

/mnt/old-home/ttn/build/guile-baux (push)

The Guile-BAUX Scripts/Modules

Copyright © 2010, 2011 Thien-Thi Nguyen

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is included in the appendix entitled “GNU FDL”.

Table of Contents

The Guile-BAUX Scripts/Modules	1
1 Overview	2
2 Upstream	4
3 Invocation	5
3.1 Commands for Guile-BAUX introspection	5
3.2 Commands that work on/in the current installation	5
4 Integration	7
4.1 Bootstrapping	7
4.2 Using	7
4.3 Distributing	9
5 Reference	10
5.1 Program modules	10
5.1.1 re-prefixed-site-dirs	10
5.1.2 as-C-byte-array	10
5.1.3 frisk	11
5.1.4 inner-upstream	11
5.1.5 punify	11
5.1.5.1 command	12
5.1.5.2 exports	12
5.1.6 tsar	12
5.1.6.1 command	12
5.1.6.2 extraction	13
5.1.6.3 categories	13
5.1.6.4 aggregation	14
5.1.7 tsin	14
5.1.7.1 command	14
5.1.7.2 directives	15
5.1.7.3 formatting	15
5.1.8 c-tsar	17
5.1.8.1 command	17
5.1.8.2 C preprocessor	17
5.1.8.3 anatomy of a docstring	18
5.1.8.4 schemey arg names	18
5.1.8.5 options	19
5.1.8.6 aggregation	20
5.1.9 c2x	20

5.1.9.1	command	20
5.1.9.2	C preprocessor	20
5.1.9.3	condensation	21
5.1.10	gen-scheme-wrapper	21
5.1.10.1	command	21
5.1.10.2	module name construction	22
5.1.10.3	dlname construction	22
5.1.10.4	thunk name ~A interpolation	22
5.1.10.5	complete example	23
5.2	Support modules	23
5.2.1	common	23
5.2.2	forms-from	24
5.2.3	alist-from-plist	24
5.2.4	elide-dot-dotdot	24
5.2.5	file-newer-than	25
5.2.6	filenamez	25
5.2.7	frisker	25
5.2.8	pke	26
5.2.9	read-string	26
5.2.10	scheme-scanner	26
5.2.11	stemname	27
5.2.12	ts-base	27
5.2.12.1	two-part filename	27
5.2.12.2	texinfo snippet	27
5.2.12.3	archive	28
5.2.12.4	constant variables	29
5.2.12.5	other procedures	29
5.2.13	ts-output	29
5.2.14	write-string	29
5.2.15	temporary-file	30
5.2.16	a-dash-dash-b	30
6	Extending	31
6.1	script header, copyright notice	31
6.2	text for --help	31
6.3	define-module form	32
6.4	script body	32
6.5	main procedure	33
6.6	scripts that do not export	33
7	Personal Use	35
7.1	support	35
7.2	program	36
Appendix A	GNU Free Documentation License	37

Index..... **45**

The Guile-BAUX Scripts/Modules

This manual corresponds to Guile-BAUX 20120309.1509.1c4bb92
(‘ChangeLog’ last modified 2012-03-09 15:08:24 UTC).

Repository origin: `/mnt/old-home/ttn/build/guile-baux (fetch)`
(Canonical: <http://www.gnuvola.org/software/guile-baux.git>.)

1 Overview

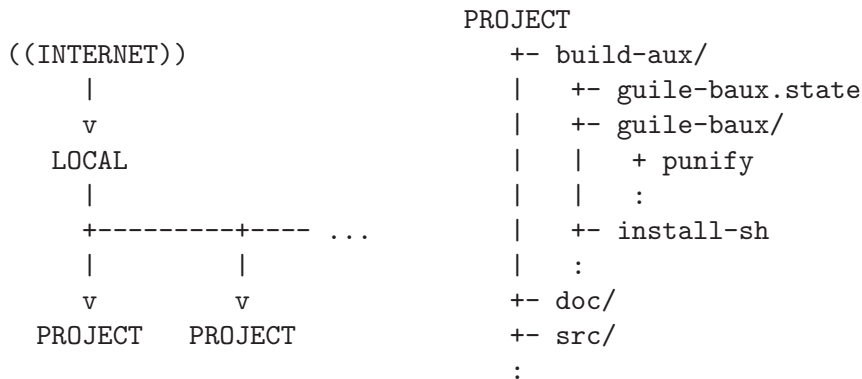
Guile-BAUX is a collection of scripts and modules intended to be installed in your project's *aux dir* (i.e., ‘`build-aux/`’, or equivalent dir specified by `AC_CONFIG_AUX_DIR` in ‘`configure.ac`’). There are modules for:

- extracting documentation / generating texinfo
- doing (Scheme) source-source scanning / transforming
- constructing / displaying modules’ dependency graph
- generating C boilerplate

Guile-BAUX also includes the program ‘`guile-baux-tool`’ for managing inter-module dependencies and doing the actual file copy/delete operations to keep the *current installation* in your project up to date.

In the spirit of `gnulib`¹, Guile-BAUX is not formally released and does not support ‘`make install`’. Instead, you clone the continuously evolving (but usably stable, one hopes) public repository and update it as you see fit (see [Chapter 2 \[Upstream\]](#), page 4).

From this local repository, you import modules into your project’s *aux dir* and use them from there. At any time you can drop modules you no longer need. See [Chapter 3 \[Invocation\]](#), page 5, for more info. In the *aux dir*, ‘`guile-baux.state`’ (i.e., the *state file*) records the list of requested modules, the list of supporting modules, and the list of files that actually manifest the functionality of these modules (in subdirectory ‘`guile-baux`’). Here is a small diagram showing how the current installation fits into the scheme of things:



Unlike `gnulib`, the focus of these modules (the scripts are also modules) is solely on supporting the portability of your project’s maintenance and build processes, rather than of the project’s code. Here, portability is between different versions of Guile, from 1.4.x and upward².

¹ homepage: <http://www.gnu.org/software/gnulib/>

² Long-time Guile watchers will note a similarity between this collection and the ‘`guile-tools`’ programs. In fact, most of these programs derive from Guile 1.4.x’s ‘`scripts/`’:

<http://www.gnuvola.org/software/guile/doc/Miscellaneous-Tools.html>

Moving them to a separate package is a kind of “distribution design bugfix”; had we done this from the beginning, we would have enjoyed faster updates and wider uptake, earlier. Luckily with Free Software, it’s never too late to try another way.

Furthermore, there is no automatic integration with the GNU autotools as is done with gnuilib; you need to make some straightforward decisions and enact them, manually (see [Chapter 4 \[Integration\]](#), page 7).

Lastly, to keep things simple, all code is GPLv3+ (see 'COPYING').

2 Upstream

Guile-BAUX is maintained in a Git repository. To create a local copy:

```
git clone http://www.gnuvola.org/software/guile-baux.git
```

This will create the subdirectory ‘guile-baux’ with many files, the most relevant being ‘guile-baux-tool.in’ in the top-level directory. To create ‘guile-baux-tool’ from this file, run ‘make’. See [Chapter 3 \[Invocation\], page 5](#), for how to use ‘guile-baux-tool’. Normally, this is all you need to do in terms of preparation, although it’s strongly suggested that you also take a look at the manual in subdirectory ‘doc’.

Although you can run ‘guile-baux-tool’ in place by typing its full filename for each invocation, it’s more convenient to set up a shell alias (if your shell has that feature) or a symlink. For example:

```
cd ../guile-baux
ln -s $(pwd)/guile-baux-tool ~/bin/gbaux
```

In this example, presuming ‘~/bin’ is to be found in the environment variable *PATH*, the symlink makes the shorter command ‘gbaux’ available. As a convenience, you can achieve the same effect with ‘make symlink to=~/bin/gbaux’ (see ‘GNUmakefile’). Be aware that if you use a hard link, running ‘make’ might not update ‘guile-baux-tool’ as you would expect.

Every so often, you might want to check the upstream repository to see if there are any interesting changes since the initial clone. If so, you can incorporate those changes into the local copy:

```
cd ../guile-baux
git pull
make
```

As a convenience, you can achieve the same effect with ‘make update’ (see ‘GNUmakefile’).

If you find a bug in Guile-BAUX, please report it to ttn@gnuvola.org (Thien-Thi Nguyen). In the meantime, if you know how to fix it, it’s a good idea to create another repository (thus, avoiding to confuse ‘git pull’), fix the problem there, and use that. In this case, you could clone upstream (again), but it’s more efficient to clone from the existing local one:

```
cd ../guile-baux
git clone ../guile-baux-fixed
#      ^-- note space
#      (two args: "." and "../guile-baux-fixed")
```

This example creates a sibling directory of ‘guile-baux’ named ‘guile-baux-fixed’. Don’t forget to run ‘make’ there, as well. This style of cloning is also useful for maintaining an “old” (stable, known good) Guile-BAUX tree on disk for use by one project while another advances to the latest update. For long-lived local repositories, consider also updating ‘doc/humble-narrator.texi’ so that downstream users know who to contact in case of problems. “Recursion: n. See recursion.”

3 Invocation

Aside from the usual ‘`--help`’ and ‘`--version`’ invocations, ‘`guile-baux-tool`’ is generally invoked as:

```
guile-baux-tool [options] command [args...]
```

That is, at least one arg, the *command*, must be given. Here are the recognized commands. If you specify args for a command that takes none, ‘`guile-baux-tool`’ signals a “spurious args” error.

3.1 Commands for Guile-BAUX introspection

‘`list`’ Display available module names to stdout, one per line, and exit. If ‘`--verbose`’, display more info on alternating lines.

‘`describe module...`’

‘`describe-all`’

Show information on each *module* and exit. For program modules, this additionally invokes the program with ‘`--version`’ and ‘`--help`’. The command ‘`describe-all`’ does this for all modules.

‘`history module...`’

Summarize the commit history (via ‘`git log`’) of the file that implements *module* and exit. If no module is specified, summarize the commit history of the entire repository.

3.2 Commands that work on/in the current installation

‘`status`’ Display current installation status, including the name of the state file (normally ‘`build-aux/guile-baux.state`’), the lists of requested and supporting modules, and the list of files involved.

‘`update`’ Write the state file, including (in a comment) the *Guile-BAUX version number*, which is the date and abbreviated hash of the HEAD commit, formatted as ‘`YYYYMMDD.HHMM.HASH`’. You can use Guile-BAUX version number in bug reports or to ‘`git checkout`’ (using the hash portion) a particular Guile-BAUX tree later.

Next, compute the list of files involved with the current set of requested and supporting modules and update the contents of aux dir subdirectory ‘`guile-baux`’ (by adding and deleting files). If ‘`guile-baux`’ becomes empty, delete it.

This command is useful for synchronizing your project after updating Guile-BAUX (see [Chapter 2 \[Upstream\]](#), page 4).

‘`import module...`’

Add each *module* to the list of requested modules (perhaps moving it from the list of supporting modules), and update the aux dir.

‘`drop module...`’

Remove each *module* from the list of requested modules (perhaps moving it to the list of supporting modules), and update the aux dir.

`'reset'` Apply `'drop'` on the full list of requested modules. Leave state file “empty” (but syntactically valid).

Some optional args modify the behavior of `'guile-baux-tool'`:

`'--verbose'` (short form: `'-v'`)

This makes `'guile-baux-tool'` display its proceedings.

`'--dry-run'` (short form: `'-n'`)

When specified, `'guile-baux-tool'` does not modify the file system, but instead displays what it **would** do.

4 Integration

This chapter describes how to integrate Guile-BAUX into the standard GNU autotools flow. Now that you have imported Guile-BAUX scripts and modules into your project (see [Chapter 2 \[Upstream\]](#), page 4, see [Chapter 3 \[Invocation\]](#), page 5), you probably want to use them, and you might want to distribute them so that your users can use them, too.

4.1 Bootstrapping

Most projects have a bootstrap script (typically named ‘`autogen.sh`’ or ‘`bootstrap.sh`’) that invokes ‘`autoreconf`’ (or its equivalent component `gettext`, `gettext`, `libtool`, and various auto tools commands) to prepare a freshly checked-out tree for “user-level” configuration (i.e., invocation of the ‘`configure`’ script). This is a good place to add a call to ‘`guile-baux-tool`’ as well. If you plan to keep the state file under version control, it is sufficient to add:

```
guile-baux-tool update
```

Otherwise, the ‘`guile-baux-tool`’ invocation should name all the modules your project wants to import:

```
guile-baux-tool import MODULE...
```

Don’t forget to mention Guile-BAUX and perhaps the Guile-BAUX version number (see [Chapter 3 \[Invocation\]](#), page 5) in ‘`HACKING`’.

4.2 Using

First, some background: Normally, given module name (`a b c`), Guile looks for, under each directory `dir` in the Scheme variable `%load-path`, a file ‘`dir/a/b/c`’ (or ‘`dir/a/b/c.scm`’) to load.

This direct correspondance between module name and directory/file name is the reason why all Guile-BAUX modules have names which start with `guile-baux` (e.g., the script ‘`punify`’ is implemented as a module named `(guile-baux punish)`), and are found in the `aux` dir subdirectory by the same name. In this arrangement, ‘`guile-baux`’ is like ‘`a/b`’, and the `aux` dir is like `dir`, above.

So, how can we add the `aux` dir to `%load-path`? There are many ways, but the most straightforward is to add the desired directory name “onto the front of” environment variable `GUILE_LOAD_PATH`. For example (using Bourne shell syntax):

```
# Don't clobber previous value, if any.
GUILE_LOAD_PATH="/hack/project/build-aux:$GUILE_LOAD_PATH"
export GUILE_LOAD_PATH

# Invocation; Guile should DTRT.
/hack/project/build-aux/guile-baux/punify ...
```

This is fine for invocation from the shell, but is unwieldy for invocation from a Makefile, as is typically the case. Luckily, Guile-BAUX provides the script ‘`gbaux-do`’¹, that sets the

¹ Guile-BAUX module name: `gbaux-do`; Scheme module name: `(guile-baux gbaux-do)`.

environment appropriately², and ‘exec’s its first argument, passing the rest of the arguments to its child process. The invocation thus shrinks from:

```
## Makefile.am fragment
.scm.puny:
    GUILE_LOAD_PATH="/hack/project/build-aux:$GUILE_LOAD_PATH" \
    /hack/project/build-aux/guile-baux/punify $< > $@
```

to:

```
## Makefile.am fragment
.scm.puny:
    /hack/project/build-aux/guile-baux/gbaux-do punify $< > $@
```

This is fine for one Makefile, but if you need to call ‘gbaux-do’ from many directories, you might want to add to ‘configure.ac’ a further abbreviation, something like:

```
dnl configure.ac fragment
dnl Use curly braces for usability in shell scripts.
AC_SUBST([gbauxdo],[’${top_srcdir}/build-aux/guile-baux/gbaux-do’])
```

```
## Makefile.am fragment
.scm.puny:
    $(gbauxdo) punify $< > $@
```

At last, something very easy on the eyes! Alternatively, if you invoke Guile-BAUX programs always from a makefile (never from a script), another way is to set the environment in the invocation command line directly. For this, you first arrange for the ‘configure’ script to define a makefile variable `auxd`; the rest can be done by defining variables in ‘common.mk’ (for instance) to be included by each ‘Makefile.am’.

```
dnl configure.ac fragment
AC_CONFIG_AUX_DIR([build-aux])
dnl Bummer, $ac_aux_dir is not documented as of Autoconf 2.65.
dnl AC_SUBST([auxd],[’$ac_aux_dir’])
AC_SUBST([auxd],[build-aux])

## common.mk fragment
abs_gbauxd = $(abs_top_srcdir)/$(auxd)/guile-baux
# NB: double '$'; '#' immediately after to ensure no trailing space.
gbauxdo = GUILE_LOAD_PATH="$(abs_gbauxd):$$GUILE_LOAD_PATH" \
    $(abs_gbauxd)/#
# NB: no space between $(gbauxdo) and the command
punify = $(gbauxdo)punify

## Makefile.am fragment
include $(top_srcdir)/../common.mk
.scm.puny:
    $(punify) $< > $@
```

² Actually, to keep Guile-BAUX installation simple, this script relies on the name by which it was invoked to compute the parent directory. This means it fails if invoked through a symbolic or hard link. Lame! This is a code-enhancement opportunity (patches welcome).

4.3 Distributing

To distribute the current installation, add the names of the desired files to the top-level ‘Makefile.am’ variable `EXTRA_DIST`. If you want to distribute all of them, simply add ‘guile-baux’. If you distribute a bootstrap script that does ‘guile-baux-tool update’ (see [Section 4.1 \[Bootstrapping\], page 7](#)), you should also distribute the state file, as well. For example:

```
## top-level Makefile.am fragment
EXTRA_DIST += $(auxd)/guile-baux
EXTRA_DIST += $(auxd)/guile-baux.state
```

Here, `auxd` is defined by the ‘configure’ script (see [Section 4.2 \[Using\], page 7](#)).

5 Reference

This chapter describes all the program/support modules; each section *foo* describes the Guile-BAUX module of the same name, implemented by a scheme module named `'(guile-baux foo)'`.

5.1 Program modules

Note that a program that exports items functions as a support module, as well.

5.1.1 re-prefixed-site-dirs

Usage: `re-prefixed-site-dirs guile-config-program vprefix`

Retrieve build information either from `%guile-build-info` (if builtin) or by running `guile-config-program` (otherwise), and display three lines representing a shell-script fragment that sets variables `vprefix_libsite`, `vprefix_site` and `vprefix_cv_minstroot`.

The last variable is usually equal to one of the first two. It is distinguished by being the first directory in `%load-path` whose name ends in `‘/site’`. Also, its name includes `‘_cv_’`, making it easy to cache.

All values are *re-prefixed*, which means the initial portion is replaced with either `#{prefix}` or `#{exec_prefix}`. This is useful when `‘make check’` does an in-tree install and needs to embed (before `‘make install’`) or otherwise fix up (after `‘make install’`) temporary paths.

Typically, you would use this program by calling it from the `‘configure’` script, capturing and caching its output for further processing. For example:

```

GUILE_PROGS

AC_CACHE_CHECK([where to install modules],[mypkg_cv_minstroot],[
eval 'GUILE="$GUILE" \
      $srcdir/build-aux/guile-baux/gbaux-do \
      re-prefixed-site-dirs "$GUILE_CONFIG" mypkg'
])

GUILE_LIBSITE="$mypkg_cv_minstroot"
AC_SUBST([GUILE_LIBSITE])

```

In this example we use `GUILE_PROGS` to determine the absolute filenames for programs `‘guile’` and `‘guile-config’`, and `‘gbaux-do’` for dispatch (see [Section 4.2 \[Using\], page 7](#)).

The result is normally `#{exec_prefix}/lib/guile/site` for Guile 1.4.x, and `#{prefix}/share/guile/site` for other versions.

5.1.2 as-C-byte-array

Usage: `as-C-byte-array [options] [file...]`

Write concatenated contents of *file...* as one C `uint8_t` array. The array length is one more than the combined file sizes; its last element is always `\0` (i.e., `#\nul`).

```

-g, --global          omit static qualifier
-t, --type NAME       use type name [uint8_t]

```

`-v, --var NAME` use variable *name* [bytes]

This program is useful for embedding source code (e.g., for “internal load” implementation). In such cases, you probably want to remove extraneous bits to make loading more efficient (see [Section 5.1.5 \[punify\]](#), page 11).

5.1.3 frisk

Usage: `frisk [options] file...`

Analyze *file...* module interfaces in aggregate (as a *body*), and display a summary. Modules that are `define-moduled` are considered *internal* (and those not, *external*). When module *x* uses module *y*, *x* is said to be (a) *downstream* of *y*, and *y* is (an) *upstream* of *x*.

Normally, the summary displays external modules and their internal downstreams, as this is the usual question asked by a body. There are several options that modify this output.

`-u, --upstream` show upstream edges
`-d, --downstream` show downstream edges (default)
`-i, --internal` show internal modules
`-x, --external` show external modules (default)

If given both upstream and downstream options (`'frisk -ud'`), the output is formatted:

```
c module --- up-ls --- down-ls
```

where *c* is either ‘i’ or ‘x’, and each element of *up-ls* and *down-ls* is:

```
(type module-name ...)
```

In all other cases, the "*c module*" occupies its own line, and subsequent lines list the up- or downstream edges, respectively, indented by some non-zero amount of whitespace.

Top-level `use-modules` (or `load` or `primitive-load`) forms in a file that do not follow a `define-module` result in an edge where the downstream is the *default module*, normally (`guile-user`). This can be set to another value by using the option `'--default-module MOD'` (short form `'-m'`).

Another option is `'--filez'` (short form `'-z'`), which reads NUL-terminated filenames from the standard input, in addition to those specified on the command-line.

5.1.4 inner-upstream

Usage: `inner-upstream [options] [file...]`

Frisk *file...* and display the inner upstreams of modules with the *common prefix*, formatted as specified.

Options (defaults in square braces):

`-c, --common MPRE` omit modules not named with prefix *mpre* [()]
`-l, --leaf` output (NAME) as NAME
`-f, --format FMT` format for output ["~S~%"]
`-z, --filez` read NUL-terminated filenames from stdin

5.1.5 punish

5.1.5.1 command

Usage: `punify [-i] [-n] [file...]`

Read forms from *file...* (or standard input if no files are specified) and write them to standard output, removing comments and non-essential whitespace. This is useful when installing Scheme source to space-limited media. An exception is made for certain whitespace characters appearing in a string. They are expanded to their two-character “escaped” form (see `write-punily`, below).

Option ‘`--newline-after-top-level-form`’ (or ‘`-n`’ for short) means to output a new-line after each top-level form. Option ‘`--inplace`’ (or ‘`-i`’ for short) means to modify the file in place, displaying to standard output an informational message for each file processed.

5.1.5.2 exports

`write-punily form` [Procedure]

Write *sexp form* to the current output port, avoiding unnecessary whitespace. However, write strings with certain whitespace characters expanded to their two-character “escaped” form:

```
#\bel  \a      #\newline \n      #\ht   \t
#\np   \f      #\cr      \r      #\vt   \v
```

form should contain only objects that can be externally represented with `display` and `write`.

`write-line-punily form` [Procedure]

Write *form* punily (via `write-punily`), then display a `#\lf` character.

5.1.6 tsar

5.1.6.1 command

Usage: `tsar [options] command file...`

Create or update a texinfo snippet archive, scanning Scheme source files in the process.

Commands:

```
create  scan file...; write a new archive
update  scan file...; update entries in an existing archive, creating one if necessary
rescan  ignore file...; instead, scan only files already named in the archive that are
        newer than the archive; update entries
concat  create a new archive from archive file...
```

Options (defaults in square braces):

```
-f, --file ARCHIVE  Operate on archive.
-c, --coding CODING Use encoding coding [binary].
-z, --zstdin        Read NUL-terminated filenames from stdin.
-l, --language NAME Prefix category with name.
-m, --default MOD   Use mod for non-moduled items [(guile-user)].
-v, --verbose        Display information to stderr.
```

Commands ‘`update`’ and ‘`rescan`’ require ‘`-f`’. If both ‘`-z`’ and *file...* are specified, ‘`-z`’ filenames are processed first.

5.1.6.2 extraction

To *scan* is to read in a Scheme source file and extract texinfo snippets in the form of a comment block preceding and “touching” a top-level `define` (or similar) form, with each line in the comment beginning flush left and starting with an equal number of semicolons. A snippet may include trailing *options*, one per line, of the form:

```
-key: value
```

Note the leading ‘-’ (hyphen). Scanning separates the options from the rest of the text, and associates these parts with an inferred name, module, category, arglist (if applicable) and originating location (filename, line and column numbers, byte offsets delimiting the documented item’s region). See [Section 5.2.12 \[texinfo snippet\], page 27](#). For example, the file `/tmp/zzz` (with line numbers on the left):

```
1 (define-module (my mod))
2
3 ;; This procedure foos, or bars, depending on @var{baz}.
4 ;;-Author: Martin Grabmueller
5 (define (foo/bar baz)
6   (if baz (foo) (bar)))
```

results in the following information:

```
name:      foo/bar
module:    (my mod)
category:  procedure
arglist:   (baz)
location:  /tmp/zzz, line 5, column 0, region 113-158
options:   ((Author . "Martin Grabmueller"))
blurb:     "This procedure foos, or bars, depending on @var{baz}."
```

Scanning also recognizes comment blocks whose first line has the form `ws{title}ws`, where `ws` is horizontal whitespace (space or tab), for example:

```
;; {Elucidation of Flow}
;;
;; We use the ‘‘flow’’ (@pxref{Flow Details})
;; algorithm because it’s so pleasant.
```

In this case, the texinfo snippet name is *title* (a string) and the category `#f`, and options are not recognized.

5.1.6.3 categories

With the exception of `expression`, categories are all definitions of some type or other.

syntax A macro definition, recognized by `defmacro`, `defmacro-public`, `defmacro*`, `defmacro*-public`, or `define-syntax`.

procedure

A procedure definition, recognized by either:

`(define (name ...) ...)` or `(define ... (lambda ...))`, i.e., a definition with the last (recursively extracted) element of the sequence a `lambda` form.

In the place of `define`, there may be instead `define-public`, `define*` or `define*-public`. In the place of `lambda`, there may be instead `lambda*`.

type A `define-record-type` form.

define-module
 A `define-module` form.

define
define-public
 A definition that doesn't fit elsewhere.

expression
 A non-definition form or literal.

These categories are a good fit for documenting pure-Scheme programs and libraries. In the case where Scheme code is mixed with other (typically, C language) code, you can use the command-line option `--language` to prefix the category with a name useful for disambiguating the mixed interfaces. For example, specifying `-l scheme` results in `scheme syntax`, `scheme procedure`, and so on.

5.1.6.4 aggregation

A *texinfo snippet archive* file is an aggregation of texinfo snippets in which a `name`, `module` pair is unique. On collision between old and new snippets, `tsar` saves the new, discarding the old.

5.1.7 tsin

5.1.7.1 command

Usage: `tsin [options] file...`

Process `@tsin` directives in each *file*, writing output to a new file made by replacing the extension (if any) of the `basename` of *file* with `.texi` (e.g., for *file* `'a/b/c.d.e'`, write `'c.d.texi'` in the current working directory).

Options (defaults in square braces):

<code>-c, --coding CODING</code>	Use encoding <i>coding</i> [<code>binary</code>].
<code>-f, --file ARCHIVE</code>	Consult <i>archive</i> for texinfo snippets.
<code>-m, --default MOD</code>	Use <i>mod</i> for non-moduled items [<code>(guile-user)</code>].
<code>-e, --error-if-missing</code>	Fail if a reference cannot be resolved.
<code>-s, --same-dir</code>	Write to same directory as <i>file</i> [<code>cwd</code>].
<code>-v, --verbose</code>	Write progress info to stderr.

The option `-f` is obligatory. Presuming *file* lives in the source tree (as opposed to the build tree), you can use option `-s` to better conform to the GNU Coding Standards, which says:

GNU distributions usually contain some files which are not source files—for example, Info files, and the output from Autoconf, Automake, Bison or Flex. Since these files normally appear in the source directory, they should always appear in the source directory, not in the build directory. So Makefile rules to update them should put the updated files in the source directory.

See Section “Makefile Basics” in *The GNU Coding Standards*.

5.1.7.2 directives

Most lines (bytes delimited by newline or end-of-file) are passed to the output. The exceptions are *directive lines*, of the form:

```
@tsin directive args...
```

A directive may write zero or more lines to the output file. Here are the directives:

```
c text [tsin Directive]
C text [tsin Directive]
```

These are *comment* directives, intended for the input file author to communicate progress. They display the proper (i.e., read-able) Scheme string *text* to stderr with format:

```
filename:lno: text
```

The difference between them is that `c` only writes when given the command-line option ‘`--verbose`’, whereas `C` writes unconditionally.

```
m module [tsin Directive]
```

Aside from the current position of the input file, ‘`tsin`’ maintains one other piece of state: the order of the list of referenced modules read in from the archive, used for the `i` directive (described below). The `m` directive moves *module* (a list of symbols, aka module name) to the head of this list. Additionally, as a convenience, ‘`tsin`’ emits the line:

```
@set TSINCURMOD module
```

You can use `@value{TSINCURMOD}` henceforth (until the next `m` directive) to stand for *module*.

```
i name [module [flags...]] [tsin Directive]
```

This is the *interpolation* directive. ‘`tsin`’ looks up the texinfo snippet with *name* and *module* and formats it to the output. If *module* is omitted or ‘`-`’ (hyphen), ‘`tsin`’ searches the list of referenced modules. If the found module *actual* is not the first, perform an implicit `m actual` (emitting `@set TSINCURMOD actual`) prior to writing the rest of the output.

If a reference cannot be resolved, ‘`tsin`’ writes a message to standard error. Then, if given command-line option ‘`--error-if-missing`’, ‘`tsin`’ exits failurefully.

The *flags* are symbols that affect formatting (see below).

5.1.7.3 formatting

On output, ‘`tsin`’ formats each texinfo snippet *ts* according to its category and other attributes (see [Section 5.2.12 \[ts-base\]](#), page 27).

- If *ts* has no category, check *flags* and output the *ts* name (the title of the titled text block) using the first flag, if specified, as a Texinfo heading command, followed by a blank line. As a special case, ‘`-`’ (hyphen) stands for **heading**. Then output the *ts* blurb and a newline, without further processing. For example, this input:

```
(before)
@tsin i "Cool Stuff"
@tsin i "Cool Stuff" - section
```

(after)

produces this output:

(before)

blurb

@section Cool Stuff

blurb

(after)

- If *ts* has args, output @def_{fn}, otherwise @def_{vr}. This begins the *definition line*.
- Output the category, converted to a string (if necessary) and capitalized. Two examples:

procedure ⇒ "Procedure"

"constant variable" ⇒ "Constant Variable"

- (if args) If *ts* includes a sig option, output that (string value) directly. Otherwise output the args formatted according to the req, opt, var specification (*ts sig*). Basically, the first req argument names are displayed as-is; the next opt and var argument names are displayed with a leading square bracket; if var is non-zero, display ‘...’; finish with the matching number of square brackets. Here are some examples, assuming the argument names are a, b, c, etc.

n 0 0 ⇒ arg₁ arg₂ ... arg_n (n > 0)

0 0 0 ⇒

0 0 1 ⇒ [a...]

0 1 0 ⇒ [a]

0 2 0 ⇒ [a [b]]

0 1 1 ⇒ [a [b...]]

0 2 1 ⇒ [a [b [c...]]]

4 2 0 ⇒ a b c d [e [f]]

If an argument name is a list, it undergoes further *keyword argument* processing, intended to reduce clutter on the definition line:

- In the definition line, output @t{keyword value}@dots{} in place of the list, mixed in with the other arguments.
- Between the definition line and the *ts* blurb, output @cindex entries, one line per list element, of the form:

@cindex @t{#:keyword-arg-name}, @t{procedure-name}

Note the ‘#:’ (hash, colon) prefix. Also, you will need to include a concept index, or an index which includes concepts, in your document, for these entries to be useful (see Section “Indices” in *The GNU Texinfo Manual*).

To disable @cindex entries, specify flag no-kw-index.

- Output a comma-separated list of keyword arg names, indented with the leading text ‘Keywords:’ and a ragged right edge (should the list require more than one line)¹.

¹ Implementation Note: We render the keyword arg list with @example instead of (say) @quotation and @raggedright because the latter is not yet (2010Q3) widely available. This approach requires “manual” filling, with fill column hardcoded to 72. Obviously a Code Improvement Opportunity...

To disable this output, specify flag `no-kw-list`. To output the list sorted (with `string<?`), specify flag `kw-sort`.

For example, given a procedure `foo` with one *req* arg, no *opt* args, and two *var* args with names (a (b c d) e), the initial output would be:

```
@deffn {Procedure} foo a [@t{keyword value}@dots{} [e@dots{}]]
@cindex @t{#:b}, @t{foo}
@cindex @t{#:c}, @t{foo}
@cindex @t{#:d}, @t{foo}
@example
@r{Keywords:} b@r{,} c@r{,} d
@end example
```

- Output the *ts* blurb, followed by a newline.
- Output the matching `@end` (`deffn` or `defvr`) line.

5.1.8 c-tsar

5.1.8.1 command

Usage: `c-tsar` [options] command file... -- [cpp-options]

Create or update a texinfo snippet archive, scanning C source files in the process. Commands:

```
create  scan file...; write a new archive
update  scan file...; update entries in an existing archive, creating one if necessary
rescan  scan files named in an existing archive which are newer than the archive;
        update entries
```

Options (defaults in square braces):

```
--cpp PROGRAM    Use program to preprocess (see below).
-f, --file ARCHIVE  Operate on archive.
-c, --coding CODING Use encoding coding [binary].
-l, --language NAME Prefix category with name.
-m, --default MOD   Use mod for non-moduled items [(guile-user)].
-v, --verbose       Display information to stderr.
```

Commands ‘update’ and ‘rescan’ require ‘-f’. The default C preprocessor is taken as either the value of env var `CPP`, or ‘`cpp`’. You can specify a program with args for *program* (e.g., ‘`gcc -E`’).

5.1.8.2 C preprocessor

Document extraction is done by applying the C pre-processor to each file along with *cpp-options*², and processing the “magically snarfed” output. In addition to the usual magic, ‘`c-tsar`’ uses extra magic powers to determine and abstract over *revised* vs *classic* magic (tried in the that order).

The revised magic involves defining `SCM_MAGIC_SNARF_DOCS` and looking for entries framed by “`^^ {`” and “`^^ }`”. At the moment ‘`c-tsar`’ supports *procedure* snarfing (as generated by `SCM_DEFINE`) only.

² Note the ‘--’ (dash-dash) used to separate the program arguments from the *cpp-options*.

The classic magic involves defining `SCM_MAGIC_SNARFER` and looking for entries framed by `"SCM_I"` and `"SCM_E"`. `'c-tsar'` supports DP, DR and D1 snarfing (as generated by `SCM_DEFINE`, `SCM_REGISTER_PROC` and `SCM_DEFINE1`, respectively).

The `'c-tsar'` magic is not extensible (although it might be in the future); please see the source and the header `'libguile/snarf.h'` for more information. In any case, only if this phase completes successfully for all (relevant) files does `'c-tsar'` update the archive.

5.1.8.3 anatomy of a docstring

Let's look at the different forms of docstring `'c-tsar'` recognizes. First is the “series of C strings” form, in which every line of the docstring is represented by a C string most-commonly ending with newline character.

```
SCM_DEFINE
(foo, "foo", 2, 0, 0,
 (SCM bar, SCM baz),
 "Doc string, first line.\n"
 "Line 2; end of paragraph.\n\n"
 "More \"information\" here!")
```

Note that each internal `"` (double-quote) must be escaped in this form. An empty line (to separate paragraphs) can manifest on a physical line of its own or as an extra newline, as in “Line 2 . . .” above. Because this form is somewhat tedious to maintain, `'c-tsar'` supports an alternative syntax, used by Emacs' source code (the parts written in C). Here is the above example, reformulated using the “Emacs” form:

```
(foo, "foo", 2, 0, 0,
 (SCM bar, SCM baz),
 doc: /*****
Doc string, first line.
Line 2; end of paragraph.

More "information" here! */)
```

The series of C strings is replaced by the token `'doc:'` followed by a stylized comment. The text in the comment no longer needs explicit newlines or any escaping for double quotes. On the other hand, the formatting of the token with respect to the comment start is not flexible; it must match the regular expression:

```
"^[\t]*doc: /[*][^ ]+ $"
```

Note that simply `"/*"` does NOT match. This restriction may be lifted in the future. Another drawback of the Emacs form is that you cannot include `"*/"` in the docstring. (If you need to do that, use the series of C strings form.)

5.1.8.4 schemey arg names

When a procedure (or macro) takes arguments, `'c-tsar'` unconditionally³ changes `'_'` (underscore) to `'-'` (hyphen) in those argument names, so you need to use hyphens in the docstring, as well. For example:

³ This just looks better; don't fight it.

```
SCM_DEFINE
(zz, "zz", 4, 0, 1,
 (SCM a_long_variable_name,
  SCM another_one,
  SCM too_much_too_much,
  SCM i_say,
  SCM rest),
 doc: /*****
You really want to keep @var{a-long-variable-name},
@var{another-one} and @var{too-much-too-much} apart
from what @var{i-say} and all the @var{rest}.
Is that clear? */)

```

5.1.8.5 options

A docstring (in any of the recognized forms described above) can include *options* (see [Section 5.1.6 \[tsar\], page 12](#)), most of which ‘c-tsar’ saves transparently into the archive. The unique exception is the option `args`, which can be used to modify or even completely replace the “req/opt/var counts” and argument names as scanned from the C file. This is useful for adding additional abstraction or stylization, especially in light of hairy (yet sanely documentable) internal argument manipulation. For example:

```
SCM_DEFINE
(zz, "zz", 1, 0, 1,
 (SCM first, SCM rest),
 doc: /*****
Do something to @var{one} as
well as @var{two} and @var{three}.

-args: (3 0 0 one two three) */)

```

Here, the `args` are saved as `#(3 0 0 one two three)`, to be subsequently (see [Section 5.1.7 \[tsin\], page 14](#)) rendered as:

```
@defn {Procedure} zz one two three
```

instead of:

```
@defn {Procedure} zz first [rest@dots{}]
```

(had `args` not been specified). Generally, the value of the `args` option must be a proper list having one of the forms (examples are based on the one above, showing only differences):

```
(req opt var [name...])
```

This is the *fully-specified* form exemplified above.

```
(- opt var [name...])
```

This uses the scanned *req* (and *req* names) directly, which is nice if *req* is lengthy or the C-sourced names are acceptable as-is. The argument names are mapped to *opt* and *var*.

```
(- 2 0 two three) ⇒ #(1 2 0 first two three)
(- 1 1 two three) ⇒ #(1 1 1 first two three)
(- 2 1 two three four) ⇒ #(1 2 1 first two three four)

```

(*name...*)

This maps *name...* to the scanned *req*, *opt* and *var*.

```
(head tail) ⇒ #(1 0 1 head tail)
(a b) ⇒ #(1 0 1 a b)
```

Note that the saved arg vector need not correspond in any way to the reality of the C source. It is up to you to choose properly congruent values.

5.1.8.6 aggregation

See [Section 5.1.6 \[tsar\]](#), page 12, for a description of this phase. (However, ‘c-tsar’, unlike ‘tsar’, does not provide the ‘concat’ command.)

5.1.9 c2x

5.1.9.1 command

Usage: `c2x [options] infile -- [cpp-options]`

Process *infile* using the C preprocessor. Write output to a file or to the standard output when no filename has been specified or when the filename is "-". If there are errors during processing, delete the output file and exit with non-zero status.

Options (defaults in square braces):

```
-o, --output OUTFILE Write to outfile [stdout].
--cpp PROGRAM Use program to preprocess (see below).
-d, --dumb Inhibit condensation pass.
```

The default C preprocessor is taken as either the value of env var `CPP`, or ‘cpp’. You can specify a program with args for *program* (e.g., ‘gcc -E’).

5.1.9.2 C preprocessor

Extraction is done by applying the C pre-processor to *infile* along with *cpp-options*⁴, and processing the “magically snarfed” output.

Immediately prior to invoking the C pre-processor, if *outfile* is specified, ‘c2x’ writes a C comment into *outfile*. This is because normally *infile* ‘#include’s *outfile* and often that directive is not guarded. It is not sufficient to simply touch *outfile*, as some older C pre-processors have trouble with empty input.

In addition to the usual magic, ‘c2x’ uses extra magic powers to determine and abstract over *revised* vs *classic* magic (tried in the that order).

The revised magic involves defining `SCM_MAGIC_SNARF_INITS` and looking for entries framed by “^^” and “^: ^”.

The classic magic involves defining `SCM_MAGIC_SNARFER` and looking for entries framed by “SCM_I” and “SCM_D”.

The ‘c2x’ magic is not extensible (although it might be in the future); please see the source and the header ‘libguile/snarf.h’ for more information.

⁴ Note the ‘--’ (dash-dash) used to separate the program arguments from the *cpp-options*.

5.1.9.3 condensation

'c2x' knows how to condense some kinds of entries, converting repeated calls to some init function (with varying args) to one or more space-efficient data tables specifying the args and a loop on that init function. This reduces the code footprint which may yield some performance gain.

```
/* before */
scm_make_gsubr (s_lob_lo_creat, 2, 0, 0, (SCM (*)()) lob_lo_creat);
scm_make_gsubr (s_lob_lo_open, 3, 0, 0, (SCM (*)()) lob_lo_open);
kwd_envvar = scm_permanent_object (scm_c_make_keyword ("envvar"));
kwd_compiled = scm_permanent_object (scm_c_make_keyword ("compiled"));

/* after */
{
  struct { SCM *to; char const *s; } *pairs1w, pairs1[2] = {
    {&kwd_compiled, "compiled"},
    {&kwd_envvar, "envvar"}
  };
  for (pairs1w = pairs1; pairs1w < pairs1 + 2; pairs1w++)
    *pairs1w->to = scm_permanent_object (scm_c_make_keyword (pairs1w->s));
}
{
  struct { char const *nm; SCM (*fn)(); } *d, def[2] = {
    {s_lob_lo_open, lob_lo_open},
    {s_lob_lo_creat, lob_lo_creat}
  };
  /* NOTE: All v are zero. */
  unsigned char *r, rov[2] = {
    /* (3 0 0) */ 0x03,
    /* (2 0 0) */ 0x02
  };
  for (d = def, r = rov; r < rov + 2; d++, r++)
    scm_make_gsubr (d->nm, (*r) & 0xf, ((*r) >> 4) & 0xf, 0, d->fn);
}
```

In this example, all the procedures have a *var count* (from the triple (REQ OPT VAR)) of zero, so 'c2x' packs each triple into a single byte. Presence of a procedure with non-zero var count forces use of `unsigned short`, instead.

The current 'c2x' can condense these classic⁵ constructs: `SCM_DEFINE`, `SCM_KEYWORD`, `SCM_SYMBOL`. To disable condensation completely, specify '`--dumb`' on the command line.

5.1.10 gen-scheme-wrapper

5.1.10.1 command

Usage: `gen-scheme-wrapper [options] la-file`

⁵ Support for revised snarfing is currently (2011 Q2) work-in-progress. Stay tuned!

Output two Scheme forms that can dynamically link and initialize the shared object library represented by libtool archive description file *la-file*, additionally (re-)exporting the library's interface elements. *la-file* should be named *stem.la*.

Options (defaults in square braces):

```
-o, --output OUTFILE Write to outfile [stdout].
-g, --group GROUP   Use group as module name prefix.
-m, --module NAME   Specify name explicitly.
-e, --exports FILE  Read exports from file [stem.exports].
-t, --thunk FUNC    Call func to initialize the module.
-i, --instdir DIR   Look for shared object library in dir.
-d, --dlname        Mine dlname from la-file [stem].
-v, --verbose       Display progress messages to stderr.
```

If ‘--module NAME’ is specified, that is the module name to use. Otherwise, the *group* (either a single symbol or a list of symbols) is appropriately prefixed onto *stem* to form the module name. Both ‘--thunk’ and ‘--instdir’ are required. If *func* contains the two-character sequence ‘~A’, then that portion is replaced by the module name, without parentheses, and with characters not in `char-set:letter+digit` replaced by underscore.

5.1.10.2 module name construction

Here are some examples that show the resulting module name given various command-line options. Note that ‘--module’ overrides everything.

```
foo.la ⇒ (foo)
-g foo bar.la ⇒ (foo bar)
-g '(foo bar)' baz.la ⇒ (foo bar baz)
-m '(a b c)' -g foo bar.la ⇒ (a b c)
```

5.1.10.3 dlname construction

Normally, the filename specifying the shared-object library is a straightforward concatenation (more precisely, using *in-vicinity*) of *dir*, from ‘--instdir’, and *stem*. Optional arg ‘--dlname’ means to mine the *filename* from the line in *la-file* that has the form:

```
dlname='filename'
```

and use that in place of *stem*. Most of the time, this will in fact include *stem* plus one or more system-specific extensions (e.g., ‘foo.so.0’). Unfortunately, whether or not one should use ‘--dlname’ is highly dependent on the collective mood of Libtool’s libltdl, Guile’s module system support for dlopening and, of course, the phase of the moon. Thus, we have not yet characterized (for automatic configuration) the process. This means you might have to expose this weary detail to the package users. We hope to study this more and handle things better in the future.

5.1.10.4 thunk name ~A interpolation

Here are some examples that show the resulting thunk name given various module names and the constant ‘-t bef_~A_aft’ command-line option.

```
(foo) ⇒ bef_foo_aft
(a b c) ⇒ bef_a_b_c_aft
(!ok!9 -z-) ⇒ bef__ok_9__z__aft
```

```
(++ -- ++) ⇒ bef_____aft
```

5.1.10.5 complete example

Here is a Makefile fragment to create a wrapper for (my mod), with the shared object library to be installed in pkglibdir. The init thunk is named `init_my_mod`.

```
mod.exports: $(mod_la_SOURCES)
> $@ \
$(SED) -e '/^(mymod_[^,]*,?!d' \
-e 's/^\.*"(.*)".*/\1/' \
$^

mod.scm: mod.la mod.exports
$(gx) gen-scheme-wrapper \
-o $@ $< -g my -t init_~A \
-i '$(pkglibdir)'
```

5.2 Support modules

5.2.1 common

This module provides common procedures useful in a variety of contexts.

- fs** *s* [*args...*] [Procedure]
Apply `simple-format` to *s* and *args*, returning the result as a string.
- fso** *s* [*args...*] [Procedure]
Apply `simple-format` to *s* and *args*, sending the result to the current output port.
- fse** *s* [*args...*] [Procedure]
Apply `simple-format` to *s* and *args*, sending the result to the current error port.
- die** *exit-value* [*s* [*args...*]] [Procedure]
If *s* is specified, apply `fse` to *s* and *args*. Then `exit` with *exit-value*.
- check-hv** *args config* [*flags...*] [Procedure]
Check *args* (list of strings) for second element being ‘`--help`’ or ‘`--version`’. If found, display the respective information, using *config*, to `stdout` and then `exit` successfully. If not found, return `#f`. The recognized *config* keys are:
- package** A string describing program affiliation (for ‘`--version`’).
- version** A string, or a thunk that yields a string when called, to use instead of the default “VERSION UNKNOWN”. Output, depending on whether or not `package` is specified, is:
- ```
PROGRAM (PACKAGE) VERSION
PROGRAM VERSION
```
- where *program* is `(basename (car args))`.
- help** Either a (typically multi-line) string, a thunk that produces a string, or the symbol `commentary`, which means use `file-commentary` from module `(ice-9 documentation)` to obtain the string.

All strings are trimmed of leading and trailing whitespace.

Lastly, *flags* are zero or more symbols that further change the default behavior:

- `no-exit` means don't `exit`; instead, after doing output return `#t`.
- `v-before` means for `'--help'`, first do the output for `'--version'`.

`qop<-args args option-spec` [Procedure]

Do (`getopt-long args option-spec`), and return a procedure *qop* that encapsulates the result. You can then call *qop* in various ways:

(*qop* `#t`) Return the raw result of the `getopt-long` call.

(*qop* *key*)

Return the value associated with *key*, or `#f`.

As a special case, if *key* is the empty list, then return the (possibly empty) list of strings comprising the non-option *args*. Note that `getopt-long` stops processing *args* if it sees `'--'` (hyphen, hyphen); all elements following it are considered non-option.

(*qop* *key* *proc*)

If *key* has an associated value, call *proc* with the value and return its result. Otherwise, return `#f`. This is a shorthand for `(and=> (qop key) proc)`.

### 5.2.2 forms-from

This module provides procedures to `read` forms from various places, useful for configuration files and other small, well-defined data sets. See [Section 5.2.10 \[scheme-scanner\], page 26](#), for more fanciness.

`forms<-port port` [Procedure]

Return a list of forms `read` from *port*.

`forms<-file filename` [Procedure]

Return a list of forms `read` from file *filename*.

### 5.2.3 alist-from-plist

This module provides the procedure `alist<-plist`.

`alist<-plist plist` [Procedure]

Return *plist* as an association list with order reversed. For example:

```
(alist<-plist '(p1 v1 p2 v2))
⇒ ((p2 . v2) (p1 . v1))
```

### 5.2.4 elide-dot-dotdot

This module provides three procedures that operate on filename *components*, that is, parts between the `'/'` (slash) characters.

`filename-components string` [Procedure]

Return a list of filename components parsed from *string*. Components are delimited by `"/"`, which is discarded. Null string components are also discarded.

**filename-components-append** *ls* [Procedure]

Return a string composed by prefixing each element of *ls* with `"/`.

**elide-dot-dotdot** *abs-name need-trailing-sep?* [Procedure]

Return a new string made by removing file name components from string *abs-name* that are `.` (dot), as well as file name components followed by `..` (dotdot), along with the `..` itself; note that these simplifications are done without checking the resulting file names in the file system.

The returned string normally does not end in `/` (slash), aside from the case where it names the root directory `"/`. However, if *need-trailing-sep?* is non-`#f`, it does.

### 5.2.5 file-newer-than

**file-newer-than** *a b [component]* [Procedure]

Return `#t` if *a* is newer than *b*. Both *a* and *b* may be a filename (string) or an object returned from `stat`. If *a* does not exist, the answer is `#f`; otherwise, if *b* does not exist, the answer is `#t`.

Optional arg *component* specifies a procedure to use instead of the default `stat:mtime`. It should return a numeric value.

### 5.2.6 filenamez

This module provides two procedures that read NUL- (i.e., `#\nul`, `\0`) terminated filenames. Many programs produce such lists.

**read-filenamez** *port* [Procedure]

Read and return a list of NUL-terminated filenames from *port*.

**filenamez<-file** *filename* [Procedure]

Read and return a list of NUL-terminated filenames from input file *filename*.

### 5.2.7 frisker

**frisker** [*options...*] [Procedure]

Return a procedure *frisk* that takes a list of files. *options* is an alist. Recognized keys are:

**default-module**

The module to use as default module instead of `(guile-user)`.

*frisk* returns another procedure *report*, that takes one arg, *request* (a symbol), and returns:

|                 |                                                |
|-----------------|------------------------------------------------|
| <b>modules</b>  | entire list of modules                         |
| <b>internal</b> | list of internal modules                       |
| <b>external</b> | list of external modules                       |
| <b>i-up</b>     | list of modules upstream of internal modules   |
| <b>x-up</b>     | list of modules upstream of external modules   |
| <b>i-down</b>   | list of modules downstream of internal modules |
| <b>x-down</b>   | list of modules downstream of external modules |
| <b>edges</b>    | list of edges                                  |

Note that `x-up` will always return the empty list, since by (lack of) definition, we only know external modules by reference.

The module and edge objects managed by `report` can be examined in detail by using the other procedures described below. Be careful not to confuse a freshly consed list of symbols, like `(a b c)` with the *managed module* `(a b c)`. For this reason, `request` may also be a list of symbols (i.e., a module name), in which case `report` returns the managed module.

|                                                                                                                         |             |
|-------------------------------------------------------------------------------------------------------------------------|-------------|
| <code>mod-up-ls</code> <i>module</i>                                                                                    | [Procedure] |
| Return the upstream modules list of <i>module</i> .                                                                     |             |
| <code>mod-down-ls</code> <i>module</i>                                                                                  | [Procedure] |
| Return the downstream modules list of <i>module</i> .                                                                   |             |
| <code>mod-int?</code> <i>module</i>                                                                                     | [Procedure] |
| Return <code>#t</code> if <i>module</i> is <i>internal</i> (has a <code>define-module</code> form).                     |             |
| <code>edge-type</code> <i>edge</i>                                                                                      | [Procedure] |
| Return the symbolic type of <i>edge</i> , one of <code>regular</code> , <code>autoload</code> , <code>computed</code> . |             |
| <code>edge-up</code> <i>edge</i>                                                                                        | [Procedure] |
| Return the upstream-side module of <i>edge</i> .                                                                        |             |
| <code>edge-down</code> <i>edge</i>                                                                                      | [Procedure] |
| Return the downstream-side module of <i>edge</i> .                                                                      |             |

### 5.2.8 pke

|                                                                  |             |
|------------------------------------------------------------------|-------------|
| <code>pke</code> [ <i>x...</i> ]                                 | [Procedure] |
| Like <code>pk</code> , except output goes to current error port. |             |

### 5.2.9 read-string

|                                                       |             |
|-------------------------------------------------------|-------------|
| <code>read-string</code> <i>s</i>                     | [Procedure] |
| Return the object made from reading string <i>s</i> . |             |

### 5.2.10 scheme-scanner

This module provides the procedure `scheme-scanner`.

|                                                                                                                                                                                                                                                                                                                                                                                    |             |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| <code>scheme-scanner</code> [ <i>custom</i> ]                                                                                                                                                                                                                                                                                                                                      | [Procedure] |
| Return a procedure <i>scanner</i> that takes one argument, <i>port</i> . <i>scanner</i> reads a top-level element from <i>port</i> , returning a pair that describes it, or the symbol <code>eof</code> on end-of-file. In a description pair, the CAR is the element <i>type</i> , the CADR is its <i>location</i> and CDDR is an alist whose contents are <i>type</i> -specific. |             |
| <pre>(comment AT (leading-semicolons . N)            (text . LINE))  (whitespace AT (text . LINE))</pre>                                                                                                                                                                                                                                                                           |             |

```
(hash-bang-comment AT (line-count . N)
 (text-list LINE1 LINE2 ...))
```

```
(hash-bar-comment AT (text . TEXT))
```

```
;; custom form (see below)
```

```
(form AT (line-count . N)
 (sexp . SEXP))
```

*at* is a form (at *lno col beg end*); *line* is a string sans newline; *text* is a string that may contain newlines; *lno*, *col*, *beg*, *end* and *n* are integers; *sexp* is what (`read port`) returns.

*scanner* may throw `incomplete-hash-bang-comment` or `incomplete-hash-bar-comment` if end-of-file is encountered while scanning those respective comment types.

Optional arg *custom* is a procedure to be called right before recognizing the (default) form element. It takes one arg, *port*. If *custom* returns `#f`, *scanner* falls through to the default, else it returns what *custom* returns.

### 5.2.11 stemname

`stemname filename` [Procedure]

Return the `basename`, removing as well any extension (last ‘.’ through end) of *filename*. For example:

```
(stemname "a/b/c.d.e.f")
⇒ "c.d.e"
```

### 5.2.12 ts-base

This module provides procedures and constants used by the documentation munging programs. The “ts” is short for *texinfo snippet*. See [Section 5.1.6 \[tsar\]](#), page 12, See [Section 5.1.7 \[tsin\]](#), page 14.

#### 5.2.12.1 two-part filename

`split-filename filename` [Procedure]

Split *filename* into its directory and non-directory portions. Return a pair (*dir* . *non-dir*), where *dir* always ends with ‘/’ (slash).

`unsplit pair` [Procedure]

Return a new string made from appending the CAR and CDR of *pair* (both strings).

#### 5.2.12.2 texinfo snippet

The *texinfo snippet* record type—‘ts’ for short—has fields oriented towards procedure (and ilk, which “take arguments”) definitions, but capable of handling also variable (and ilk, which are values, and do not take arguments) definitions and *titled text blocks*. Fields are:

`name`            symbol or string

|                 |                                                                                                                                                                                                                                                                                                                                                                         |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>module</b>   | module name (i.e., list of symbols)                                                                                                                                                                                                                                                                                                                                     |
| <b>filename</b> | pair whose CAR is the <code>dirname</code> and CDR the <code>basename</code> of the file where the snippet originates                                                                                                                                                                                                                                                   |
| <b>blurb</b>    | snippet itself (string)                                                                                                                                                                                                                                                                                                                                                 |
| <b>category</b> | symbol, string, or <code>#f</code>                                                                                                                                                                                                                                                                                                                                      |
| <b>sig</b>      | (if arguments) either a proper list of symbols, or the vector <code> #(req opt var [name ...]) </code> , where <code>req</code> , <code>opt</code> and <code>var</code> are non-negative integers whose sum corresponds with the number of <code>name</code> symbols (or list of symbols aggregating all “keyword argument” names)<br>(if no arguments) <code>#f</code> |
| <b>at</b>       | vector <code> #(lno col beg end) </code> , where <code>lno</code> and <code>col</code> are line and column numbers, respectively, and <code>beg</code> and <code>end</code> are file positions (non-negative byte offsets) that delimit the documented form                                                                                                             |
| <b>options</b>  | alist of options scanned by ‘ <code>tsar</code> ’                                                                                                                                                                                                                                                                                                                       |

There is one constructor, an accessor for each field, and no modifier procedures.

`make-ts name module filename blurb category sig at options` [Procedure]  
Return a new texinfo snippet.

|                             |             |
|-----------------------------|-------------|
| <code>ts:name ts</code>     | [Procedure] |
| <code>ts:module ts</code>   | [Procedure] |
| <code>ts:filename ts</code> | [Procedure] |
| <code>ts:blurb ts</code>    | [Procedure] |
| <code>ts:category ts</code> | [Procedure] |
| <code>ts:sig ts</code>      | [Procedure] |
| <code>ts:at ts</code>       | [Procedure] |
| <code>ts:options ts</code>  | [Procedure] |

Return the specified field from texinfo snippet `ts`.

### 5.2.12.3 archive

The *archive* record type (‘`ar`’ for short) holds texinfo snippets and their metadata. Its contents are saved on disk.

|                |                                                                    |
|----------------|--------------------------------------------------------------------|
| <b>coding</b>  | symbol                                                             |
| <b>dirs</b>    | list of directory names, each ending with ‘/’ (slash)              |
| <b>files</b>   | list of pairs whose CAR is a directory and whose CDR is a basename |
| <b>modules</b> | list of module names (each a list of symbols)                      |
| <b>items</b>   | list of texinfo snippets                                           |

There is one constructor, an accessor for each field, and no modifier procedures.

`make-ar coding dirs files modules items` [Procedure]  
Return a new archive.

|                         |           |             |
|-------------------------|-----------|-------------|
| <code>ar:coding</code>  | <i>ar</i> | [Procedure] |
| <code>ar:dirs</code>    | <i>ar</i> | [Procedure] |
| <code>ar:files</code>   | <i>ar</i> | [Procedure] |
| <code>ar:modules</code> | <i>ar</i> | [Procedure] |
| <code>ar:items</code>   | <i>ar</i> | [Procedure] |

Return the specified field from archive *ar*.

#### 5.2.12.4 constant variables

|                         |                                                 |                    |
|-------------------------|-------------------------------------------------|--------------------|
| <code>MAGIC</code>      |                                                 | [Constant String]  |
|                         | The four-byte string " <code>^T^S^A^R</code> ". |                    |
| <code>FINISH</code>     |                                                 | [Constant String]  |
|                         | The two-byte string " <code>^\n</code> ".       |                    |
| <code>FINISH-LEN</code> |                                                 | [Constant Integer] |
|                         | The byte length of <code>FINISH</code> .        |                    |

#### 5.2.12.5 other procedures

|                           |                                                                                                                                                                                                             |             |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| <code>read-ar-file</code> | <i>bummer</i> <i>expected-coding-for-merge?</i> <i>filename</i>                                                                                                                                             | [Procedure] |
|                           | Read <i>filename</i> as a texinfo snippet archive. The file must declare its encoding to be <i>expected-coding</i> . On encoding mismatch or other error, call <i>bummer</i> with a format string and args. |             |
|                           | If <i>for-merge?</i> , return four values (each a list of): directories (string), files (split), modules (list of symbols) and items (texinfo snippet object).                                              |             |
|                           | Otherwise, return a single <code>ar</code> object, whose <code>modules</code> and <code>items</code> members are hash tables.                                                                               |             |

#### 5.2.13 ts-output

|                                        |                                                                                                                                                                                                                                                                                                                                                                              |             |
|----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| <code>extract-options-deleting!</code> | <i>ls</i>                                                                                                                                                                                                                                                                                                                                                                    | [Procedure] |
|                                        | Return an alist of options destructively extracted from <i>ls</i> , a list of strings, each ending in <code>#\newline</code> . An option line begins with <code>'-</code> (hyphen), followed by a Scheme symbol (the <i>key</i> ), <code>:'</code> (colon), and one or more spaces. The rest of the line (sans trailing whitespace) constitutes the <i>value</i> (a string). |             |
|                                        | Note that if the first element of <i>ls</i> is an option line, it will fail to be removed from <i>ls</i> .                                                                                                                                                                                                                                                                   |             |
| <code>ar&lt;-snippets</code>           | <i>coding</i> <i>snippets</i>                                                                                                                                                                                                                                                                                                                                                | [Procedure] |
|                                        | Return a new <code>ar</code> object with <i>coding</i> containing <i>snippets</i> , a list of ts objects. This is a wrapper for <code>make-ar</code> that ensures uniqueness of the filenames, modules and snippets.                                                                                                                                                         |             |
| <code>write-ar</code>                  | <i>ar</i> <i>port</i>                                                                                                                                                                                                                                                                                                                                                        | [Procedure] |
|                                        | Serialize <i>ar</i> to output <i>port</i> .                                                                                                                                                                                                                                                                                                                                  |             |

#### 5.2.14 write-string

|                           |                                                                               |             |
|---------------------------|-------------------------------------------------------------------------------|-------------|
| <code>write-string</code> | <i>s</i>                                                                      | [Procedure] |
|                           | Like <code>write</code> for string <i>s</i> , but with the following escapes: |             |

```

#\bel \a #\newline \n #\ht \t
#\np \f #\cr \r #\vt \v

```

Another difference is that *s* is sent unconditionally to the current output port.

### 5.2.15 temporary-file

**unlink-port-filename** *port* [*close?*] [Procedure]

Remove the filename associated with *port* from the filesystem. Optional arg *close?* non-#f means do `close-port` as well (prior to the actual unlink operation).

**unique-i/o-file-port** *base* [*suffix*] [Procedure]

Return a new i/o port opened on a file named using string *base*. The actual assigned filename, which can be retrieved using `port-filename`, is *base* appended with six random characters. Optional arg *suffix* is a string to append to the assigned filename. For example:

```

(define p (unique-i/o-file-port "/tmp/foo-" ".c"))
(port-filename p)
⇒ "/tmp/foo-hoQtXh.c"

```

Implementation/portability note: This procedure is a wrapper around `mkstemp` and `mkstemp!` (tried in that order).

**temporary-file** [Procedure]

Return an input/output port to a unique temporary file named using the directory prefix `P_tmpdir` defined in `'stdio.h'`. The file is automatically deleted when the port is closed or the program terminates. You can use the environment variable `'TMPDIR'` to override the default directory prefix.

Implementation/portability note: This procedure is a wrapper around `tmpfile` and `unique-i/o-file-port` (tried in that order), falling back to a “manual” implementation if all else fails. The filename portion of the prefix (i.e., the non-directory part) is unspecified for `tmpfile`; for the other approaches, this is computed as:

```

(or (car (command-line))
 "tmp")

```

### 5.2.16 a-dash-dash-b

**a-dash-dash-b** *args* [Procedure]

Return two values made by splitting *args*, a list of strings, on `'--'` (dash-dash), omitting it from the right-hand side.

```

(a-dash-dash-b '("prog" "arg" "--" "extra"))
⇒ ("prog" "arg")
⇒ ("extra")

```

## 6 Extending

Integrating Guile-BAUX into your project takes a bit of work (see [Chapter 4 \[Integration\]](#), [page 7](#)), but the good news is that this infrastructure lends itself to other scripts and modules specific to your project’s maintenance. Although you cannot use ‘guile-baux-tool’ to manage their dependencies<sup>1</sup>, it’s easy enough to add a file to subdirectory ‘guile-baux’ and modify the first few lines (aka *script header*), `define-module` form, and procedure `main` to interoperate well with those from upstream.

On the flip side, if your local script/module might be of general use, and you are willing to (re-)license it as GNU GPLv3+, why not contribute it upstream?

In the following description, as a running example, we build a small program ‘check-eol-ws’ for the project “foo”. This program exports a single procedure, as well.

### 6.1 script header, copyright notice

The script header is actually a small ‘/bin/sh’ dispatcher to the (runtime-selectable) Guile interpreter, specifying the entry point and the (pass-through) command-line arguments. The module name should start with `guile-baux`. You should always include a copyright notice.

```
#!/bin/sh
exec ${GUILLE-guile} -e '(guile-baux check-eol-ws)' -s $0 "$@"
!#
;;; check-eol-ws --- scan files for end-of-line whitespace

;;; Copyright (C) 2010 J. R. Hacker
;;;
;;; license-notice
```

### 6.2 text for --help

Guile provides procedure ([ice-9 documentation](#)) `file-commentary`, which Guile-BAUX wraps in `check-hv` (see [Section 5.2.1 \[common\]](#), [page 23](#)), so might as well use it.

```
;;; Commentary:

;;; Usage: check-eol-ws [FILE...]
;;;
;;; Check each FILE (or standard input if none specified)
;;; for end-of-line whitespace, displaying occurrences to
;;; standard output. Option --error (-e, for short) means
;;; exit failurefully on first occurrence.
;;;
;;; Usage from Scheme:
;;; (string-trailing-whitespace? STRING) => BOOL

;;; Code:
```

---

<sup>1</sup> yet?

We include the section “Usage from Scheme” because the exported procedure is not documented elsewhere. Usually, however, it is better to write texinfo documentation (as part of your project’s manual), so that the procedure can be indexed.

### 6.3 define-module form

The module name here must coincide with the one in the script header. We `#:export` first, perhaps out of R6RS envy... We always export `main` because some versions of Guile require the entry point to be public, and doing so doesn’t hurt those that don’t.

```
(define-module (guile-baux check-eol-ws)
 #:export (string-trailing-whitespace?
 main)
 #:use-module ((guile-baux common)
 #:select (fso check-hv qop<-args))
 #:use-module ((ice-9 rdelim)
 #:select (read-line)))
```

### 6.4 script body

The rest of the file defines the exported procedures and `main`, along with whatever support they require. It does not have any top-level (non-`define`) expressions. In this example, top-level forms are ordered topologically (definition before use) so that `main` is last, but that’s purely a matter of style.

The comment immediately prior to `string-trailing-whitespace?` is formatted for easy extraction (see [Section 5.1.6 \[tsar\], page 12](#)).

```
;; Return non-@code{#f} if @var{string}
;; ends with space, tab or carriage-return.
;;
(define string-trailing-whitespace?
 (let ((rx (make-regexp "[\\t\\r]$")))
 (lambda (string)
 (regexp-exec rx string))))

(define (scanner die?)
 (lambda (filename)
 (let ((p (open-input-file filename)))
 (let loop ((line (read-line p)) (lno 1))
 (define (more)
 (loop (read-line p) (1+ lno)))
 (cond ((eof-object? line)
 (close-port p)
 ((string-trailing-whitespace? line)
 (fso "~A:~A: trailing whitespace!~%"
 filename lno)
 (and die? (exit #f))
 (more))
 (else
```

```
(more))))))
```

## 6.5 main procedure

In `main`, we use `check-hv` and `qop<-args` (see [Section 5.2.1 \[common\]](#), page 23).

```
(define (main args)
 (check-hv args '((package . "foo")
 (version . "1.0")
 (help . commentary))
 'v-before)
 (let ((qop (qop<-args args '((error (single-char #\e))))))
 (for-each (scanner (qop 'error))
 (qop '()))))
```

## 6.6 scripts that do not export

In the preceding example, we did ‘`--help`’ and ‘`--version`’ checking in `main`, and generally avoided non-definition expressions in the script body, primarily so that the module (`guile-baux check-eol-ws`) can be used (via a reference) in another module’s `define-module` form.

For scripts that do not export anything, you can use a more relaxed style, freely mixing definitions and (side-effecting) expressions in the script body. Additionally, the script header need not specify an entry point. For example:

```
#!/bin/sh
exec ${GUILÉ-guile} -s $0 "$@"
!#
;;; count-parens --- count parentheses (or pairs of them)

;;; Copyright (C) 2010 J. R. Hacker
;;;
;;; license-notice

;;; Commentary:

;;; Usage: count-parens [--pairs | -p] [file...]
;;;
;;; Write count of parentheses in each FILE to stdout.
;;; Option --pairs reports (/ count 2), instead.

;;; Code:

(define-module (guile-baux count-parens)
 #:use-module ((guile-baux common)
 #:select (fso check-hv qop<-args)))

(define CL (command-line))
(check-hv CL '((package . "foo"))
```

```

 (version . "1.0")
 (help . commentary)))

(define (count denom)
 (lambda (filename)
 (let ((p (open-input-file filename)))
 (let loop ((n 0))
 (let ((c (read-char p)))
 (cond ((eof-object? c)
 (close-port p)
 (fso "~A: ~A~%" filename (/ n denom)))
 (else
 (loop (if (memq c '(#\ (#\)))
 (1+ n)
 n))))))))))

;; "main"
(let ((qop (qop<-args CL '((pairs (single-char #\p))))))
 (for-each (count (if (qop 'pairs) 2 1))
 (qop '())))

;;; count-parens ends here

```

## 7 Personal Use

Although the scope of Guile-BAUX is ostensibly oriented towards making life easier for the downstream users of your project, there is no harm (and indeed, some benefit) to slipping some of its functionality “sideways” for personal use. We are talking about freedom 0, after all<sup>1</sup>.

Practically, this translates into the question: How can I use Guile-BAUX program and support modules outside the scope of a particular project’s current installation?

In the following sections, we will assume Guile-BAUX lives in directory ‘\$HOME/build/guile-baux’, such that:

```
$ GBX=$HOME/build/guile-baux
$ export GBX
$ test -d $GBX/.git && echo ok
ok
```

You should substitute an appropriate value for `GBX`, of course.

### 7.1 support

There are two ways to use support modules: *as-is* and *re-prefixed*. Using a support module *as-is* is very simple. All you need to do is make sure that `GBX` is mentioned in `%load-path` somehow, e.g., by adding it to env var `GUILE_LOAD_PATH`. On the other hand, using the modules *re-prefixed* is more hairy; you need to:

- copy the module’s implementation to another place;
- edit its `define-module` form to change the first part of the module name from `guile-baux` to something appropriate to the new location;
- recurse likewise for all `#:use-modules` clauses that use a Guile-BAUX module;

and finally, make sure the new location is resolvable via `%load-path`. Luckily, the branchiness of the support modules is pretty low; degenerate recursion is not so tedious:

```
testing-lib: common
pke: common
frisker: common, scheme-scanner
ts-output: temporary-file, ts-base, common
```

(Standalone, i.e., no dependencies whatsoever: `alist-from-plist`, `read-string`, `stemname`, `write-string`, `forms-from`, `scheme-scanner`, `temporary-file`, `ts-base`, `common`, `a-dash-dash-b`, `file-newer-than`, `elide-dot-dotdot`, `filenamez`.)

To offset this extra work, a *re-prefixed* module has another property: you can modify it, for example, to remove bloat or improve an interface. (If the modification is a useful refactoring of some sort, please consider pushing it upstream.)

---

<sup>1</sup> <http://www.gnu.org/philosophy/free-sw.html>

## 7.2 program

In contrast to the support modules, program modules always have internal dependencies, so the work required for using them re-prefixed is non-trivial. (Anyway, it's possible — just use the technique outlined above.)

This section describes a couple ways to create a wrapper for the program modules for installation into `~/bin` (presumed to be the current working directory): the *one-off* wrapper and the *mux-like* wrapper.

The one-off wrapper is very simple. For example, say you want to use the Guile-BAUX `frisk` program (because `guile-tools frisk` under Guile 1.8.7 is deficient). Just create `~/bin/frisk` (don't forget to `chmod +x` it):

```
#!/bin/sh
exec $GBX/guile-baux/gbaux-do frisk "$@"
```

Henceforth, assuming `~/bin` is in env var `PATH`, you can invoke it directly, e.g., `frisk --help`. The mux-like wrapper is slightly more complicated. You create a wrapper for `gbaux-do`:

```
#!/bin/sh
program=$(basename $0)
exec $GBX/guile-baux/gbaux-do $program "$@"
```

then use that to dispatch to other programs:

```
$ for p in frisk c2x punify ; do ln -sf gbaux-do $p ; done
```

The advantage of the mux-like wrapper is that, should `GBX` change in the future, you need to update only one file and all the programs will work as before. In fact, we like this approach so much, there is support for it in the top-level `GNUmakefile`. The above can be done with:

```
make mux-like-wrapper f0-programs='frisk c2x punify'
```

Aside from `f0-programs` (default: empty), you can also specify `f0-bin` (default: `$(HOME)/bin`) and `f0-dispatch` (default: `'gbaux-do'`). If `f0-programs` is empty, then `'make mux-like-wrapper'` creates only the dispatch program.

# Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

<http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

## 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

## 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

### 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

### 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any,

- be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
  - C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
  - D. Preserve all the copyright notices of the Document.
  - E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
  - F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
  - G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
  - H. Include an unaltered copy of this License.
  - I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
  - J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
  - K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
  - L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
  - M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
  - N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
  - O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their

titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements.”

## 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

## 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

## 11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

## ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C) year your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts. A copy of the license is included in the section entitled ‘‘GNU
Free Documentation License’’.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

# Index

|                                               |    |                                             |    |
|-----------------------------------------------|----|---------------------------------------------|----|
| 'guile-baux-tool' command-line options .....  | 6  | filename-components .....                   | 24 |
| 'guile-baux-tool' commands .....              | 5  | filename-components-append .....            | 25 |
| 'guile-baux-tool' invocation filename .....   | 4  | filenamez<-file .....                       | 25 |
| -                                             |    | FINISH .....                                | 29 |
| '--dry-run' .....                             | 6  | FINISH-LEN .....                            | 29 |
| '--verbose' .....                             | 6  | focus, Guile-BAUX .....                     | 2  |
| A                                             |    | forms<-file .....                           | 24 |
| a-dash-dash-b .....                           | 30 | forms<-port .....                           | 24 |
| AC_CONFIG_AUX_DIR, from 'configure.ac' .....  | 2  | freedom 0 .....                             | 35 |
| alist<-plist .....                            | 24 | frisk, program .....                        | 11 |
| ar: coding .....                              | 29 | frisker .....                               | 25 |
| ar: dirs .....                                | 29 | fs .....                                    | 23 |
| ar: files .....                               | 29 | fse .....                                   | 23 |
| ar: items .....                               | 29 | fso .....                                   | 23 |
| ar: modules .....                             | 29 | G                                           |    |
| ar<-snippets .....                            | 29 | gen-scheme-wrapper, program .....           | 21 |
| as-C-byte-array, program .....                | 10 | gnulib .....                                | 2  |
| as-is support module usage .....              | 35 | H                                           |    |
| C                                             |    | help for program modules .....              | 5  |
| c .....                                       | 15 | history, 'guile-baux-tool' command .....    | 5  |
| C .....                                       | 15 | I                                           |    |
| c-tsar, program .....                         | 17 | i .....                                     | 15 |
| c2x, program .....                            | 20 | import, 'guile-baux-tool' command .....     | 5  |
| check-hv .....                                | 23 | inner-upstream, program .....               | 11 |
| command-line options, 'guile-baux-tool' ..... | 6  | interaction model .....                     | 2  |
| commands, 'guile-baux-tool' .....             | 5  | invoking '--help' for program modules ..... | 5  |
| D                                             |    | K                                           |    |
| definition line, output format .....          | 16 | keyword arguments, output formatting .....  | 16 |
| describe, 'guile-baux-tool' command .....     | 5  | kw-sort, flag for @tsin i .....             | 16 |
| describe-all, 'guile-baux-tool' command ..... | 5  | L                                           |    |
| die .....                                     | 23 | licensing .....                             | 3  |
| directives for 'tsin' .....                   | 15 | list, 'guile-baux-tool' command .....       | 5  |
| drop, 'guile-baux-tool' command .....         | 5  | local fixes to upstream .....               | 4  |
| E                                             |    | M                                           |    |
| edge-down .....                               | 26 | m .....                                     | 15 |
| edge-type .....                               | 26 | MAGIC .....                                 | 29 |
| edge-up .....                                 | 26 | make-ar .....                               | 28 |
| elide-dot-dotdot .....                        | 25 | make-ts .....                               | 28 |
| extract-options-deleting! .....               | 29 | mod-down-ls .....                           | 26 |
| F                                             |    | mod-int? .....                              | 26 |
| file-newer-than .....                         | 25 | mod-up-ls .....                             | 26 |
|                                               |    | mux-like program module usage .....         | 36 |

- mux-like-wrapper ('GNUmakefile' target)..... 36
- ## N
- no-kw-index, flag for @tsin i ..... 16  
 no-kw-list, flag for @tsin i ..... 16
- ## O
- one-off program module usage ..... 36  
 output formatting, texinfo snippet ..... 15  
 overview ..... 2
- ## P
- personal use ..... 35  
 pke ..... 26  
 portability ..... 2  
 punify, program ..... 11
- ## Q
- qop<-args ..... 24
- ## R
- re-prefixed support module usage ..... 35  
 re-prefixed-site-dirs, program ..... 10  
 read-ar-file ..... 29  
 read-filenamez ..... 25  
 read-string ..... 26  
 repository, upstream ..... 2  
 reset, 'guile-baux-tool' command ..... 6
- ## S
- scheme-scanner ..... 26  
 split-filename ..... 27
- status, 'guile-baux-tool' command ..... 5  
 stemname ..... 27
- ## T
- temporary-file ..... 30  
 titled text block, output format ..... 15  
 ts:at ..... 28  
 ts:blurb ..... 28  
 ts:category ..... 28  
 ts:filename ..... 28  
 ts:module ..... 28  
 ts:name ..... 28  
 ts:options ..... 28  
 ts:sig ..... 28  
 tsar, program ..... 12  
 tsin, program ..... 14
- ## U
- unique-i/o-file-port ..... 30  
 unlink-port-filename ..... 30  
 unsplit ..... 27  
 update, 'guile-baux-tool' command ..... 5  
 upstream, fixed locally ..... 4  
 upstream, initial clone ..... 4  
 upstream, update ..... 4
- ## V
- version number, Guile-BAUX ..... 5
- ## W
- write-ar ..... 29  
 write-line-punily ..... 12  
 write-punily ..... 12  
 write-string ..... 29