not made part of the initial session directory created by `isabelle mkdir`). See `~~/src/HOL/IsaMakefile` for a full-blown example.

3.4 Running Isabelle sessions

The `usedir` utility builds object-logic images, or runs example sessions based on existing logics. Its usage is:

```
Usage: usedir [OPTIONS] LOGIC NAME

  Options are:
    -C BOOL      copy existing document directory to -D PATH (default true)
    -D PATH      dump generated document sources into PATH
    -M MAX       multithreading: maximum number of worker threads (default 1)
    -P PATH      set path for remote theory browsing information
    -Q INT       set threshold for sub-proof parallelization (default 50)
    -T LEVEL     multithreading: trace level (default 0)
    -V VARIANT   declare alternative document VARIANT
    -b           build mode (output heap image, using current dir)
    -d FORMAT    build document as FORMAT (default false)
    -f NAME      use ML file NAME (default ROOT.ML)
    -g BOOL      generate session graph image for document (default false)
    -i BOOL      generate theory browser information (default false)
    -m MODE      add print mode for output
    -p LEVEL     set level of detail for proof objects (default 0)
    -q LEVEL     set level of parallel proof checking (default 1)
    -r           reset session path
    -s NAME      override session NAME
    -t BOOL      internal session timing (default false)
    -v BOOL      be verbose (default false)

  Build object-logic or run examples. Also creates browsing
  information (HTML etc.) according to settings.

  ISABELLE_USEDIR_OPTIONS=

  ML_PLATFORM=x86-linux
  ML_HOME=/usr/local/polyml-5.2.1/x86-linux
  ML_SYSTEM=polyml-5.2.1
  ML_OPTIONS=-H 500
```

Note that the value of the `ISABELLE_USEDIR_OPTIONS` setting is implicitly prefixed to *any* `usedir` call. Since the `IsaMakefile`s of all object-logics distributed with Isabelle just invoke `usedir` for the real work, one may control compilation options globally via above variable. In particular, generation of HTML browsing information and document preparation is controlled here.

Options

Basically, there are two different modes of operation: *build mode* (enabled through the `-b` option) and *example mode* (default).

Calling `usedir` with `-b` runs `isabelle-process` with input image `LOGIC` and output to `NAME`, as provided on the command line. This will be a batch

session, running `ROOT.ML` from the current directory and then quitting. It is assumed that `ROOT.ML` contains all ML commands required to build the logic.

In example mode, `usedir` runs a read-only session of `LOGIC` and automatically runs `ROOT.ML` from within directory `NAME`. It assumes that this file contains appropriate ML commands to run the desired examples.

The `-i` option controls theory browser data generation. It may be explicitly turned on or off — as usual, the last occurrence of `-i` on the command line wins.

The `-P` option specifies a path (or actual URL) to be prefixed to any *non-local* reference of existing theories. Thus user sessions may easily link to existing Isabelle libraries already present on the WWW.

The `-m` options specifies additional print modes to be activated temporarily while the session is processed.

The `-d` option controls document preparation. Valid arguments are `false` (do not prepare any document; this is default), or any of `dvi`, `dvi.gz`, `ps`, `ps.gz`, `pdf`. The logic session has to provide a properly setup `document` directory. See §3.5 and §3.6 for more details.

The `-V` option declares alternative document variants, consisting of name/tags pairs (cf. options `-n` and `-t` of the `document` tool). The standard document is equivalent to "`document=theory,proof,ML`", which means that all theory begin/end commands, proof body texts, and ML code will be presented faithfully. An alternative variant "`outline=/proof/ML`" would fold proof and ML parts, replacing the original text by a short place-holder. The form "*name*`=-`," means to remove document *name* from the list of variants to be processed. Any number of `-V` options may be given; later declarations have precedence over earlier ones.

The `-g` option produces images of the theory dependency graph (cf. §3.2) for inclusion in the generated document, both as `session_graph.eps` and `session_graph.pdf` at the same time. To include this in the final LaTeX document one could say `\includegraphics{session_graph}` in `document/root.tex` (omitting the file-name extension enables LaTeX to select to correct version, either for the DVI or PDF output path).

The `-D` option causes the generated document sources to be dumped at location `PATH`; this path is relative to the session's main directory. If the `-C` option is true, this will include a copy of an existing `document` directory as provided by the user. For example, `isabelle usedir -D generated HOL`

`Foo` produces a complete set of document sources at `Foo/generated`. Subsequent invocation of `isabelle document Foo/generated` (see also §3.5) will process the final result independently of an Isabelle job. This decoupled mode of operation facilitates debugging of serious LaTeX errors, for example.

The `-p` option determines the level of detail for internal proof objects, see also the *Isabelle Reference Manual* [2].

The `-q` option specifies the level of parallel proof checking: `0` no proofs, `1` toplevel proofs (default), `2` toplevel and nested Isar proofs. The option `-Q` specifies a threshold for `-q2`: nested proofs are only parallelized when the current number of forked proofs falls below the given value (default 50), multiplied by the number of worker threads (see option `-M`).

The `-t` option produces a more detailed internal timing report of the session.

The `-v` option causes additional information to be printed while running the session, notably the location of prepared documents.

The `-M` option specifies the maximum number of parallel worker threads used for processing independent tasks when checking theory sources (multithreading only works on suitable ML platforms). The special value of `0` or `max` refers to the number of actual CPU cores of the underlying machine, which is a good starting point for optimal performance tuning. The `-T` option determines the level of detail in tracing output concerning the internal locking and scheduling in multithreaded operation. This may be helpful in isolating performance bottle-necks, e.g. due to excessive wait states when locking critical code sections.

Any `usedir` session is named by some *session identifier*. These accumulate, documenting the way sessions depend on others. For example, consider `Pure/FOL/ex`, which refers to the examples of FOL, which in turn is built upon Pure.

The current session's identifier is by default just the base name of the `LOGIC` argument (in build mode), or of the `NAME` argument (in example mode). This may be overridden explicitly via the `-s` option.

Examples

Refer to the `IsaMakefile`s of the Isabelle distribution's object-logics as a model for your own developments. For example, see `~~/src/FOL/IsaMakefile`. The Isabelle `mkdir` tool creates `IsaMakefile`s with proper invocation of `usedir` as well.

3.5 Preparing Isabelle session documents

The `document` utility prepares logic session documents, processing the sources both as provided by the user and generated by Isabelle. Its usage is:

```
Usage: document [OPTIONS] [DIR]

  Options are:
    -c            cleanup -- be aggressive in removing old stuff
    -n NAME       specify document name (default 'document')
    -o FORMAT     specify output format: dvi (default), dvi.gz, ps,
                  ps.gz, pdf
    -t TAGS       specify tagged region markup

  Prepare the theory session document in DIR (default 'document')
  producing the specified output format.
```

This tool is usually run automatically as part of the corresponding Isabelle batch process, provided document preparation has been enabled (cf. the `-d` option of the `usedir` tool). It may be manually invoked on the generated browser information document output as well, e.g. in case of errors encountered in the batch run.

The `-c` option tells the `document` tool to dispose the document sources after successful operation. This is the right thing to do for sources generated by an Isabelle process, but take care of your files in manual document preparation!

The `-n` and `-o` option specify the final output file name and format, the default is "`document.dvi`". Note that the result will appear in the parent of the target `DIR`.

The `-t` option tells LaTeX how to interpret tagged Isabelle command regions. Tags are specified as a comma separated list of modifier/name pairs: "`+`*foo*" (or just "*foo*") means to keep, "`-`*foo*" to drop, and "`/`*foo*" to fold text tagged as *foo*. The builtin default is equivalent to the tag specification "`+theory,+proof,+ML,+visible,-invisible`"; see also the LaTeX macros `\isakeeptag`, `\isadroptag`, and `\isafoldtag`, in `~~/lib/texinputs/isabelle.sty`.

Document preparation requires a properly setup "`document`" directory within the logic session sources. This directory is supposed to contain all the files needed to produce the final document — apart from the actual theories which are generated by Isabelle.

For most practical purposes, the `document` tool is smart enough to create any of the specified output formats, taking `root.tex` supplied by the user as

a starting point. This even includes multiple runs of LaTeX to accommodate references and bibliographies (the latter assumes `root.bib` within the same directory).

In more complex situations, a separate `IsaMakefile` for the document sources may be given instead. This should provide targets for any admissible document format; these have to produce corresponding output files named after `root` as well, e.g. `root.dvi` for target format `dvi`.

When running the session, Isabelle copies the content of the original `document` directory into its proper place within `ISABELLE_BROWSER_INFO`, according to the session path and document variant. Then, for any processed theory *A* some LaTeX source is generated and put there as *A*`.tex`. Furthermore, a list of all generated theory files is put into `session.tex`. Typically, the root LaTeX file provided by the user would include `session.tex` to get a document containing all the theories.

The LaTeX versions of the theories require some macros defined in `~~/lib/texinputs/isabelle.sty`. Doing `\usepackage{isabelle}` in `root.tex` should be fine; the underlying Isabelle `latex` tool already includes an appropriate path specification for TeX inputs.

If the text contains any references to Isabelle symbols (such as `\<forall>`) then `isabellesym.sty` should be included as well. This package contains a standard set of LaTeX macro definitions `\isasym`*foo* corresponding to `\<`*foo*`>`, see [3] for a complete list of predefined Isabelle symbols. Users may invent further symbols as well, just by providing LaTeX macros in a similar fashion as in `~~/lib/texinputs/isabellesym.sty` of the distribution.

For proper setup of DVI and PDF documents (with hyperlinks and bookmarks), we recommend to include `~~/lib/texinputs/pdfsetup.sty` as well.

As a final step of document preparation within Isabelle, `isabelle document -c` is run on the resulting `document` directory. Thus the actual output document is built and installed in its proper place (as linked by the session's `index.html` if option `-i` of `usedir` has been enabled, cf. §3.1). The generated sources are deleted after successful run of LaTeX and friends. Note that a separate copy of the sources may be retained by passing an option `-D` to the `usedir` utility when running the session.

3.6 Running LaTeX within the Isabelle environment

The `latex` utility provides the basic interface for Isabelle document preparation. Its usage is:

```
Usage: latex [OPTIONS] [FILE]

  Options are:
    -o FORMAT    specify output format: dvi (default), dvi.gz, ps,
                 ps.gz, pdf, bbl, idx, sty, syms

  Run LaTeX (and related tools) on FILE (default root.tex),
  producing the specified output format.
```

Appropriate LaTeX-related programs are run on the input file, according to the given output format: `latex`, `pdflatex`, `dvips`, `bibtex` (for `bbl`), and `makeindex` (for `idx`). The actual commands are determined from the settings environment (`ISABELLE_LATEX` etc.).

The `sty` output format causes the Isabelle style files to be updated from the distribution. This is useful in special situations where the document sources are to be processed another time by separate tools (cf. option `-D` of the `usedir` utility).

The `syms` output is for internal use; it generates lists of symbols that are available without loading additional LaTeX packages.

Examples

Invoking `isabelle latex` by hand may be occasionally useful when debugging failed attempts of the automatic document preparation stage of batch-mode Isabelle. The abortive process leaves the sources at a certain place within `ISABELLE_BROWSER_INFO`, see the runtime error message for details. This enables users to inspect LaTeX runs in further detail, e.g. like this:

```
cd ~/.isabelle/IsabelleXXXX/browser_info/HOL/Test/document
isabelle latex -o pdf
```

Chapter 4

Isabelle/Scala development tools

Isabelle/ML and Isabelle/Scala are the two main language environments for Isabelle tool implementations. There are some basic command-line tools to work with the underlying Java Virtual Machine, the Scala toplevel and compiler. Note that Isabelle/jEdit (§2.1) provides a Scala Console for interactive experimentation within the running application.

4.1 Java Runtime Environment within Isabelle

The Isabelle `java` utility is a direct wrapper for the Java Runtime Environment, within the regular Isabelle settings environment (§1.1). The command line arguments are that of the underlying Java version. It is run in `-server` mode if possible, to improve performance (at the cost of extra startup time).

The `java` executable is the one within `ISABELLE_JDK_HOME`, according to the standard directory layout for official JDK distributions. The class loader is augmented such that the name space of `Isabelle/Pure.jar` is available, which is the main Isabelle/Scala module.

For example, the following command-line invokes the main method of class `isabelle.GUI_Setup`, which opens a windows with some diagnostic information about the Isabelle environment:

```
isabelle java isabelle.GUI_Setup
```

4.2 Scala toplevel

The Isabelle `scala` utility is a direct wrapper for the Scala toplevel; see also `java` above. The command line arguments are that of the underlying Scala version.

This allows to interact with Isabelle/Scala in TTY mode like this:

```
isabelle scala
scala> isabelle.Isabelle_System.getenv("ISABELLE_HOME")
scala> isabelle.Isabelle_System.find_logics()
```

4.3 Scala compiler

The Isabelle `scalac` utility is a direct wrapper for the Scala compiler; see also `scala` above. The command line arguments are that of the underlying Scala version.

This allows to compile further Scala modules, depending on existing Isabelle/Scala functionality. The resulting class or jar files can be added to the `CLASSPATH` via the `classpath` Bash function that is provided by the Isabelle process environment. Thus add-on components can register themselves in a modular manner, see also §1.1.3.

Note that jEdit (§2.3) has its own mechanisms for adding plugin components, which needs special attention since it overrides the standard Java class loader.

Chapter 5

Miscellaneous tools

Subsequently we describe various Isabelle related utilities, given in alphabetical order.

5.1 Displaying documents

The `display` utility displays documents in DVI or PDF format:

```
Usage: display [OPTIONS] FILE

  Options are:
    -c           cleanup -- remove FILE after use

  Display document FILE (in DVI format).
```

The `-c` option causes the input file to be removed after use. The program for viewing `dvi` files is determined by the `DVI_VIEWER` setting.

5.2 Viewing documentation

The `doc` utility displays online documentation:

```
Usage: doc [DOC]

  View Isabelle documentation DOC, or show list of available documents.
```

If called without arguments, it lists all available documents. Each line starts with an identifier, followed by a short description. Any of these identifiers may be specified as the first argument in order to have the corresponding document displayed.

The `ISABELLE_DOCS` setting specifies the list of directories (separated by colons) to be scanned for documentations. The program for viewing `dvi` files is determined by the `DVI_VIEWER` setting.

5.3 Shell commands within the settings environment

The `env` utility is a direct wrapper for the standard `/usr/bin/env` command on POSIX systems, running within the Isabelle settings environment (§1.1). The command-line arguments are that of the underlying version of `env`. For example, the following invokes an instance of the GNU Bash shell within the Isabelle environment:

```
isabelle env bash
```

5.4 Getting logic images

The `findlogics` utility traverses all directories specified in `ISABELLE_PATH`, looking for Isabelle logic images. Its usage is:

```
Usage: findlogics

  Collect heap file names from ISABELLE_PATH.
```

The base names of all files found on the path are printed — sorted and with duplicates removed. Also note that lookup in `ISABELLE_PATH` includes the current values of `ML_SYSTEM` and `ML_PLATFORM`. Thus switching to another ML compiler may change the set of logic images available.

5.5 Inspecting the settings environment

The Isabelle settings environment — as provided by the site-default and user-specific settings files — can be inspected with the `getenv` utility:

```
Usage: getenv [OPTIONS] [VARNAMES ...]

  Options are:
    -a           display complete environment
    -b           print values only (doesn't work for -a)
    -d FILE      dump complete environment to FILE
                 (null terminated entries)

  Get value of VARNAMES from the Isabelle settings.
```

With the `-a` option, one may inspect the full process environment that Isabelle related programs are run in. This usually contains much more vari-

ables than are actually Isabelle settings. Normally, output is a list of lines of the form *name=value*. The `-b` option causes only the values to be printed.

Option `-d` produces a dump of the complete environment to the specified file. Entries are terminated by the ASCII null character, i.e. the C string terminator.

Examples

Get the ML system name and the location where the compiler binaries are supposed to reside as follows:

```
isabelle getenv ML_SYSTEM ML_HOME
  ML_SYSTEM=polyml
  ML_HOME=/usr/share/polyml/x86-linux
```

The next one peeks at the output directory for Isabelle logic images:

```
isabelle getenv -b ISABELLE_OUTPUT
  /home/me/isabelle/heaps/polyml_x86-linux
```

Here we have used the `-b` option to suppress the `ISABELLE_OUTPUT=` prefix. The value above is what became of the following assignment in the default settings file:

```
ISABELLE_OUTPUT="$ISABELLE_HOME_USER/heaps"
```

Note how the `ML_IDENTIFIER` value got appended automatically to each path component. This is a special feature of `ISABELLE_OUTPUT`.

5.6 Installing standalone Isabelle executables

By default, the main Isabelle binaries (`isabelle` etc.) are just run from their location within the distribution directory, probably indirectly by the shell through its `PATH`. Other schemes of installation are supported by the `install` utility:

```
Usage: install [OPTIONS]

  Options are:
    -d DISTDIR   use DISTDIR as Isabelle distribution
                 (default ISABELLE_HOME)
    -p DIR       install standalone binaries in DIR

  Install Isabelle executables with absolute references to the current
  distribution directory.
```

The `-d` option overrides the current Isabelle distribution directory as determined by `ISABELLE_HOME`.

The `-p` option installs executable wrapper scripts for `isabelle-process`, `isabelle`, `Isabelle`, containing proper absolute references to the Isabelle distribution directory. A typical `DIR` specification would be some directory expected to be in the shell's `PATH`, such as `/usr/local/bin`. It is important to note that a plain manual copy of the original Isabelle executables does not work, since it disrupts the integrity of the Isabelle distribution.

5.7 Creating instances of the Isabelle logo

The `logo` utility creates any instance of the generic Isabelle logo as an Encapsuled Postscript file (EPS):

```
Usage: logo [OPTIONS] NAME

  Create instance NAME of the Isabelle logo (as EPS).

  Options are:
    -o OUTFILE   set output file (default determined from NAME)
    -q           quiet mode
```

You are encouraged to use this to create a derived logo for your Isabelle project. For example, `isabelle logo Bali` creates `isabelle_bali.eps`.

5.8 Isabelle's version of make

The Isabelle `make` utility is a very simple wrapper for ordinary Unix `make`:

```
Usage: make [ARGS ...]

  Compile the logic in current directory using IsaMakefile.
  ARGS are directly passed to the system make program.
```

Note that the Isabelle settings environment is also active. Thus one may refer to its values within the `IsaMakefile`, e.g. `$(ISABELLE_OUTPUT)`. Furthermore, programs started from the make file also inherit this environment. Typically, `IsaMakefile`s defer the real work to the `usedir` utility.

The basic `IsaMakefile` convention is that the default target builds the actual logic, including its parents if appropriate. The `images` target is intended to build all local logic images, while the `test` target shall build all related examples. The `all` target shall do `images` and `test`.

Examples

Refer to the `IsaMakefile`s of the Isabelle distribution's object-logics as a model for your own developments. For example, see `~~/src/FOL/IsaMakefile`.

5.9 Make all logics

The `makeall` utility applies Isabelle make to any Isabelle component (cf. §1.1.3) that contains an `IsaMakefile`:

```
Usage: makeall [ARGS ...]

  Apply isabelle make to all components with IsaMakefile (passing ARGS).
```

The arguments `ARGS` are just passed verbatim to each `make` invocation.

5.10 Printing documents

The `print` utility prints documents:

```
Usage: print [OPTIONS] FILE

  Options are:
    -c           cleanup -- remove FILE after use

  Print document FILE.
```

The `-c` option causes the input file to be removed after use. The printer spool command is determined by the `PRINT_COMMAND` setting.

5.11 Remove awkward symbol names from theory sources

The `unsymbolize` utility tunes Isabelle theory sources to improve readability for plain ASCII output (e.g. in email communication). Most notably, `unsymbolize` replaces awkward arrow symbols such as `\<Longrightarrow>` by `==>`.

```
Usage: unsymbolize [FILES|DIRS...]

  Recursively find .thy/.ML files, removing unreadable symbol names.
  Note: this is an ad-hoc script; there is no systematic way to replace
  symbols independently of the inner syntax of a theory!

  Renames old versions of FILES by appending "~~".
```

5.12 Output the version identifier of the Isabelle distribution

The `version` utility displays Isabelle version information:

```
Usage: isabelle version [OPTIONS]

  Options are:
    -i           short identification (derived from Mercurial id)

  Display Isabelle version information.
```

The default is to output the full version string of the Isabelle distribution, e.g. "`Isabelle2012: May 2012`.

The `-i` option produces a short identification derived from the Mercurial id of the `ISABELLE_HOME` directory.

5.13 Convert XML to YXML

The `yxml` tool converts a standard XML document (stdin) to the much simpler and more efficient YXML format of Isabelle (stdout). The YXML format is defined as follows.

1. The encoding is always UTF-8.
2. Body text is represented verbatim (no escaping, no special treatment of white space, no named entities, no CDATA chunks, no comments).
3. Markup elements are represented via ASCII control characters $\mathbf{X} = 5$ and $\mathbf{Y} = 6$ as follows:

XML	YXML
<*name attribute*=*value* ...>	**X****Y***name*|**Y***attribute*=*value*...**X**
</*name*>	**X****Y****X**

   There is no special case for empty body text, i.e. `<foo/>` is treated like `<foo></foo>`. Also note that **X** and **Y** may never occur in well-formed XML documents.

Parsing YXML is pretty straight-forward: split the text into chunks separated by **X**, then split each chunk into sub-chunks separated by **Y**. Markup chunks start with an empty sub-chunk, and a second empty sub-chunk indicates close of an element. Any other non-empty chunk consists of plain text. For example, see `~~/src/Pure/PIDE/yxml.ML` or `~~/src/Pure/PIDE/yxml.scala`.

YXML documents may be detected quickly by checking that the first two characters are **XY**.

Bibliography

[1] D. Aspinall. Proof General. http://proofgeneral.inf.ed.ac.uk/.

[2] L. C. Paulson. *The Old Isabelle Reference Manual*. http://isabelle.in.tum.de/doc/ref.pdf.

[3] M. Wenzel. *The Isabelle/Isar Implementation*. http://isabelle.in.tum.de/doc/implementation.pdf.

[4] M. Wenzel. *The Isabelle/Isar Reference Manual*. http://isabelle.in.tum.de/doc/isar-ref.pdf.

Index