Theme-D User Guide

Tommi Höynälänmaa

May 27, 2026

Contents

1 Copyright

2 General

This guide covers only UNIX systems. The software has been tested in Debian and Ubuntu. Many of the commands in this guide have to be run as root. A root session is opened either with command `su root` or `sudo` depending on your system. In Ubuntu the command is `sudo`.

Symbol *rev* in the package names means the Debian revision of the packages. It is typically 1 or 2. Symbol *arch* means the host system architecture, which can be obtained by command

```
dpkg-architecture -q DEB_HOST_ARCH
```

The newest version of the software is built for Guile version 3.0.11. It is possible to build Theme-D for earlier versions of Guile starting from 3.0.8. It is also possible to build the software for Guile 3.0.7 but then you have to edit some debianization files, see section 4.1.

3 Installation

3.1 Debian forky (testing) and sid (unstable) and Ubuntu Stonking

If you use Synaptic Package Manager install the following packages:

- `theme-d-rte`
- `theme-d-translator`
- `theme-d-stdlib`

If you also want to have the documentation install package `theme-d-doc`, too. The bootstrapped Theme-D system is contained in package `theme-d-bootstrap`. In order to install the system from the command line give the following command:

```
sudo apt-get install theme-d-rte theme-d-translator theme-d-stdlib
```

and optionally one or both of the commands

```
sudo apt-get install theme-d-doc
sudo apt-get install theme-d-bootstrap
```

3.2 Debian trixie (stable) and bookworm (oldstable) and Ubuntu Focal, Jammy, Noble, Questing, and Resolute

If you are satisfied with an older version of Theme-D follow the instructions in section 3.1. Otherwise install Guile 3.0 with command

```
sudo apt install guile-3.0
```

Then follow section 4.1 to build and install Theme-D.

3.3 Debian bullseye (oldoldstable)

If you are satisfied with an older version of Theme-D follow the instructions in section 3.1. Otherwise

1. Install Guile 3.0 version 3.0.8-2 to your system: Get files `guile-3.0_3.0.8.orig.tar.xz` and `guile-3.0_3.0.8-2.debian.tar.xz` from

   https://packages.debian.org/bookworm/guile-3.0.

   Give commands

   ```
   sudo apt install build-essential gperf devscripts
   sudo apt-get build-dep guile-3.0
   ```

   Create a new directory and copy the downloaded files there. Change the working directory to the new directory and give commands

   ```
   tar xvf guile-3.0_3.0.8.orig.tar.xz
   cd guile-3.0-3.0.8.orig
   tar xvf ../guile-3.0_3.0.8-2.debian.tar.xz
   debuild -i -us -uc -b
   cd ..
   sudo dpkg -i guile-3.0-libs_3.0.8-2_arch.deb
   sudo dpkg -i guile-3.0_3.0.8-2_arch.deb
   sudo dpkg -i guile-3.0-dev_3.0.8-2_arch.deb
   sudo dpkg -i guile-3.0-doc_3.0.8-2_all.deb
   ```

   where the last command is optional.

2. Then follow section 4.1 to build and install Theme-D.

3.4 Other UNIX Systems

Follow the instructions in the next section.

4 Building

4.1 Debian-based Systems

These instructions apply to Debian-based Linux distributions such as Debian and Ubuntu. You need Guile 3.0 version >= 3.0.7.

The default directory configuration of Theme-D is stored in file `/etc/theme-d-config`. You may override this by defining environment variable `THEME_D_CONFIG_FILE` to

be the path of your own configuration file. The root directory of the Theme-D installation shall be called *theme-d-root-dir*. By default this is `/usr/share/theme-d` in Debian-based installations and `/usr/local/share/theme-d` in other installations.

Build and install Theme-D with the following steps:

1. Check if package `guile-3.0-dev` is installed with command

   ```
   dpkg -s guile-3.0-dev
   ```

   If you don't have it install it with command

   ```
   sudo apt-get install guile-3.0-dev
   ```

2. If your home directory contains file `~/.theme-d-config` delete the file.
3. Change to the directory where you want to unpack the Theme-D source code.
4. Copy files `theme-d-7.3.3.tar.xz` and `theme-d_7.3.3-`*rev*`.debian.tar.xz` into that directory.
5. Unpack Theme-D source code with command

   ```
   tar xvf theme-d-7.3.3.tar.xz
   ```

6. Change to the subdirectory `theme-d-7.3.3`.
7. Give command

   ```
   tar xvf ../theme-d_7.3.3-rev.debian.tar.xz
   ```

8. If you use Guile 3.0.7 change `GUILE_VERSION2` from `3.0.8` to `3.0.7` in `debian/rules` and conditions `(>= 3.0.8)` to `(>= 3.0.7)` in `debian/control`.
9. Give commands

   ```
   unset GUILE_LOAD_PATH
   unset GUILE_LOAD_COMPILED_PATH
   dpkg-buildpackage -b --no-sign
   cd ..
   dpkg -i th-scheme-utilities_7.3.3-rev_arch.deb
   dpkg -i libthemedsupport_7.3.3-rev_arch.deb
   ```

```
dpkg -i theme-d-rte_7.3.3-rev_arch.deb
dpkg -i theme-d-translator_7.3.3-rev_arch.deb
dpkg -i theme-d-stdlib_7.3.3-rev_all.deb
```

where *arch* is the name of your processor architecture. These commands have to be run as root.

10. If you want to install the Theme-D documentation give command

```
dpkg -i theme-d-doc_7.3.3-rev_all.deb
```

as root.

11. If you want to install the Theme-D bootstrapped environment give command

```
dpkg -i theme-d-bootstrap_7.3.3-rev_all.deb
```

as root.

4.2 Other UNIX Systems

1. If your home directory contains file ∼/.theme-d-config delete the file.
2. Install Guile 3.0 if you don't have it already. Check the version of the Guile development environment with command

```
pkg-config --modversion guile-3.0
```

See http://www.gnu.org/software/guile/.

3. Create some directory and unpack Theme-D package there with command

```
tar xvf theme-package-path/theme-d-7.3.3.tar.xz
```

The subdirectory theme-d-7.3.3 of the directory where you unpacked Theme-D shall be called *theme-d-source-dir*.

4. Give commands

```
unset GUILE_LOAD_PATH
unset GUILE_LOAD_COMPILED_PATH
```

In case you don't use a sh compatible shell these commands may be different or you may just ignore them.

5. Change to the the subdirectory *theme-d-source-dir*.

6. Give command

```
./configure
```

You may give the following options to command `./configure`:

- `--with-guile=`*version* : Specify the Guile version explicitly. Currently only version 3.0 is supported.
- `--with-guile-program=`*file* : Specify the Guile program used by the software explicitly. The default is `/usr/bin/guile-`*version*.
- `--with-guile-header-dir=`*directory* : Specify the directory where to find header file `libguile.h` for `libthemesupport` C compilation. The default is not to specify the directory explicitly.
- `--with-extension-dir=`*directory* : Specify the directory where to install Guile extensions.
- `--with-guile-module-dir=`*directory* : Specify the directory where to install Guile modules.
- `--with-guile-comp-module-dir=`*directory* : Specify the directory where to install compiled Guile modules.
- `--with-conf-dir=`*directory* : Specify the directory where to install the global Theme-D configuration file. Default is `/etc`.
- `--disable-xlat-opt-compilation` : Use Guile optimization level 1 for compiling the Theme-D translator.
- `--disable-rte-opt-compilation` : Use Guile optimization level 1 for the runtime environment compilation.
- `--without-support-library` : Don't use the `libthemedsupport` library.
- `--disable-extra-math` : Don't include the `(standard-library extra-math)` module in your installation.
- `--disable-posix-math` : Don't include the `(standard-library posix-math)` module in your installation.

If you use option `--without-support-library` option you also have to use options `--disable-extra-math` and `--disable-posix-math`.

7. Change to the the subdirectory *theme-d-source-dir* and give command

```
make
```

in order to prepare the code for installation. Install Theme-D by giving command

```
make install-complete
```

as root.

4.3 Using the Software without Installation

This sofware may also be used without installing it. This is useful if you develop Theme-D itself.

1. Install Guile 3.0 in case you do not have it already. See

   http://www.gnu.org/software/guile/

2. Create some directory and unpack Theme-D package there with command

   `tar xvf` *theme-package-path*`/theme-d-7.3.3.tar.xz`

3. Go into the the subdirectory `theme-d-7.3.3` of the directory created in the previous step. Give commands

   ```
   ./configure
   make
   ```

   See section 4.2 for the configure options.

In order to use Theme-D change to the subdirectory `meta` and give command

```
./uninstalled-env bash
```

Now the commands `theme-d-compile`, `theme-d-link`, and `run-theme-d-program` are available for you.

5 Removing the Software

5.1 Debian-based Systems

Give commands

```
dpkg --purge theme-d-stdlib
dpkg --purge theme-d-translator
dpkg --purge theme-d-rte
dpkg --purge libthemedsupport
dpkg --purge th-scheme-utilities
```

as root. In order to remove the Theme-D documentation give command

```
dpkg --purge theme-d-doc
```

as root. The bootstrapped environment can be removed with command

```
dpkg --purge theme-d-bootstrap
```

5.2 Other Systems

Give command

```
make uninstall-complete
```

as root in directory *theme-d-source-dir*.

6 File Extensions

Theme-D source files have the following extensions:

- `.thp` for proper programs
- `.ths` for scripts
- `.thi` for interfaces
- `.thb` for bodies

Theme-D compiled pseudocode files have the following extensions:

- `.tcp` for proper programs
- `.tcs` for scripts
- `.tci` for interfaces
- `.tcb` for bodies

The auxiliary module files use extension `.aux`.

7 Unit Root Directories

When you define a unit with full name

(*dir-1* `...` *dir-n* *unit-name*)

the module must have filename *unit-name* with proper extension (see the previous section) and it must be located in subdirectory

dir-1`/.../`*dir-n*`/`

of some directory *unit-root-dir*. The directory *unit-root-dir* is called a *unit root*

directory. If a unit name has only one component you may omit the parentheses from the unit name. When you compile of link a Theme-D unit you must specify one or more unit root directories where the imported modules are searched. These are called the *module search directories*. You should always have directory *theme-d-root-dir*`/theme-d-code` among the module search directories so that the standard libraries are found by the compiler and by the linker.

8 Compiling a Theme-D Unit

Give command

`theme-d-compile` *options unit-name*

where *unit-name* is the file name of the Theme-D unit. Options are

- `--module-path=` *paths* or `-m` *paths* : Module search paths separated with :'s
- `--output=` *output-filename* or `-o` *output-filename* : The output filename
- `--unit-type=` *unit-type* or `-u` *unit-type* : The unit type (`proper-program`, `script`, `interface`, or `body`)
- `--message-level=` *message-level* or `-l` *message-level* : Compiler message level, integer number from 0 to 3.
- `--expand-only` : Do only macro expansion on the source.
- `--no-expansion` : Compile the source without macro expansion.
- `--backtrace` : Print backtrace on compilation error.
- `--pretty-print` : Pretty print the pseudocode output.
- `--no-verbose-errors` : Less information in the error messages.
- `--show-modules` : Show information about loading modules.
- `--version` : Show Theme-D version number and exit.

By default the unit type is computed from the source file extension. The default module search path is *theme-d-root-dir*`:..` If you use option `-m` you may include the Theme-D default module search path in your custom path by adding an extra ":" in the beginning of the new path, e.g. `:my-path1:my-path2`. The default target file path is obtained by removing the path and the extension from the source filename and appending the appropriate extension to the result. The default message level is 1. Message level 0 means no output at all except in case of error. Message level 1 displays also message on successful compilation or linking. Message level 2 displays some debug information and level 3 a lot of debug information. When `--expand-only` is set the default target filename is `myunit.expanded.thx` for source file `myunit.thx`.

Suppose that you have your own Theme-D code at directory *my-theme-d-dir* and you have a program called `(mod-1 ... mod-n)` at location

```
mod-1/.../mod-n.thp
```

In order to compile the program give commands

```
cd my-theme-d-dir
theme-d-compile mod-1/.../mod-n.thp
```

Suppose that you have a module (an interface and a body) with name `(mod-1 ... mod-n)` in files `mod-1/.../mod-n.thi` and `mod-1/.../mod-n.thb`. In order to compile the module give commands

```
cd my-theme-d-dir
theme-d-compile mod-1/.../mod-n.thi
theme-d-compile mod-1/.../mod-n.thb
```

If you want to have the compiled files in the same subdirectory where the source files are, which is usually the case, give commands

```
cd my-theme-d-dir
theme-d-compile -o mod-1/.../mod-n.tci \
  mod-1/.../mod-n.thi
theme-d-compile -o mod-1/.../mod-n.tcb \
  mod-1/.../mod-n.thb
```

If you use Theme-D without installing it you have to use command

```
MYPATH/theme-d-VERSION/theme-d/translator/theme-d-compile.scm
```

instead of `theme-d-compile`. Here `MYPATH` is the path where you have unpacked Theme-D.

9 Linking a Theme-D Unit

A Theme-D unit can be linked either monolithically or modularly. In monolithic linking a Theme-D program is linked into a single Guile bytecode file. In modular linking Theme-D units are linked to separate Guile bytecode modules. For a Theme-D module `MYMODULE` the target interface module is named `__intf_MYMODULE.go` and the target body module `__impl_MYMODULE.go`. The auxiliary module files are named `__intf_MYMODULE.aux` and `__impl_MYMODULE.aux`. Theme-D dynamical plugin features can only be used with modular linking,

In order to link a Theme-D program monolithically give command

`theme-d-link` *options program-name*

where *program-name* is the file name of the Theme-D program. In order to link

a Theme-D unit to a Guile module give command

`theme-d-link` *options* *program-name*

where *program-name* is the file name of the Theme-D unit and *options* contains `--module`. Available options are

- `--module-path=` *paths* or `-m` *paths* : Module search paths separated with :'s
- `--guile-target-path=` *paths* : Guile target module search paths separated with :'s
- `--empty-guile-target-path` : Set Guile target module search path to be empty.
- `--full-module-path=` *paths* or `-M` *paths* : Equivalent to

  `--module-path=`*paths* `--guile-target-path=`*paths*`.`

- `--unit-type=` *unit-type* : Specify the unit type explicitly. Argument *unit-type* has to be one of `proper-program`, `script`, `body`, or `interface`.
- `--output=` *output-filename* or `-o` *output-filename* : The output filename.
- `--intermediate-file=` *filename* or `-n` *filename* : The intermediate filename.
- `--intermediate-language=` *language* or `-i` *language* : The language used for the intermediate file.
- `-x` *module*: Link (load) the module into the target program.
- `--message-level=` *message-level* or `-l` *message-level* : Linker message level, integer number from 0 to 3.
- `--no-final-compilation` : Do not compile the linker result file with `guild compile`.
- `--no-strip` : Do not strip away unused code.
- `--no-optimization` : Do not optimize linker output.
- `--no-factorization` : Do not factorize the type expressions out of procedure implementations.
- `--no-weak-assertions` : Do not check ordinary assertions. Strong assertions are always checked.
- `--backtrace` : Print backtrace on linking error.
- `--pretty-print` : Pretty print the linker output.

- `--no-verbose-errors` : Less information in the error messages.
- `--keep-intermediate` : Keep the intermediate Tree-IL or Scheme file on successful linking.
- `--no-deletion` : Keep the intermediate Tree-IL or Scheme file on linker error.
- `--link-to-cache` : Link the target file into the Guile cache.
- `--runtime-pretty-backtrace` : Generate the code to support runtime pretty printed backtraces.
- `--no-unlinked-procedure-names` : Do not generate code for reporting unlinked procedure names.
- `--module-debug-output` : Print debug messages when a module body linkage is started and ended.
- `--show-inst-number` : Print the expression numbers of the processed expressions in parametrized type instantiation.
- `--check-all-primitives` : Check that primitive procedure result values match the result types for all primitives, including those defined with `unchecked-prim-proc`.
- `--duplicates=` *symbols* : Set the values passed to `default-duplicate-binding-handler` in the target program. If there are several symbols enclose them in quotes.
- `--split` : Split the linker output.
- `--split-dir=` *dir* : Set the directory where to put the split linker output.
- `--split-basename=` *name* : Set the basename for split linker output files.
- `--guile-opt-level=` *level* : Set the optimization level for the final Guile compilation. The default is 1.
- `--extra-guild-options=` *options* : Define the extra options passed to `guild` when compiling the intermediate code to Guile bytecode.
- `--plugin` : Link a module body to a plugin. This option implies `--module`.
- `--version` : Show Theme-D version number and exit.

The available intermediate languages (backends) are:

- `tree-il-3.0` : Guile 3.0 Tree-IL.
- `guile-3.0` : Guile Scheme 3.0.

You may use aliases `tree-il` and `guile`. Using intermediate language `guile0` is equivalent to options `-i guile --no-optimization`. The option `--no-optimization` has no effect for the Tree-IL target platform. It is always optimized. By default Theme-D linker produces a Guile objcode file. Actually, Theme-D makes a Guile Tree-IL or Scheme file and uses Guile to make an objcode file from that. The default intermediate language is Tree-IL. Note that many optimizations are

performed only with Tree-IL. If you want to optimize your code for speed you should link your program without pretty backtraces when you no longer need them for debugging. If you use Tree-IL as the intermediate language pretty printing may cause the linker to crash with large programs. The syntax of the module name in the `-x` option is `"(mod1 ... modn)"`. Warning: if you use the Guile intermediate language toplevel expressions with side effects may be executed during the final compilation phase of linking.

If you use option `--module-path` or `-m` you may include the Theme-D default module search path in your custom path by an extra ":" in the path as in compilation. Suppose that you have your own Theme-D code at directory *my-theme-d-dir* and you have a program called `(mod-1 ... mod-n)` at location `mod-1/.../mod-n.thp`. In order to link the program give commands

```
cd my-theme-d-dir
theme-d-link mod-1/.../mod-n.thp
```

The previous commands place the linked file into the root of subdirectory *my-theme-d-dir*. If you want to place the linked file in the same directory where the source files are use the following commands:

```
cd my-theme-d-dir
theme-d-link -o mod-1/.../mod-n.go \
  mod-1/.../mod-n.thp
```

If you have so big program that your system hangs with it it is useful to split the linker output to several intermediate files. You can do this by giving option `--split` to the linker. The linker output files are placed on a separate subdirectory. By default this subdirectory is called *program*`.compiled`. You can change the directory name with option `--split-dir`. You can also change the basename of the output files with option `--split-basename`. Note that script `run-split-theme-d-program` does not work if you change the basename.

If option `--no-final-compilation` is not given the Tree-IL or Scheme file generated by the linker is compiled to Guile bytecode with command `guild compile`. Option `--guile-opt-level` specifies the optimization level of the final Guile compilation. Option `-O`*level* is passed to program `guild`. The default optimization level is 1. Note that invoking the Guile optimization of `letrec` expressions requires the optimization level to be at least 2. The Guile target path is used to set the value of environment variable `GUILE_LOAD_COMPILED_PATH` to the `guild` command.

10 Running a Theme-D Program

When you use Guile as the target platform Theme-D programs can be run with command

`run-theme-d-program` *metaarg* `...` *programfile programarg* `...`

where *metaarg* are the arguments passed to the script `run-theme-d-program`, *programfile* is the filename of the linked Theme-D program, and *programarg* are the arguments passed to the program. Suppose you have your linked Theme-D program in file `myprog.go`. You can run this program with command

```
run-theme-d-program myprog.go
```

When you use Guile as the target platform it is also possible to link you Theme-D program into a `.scm` intermediate file and run it with command

`guile -e main -s` *programfile*`.scm` *programarg* `...`

or

`guile -s` *programfile*`.scm` *programarg* `...`

for scripts.

If you need to import your own Scheme files into the Theme-D runtime environment (because of the foreign function interface) you can do this by defining the environment variable `THEME_D_CUSTOM_CODE`. Separate the file names with :'s. However, it is recommended to use option `-x` for this.

The program `run-theme-d-program` accepts the following arguments:

- `--no-verbose-errors` : No verbose information about errors (exceptions).
- `--backtrace` : Display backtrace on error.
- `--pretty-backtrace` : Display pretty printed backtrace on error.
- `--version` : Show Theme-D version number and exit.

Note that the `--pretty-backtrace` option works only if you have linked your Theme-D program with option `--runtime-pretty-backtrace`.

In order to run a Theme-D program with split linker output give command

`run-split-theme-d-program` *dir-name*

where *dir-name* is the directory where the linker output is generated.

In order to run a modularly linked Theme-D program give command

`run-theme-d-program-m -g` *guile-target-path* *program*

where *program* is a `.go` file created by the linker and *guile-target-path* is the path used to search Guile modules. Normally you should use path *root-dir*`:` where *root-dir* is the unit root directory of your program. Note that you don't need to include Guile cache directories into the Guile target path.

The pretty printed runtime backtrace has the following format:

number kind name module

⋮

where *kind* is the kind of the called procedure, *name* is the name of the procedure and *module* is the module where the procedure has been defined. The *kind* may take the following values:

- `toplevel`: A toplevel procedure
- `local`: A local procedure
- `instance`: An instance of a parametrized procedure
- `zero`: A procedure used to generate the zero value of a class

11 Theme-D Configuration File

The Theme-D configuration file is searched according to the following rules:

- Use the value of environment variable `THEME_D_CONFIG_FILE` is it is defined.
- Use file `.theme-d-config` in the user's home directory if present.
- Otherwise use file `/etc/theme-d-config`.

The installation procedure sets up the configuration file. Normally you don't have to edit it.

The configuration file has the following format:

```
(theme-d (var-name var-value)...)
```

All string type variable values must be enclosed in quotes. Boolean and integer values must not be enclosed in quotes The variables defined in the configuration file are:

- `guile-version`: The Guile version used by Theme-D. This is a string.
- `translator-dir`: The location of the compiler and linker implementations.
- `runtime-dir`: The location of the Theme-D runtime environment.
- `lib-dir`: The location of the Theme-D standard library.
- `examples-dir`: The location of the Theme-D examples.
- `tests-dir`: The location of the Theme-D tests.
- `tools-dir`: The location of the Theme-D tools.
- `bootstrap-dir`: The location of the Theme-D bootstrap environment sources.

- `compiler-path` Theme-D compiler path (a `.scm` file).
- `linker-path` Theme-D linker path (a `.scm` file).
- `run-path` Theme-D run script path (a `.scm` file).
- `use-support-lib?`: `#t` if the support library is used. This is a boolean value.

The values of the configuration variables can be fetched with command

`get-theme-d-config-var` *config-var-name*

where *config-var-name* is the name of the configuration variable.

12 Distributing Linked Theme-D Programs

If your target environment has Theme-D installed it is sufficient to distribute only the linked `.go` file. If your target system is using a Debian-based Linux system (e.g. Debian or Ubuntu) and you don't want to install whole Theme-D into it the easiest way to ensure that all the necessary files are present is to install packages `theme-d-rte`, `th-scheme-utilities`, and `libthemedsupport` into the target system.

If you are using Guile in a non-Debian system you have to ensure that the following files are present in the Guile library path:

- `th-scheme-utilities/*.go`
- `theme-d/common/theme-d-config.go`
- `theme-d/runtime/params.go`
- `theme-d/runtime/runtime-theme-d-environment.go`
- `theme-d/runtime/theme-d-stdlib-support.go`
- `theme-d/runtime/rte*.go`

You also need to distribute one of the following files:

- `theme-d/runtime/theme-d-support-all.go`
- `theme-d/runtime/theme-d-support-no-extra.go`
- `theme-d/runtime/theme-d-support-no-posix.go`
- `theme-d/runtime/theme-d-alt-support.go`

and create symbolic link `theme-d/runtime/theme-d-support.go` pointing to it. If your program uses modular linking you also have to distribute files `standard-library/__intf_*.go`, `standard-library/__intf_*.aux`, `standard-library/__impl_*.go`, and `standard-library/__impl_*.aux`.

In order to find the library path give command

`pkg-config --variable=siteccachedir guile-`*version*

Normally you should use file `theme-d-support-all.go`. If you don't use the Theme-D support library you must use `theme-d-alt-support.go`. If you distribute a `.go` file you also need to have `run-theme-d-program.scm`, `run-theme-d-program-m.scm`, or `run-split-theme-d-program.scm` in the target system. The choice between these files depends on how your program has been linked. The files you need to distribute are licensed under GNU Lesser General Public License.

If you use the support library the library `libthemedsupport` has to be installed in the target system. The use of the support library is recommended. In order to access the `*.go` mentioned in this section you have to build the Theme-D source package.

13 Bootstrapping Theme-D

The Theme-D source package and Debian package `theme-d-bootstrap` contain a bootstrapped version of Theme-D, i.e. Theme-D compiler and linker implemented with Theme-D itself. To install the bootstrap environment change to the directory where you want to install it and give command

```
setup-theme-d-bootstrap-env
```

Note that you must have either Theme-D installed on your system or use the uninstalled version of the software, see section 4.3.

First you have to build Theme-D written in itself using compiler and linker written in Guile. Change to the `theme-d-bootstrap` directory and give the following commands:

```
cd build1/theme-d-in-theme-d
make -f user.mk
```

or if you want to use modular linking

```
cd build1/theme-d-in-theme-d
LINK_MODULES=1 make -f user.mk
```

Then you build Theme-Dusing the compiler and linker built in the previous step:

```
cd ../../bootstrap/theme-d-in-theme-d
make -f user.mk
```

or if you want to use modular linking

```
cd ../../bootstrap/theme-d-in-theme-d
LINK_MODULES=1 make -f user.mk
```

If you use modular linking with `build1` you have to define variable `MODULAR_TRANSLATOR` with `bootstrap`, e.g.

```
cd ../../bootstrap/theme-d-in-theme-d
MODULAR_TRANSLATOR=1 LINK_MODULES=1 make -f user.mk
```

Now you have the bootstrapped Theme-D compiler and linker in files `theme-d-compile-b.go` and `theme-d-link-b.go` in subdirectory `theme-d-bootstrap/bootstrap/theme-d-in-theme-d`. We denote the path to this directory by `BOOTSTRAPPATH`. You can use these programs with commands

```
run-theme-d-program BOOTSTRAPPATH/theme-d-compile-b.go ARGUMENTS
```

and

```
run-theme-d-program BOOTSTRAPPATH/theme-d-link-b.go ARGUMENTS
```

or if you use modular linking

```
run-theme-d-program-m -g BOOTSTRAPPATH2:  \
   BOOTSTRAPPATH2/theme-d-in-theme-d/theme-d-compile-b.go ARGUMENTS
```

and

```
run-theme-d-program-m -g BOOTSTRAPPATH2:  \
   BOOTSTRAPPATH2/theme-d-in-theme-d/theme-d-link-b.go ARGUMENTS
```

where `BOOTSTRAPPATH2` is the path to directory `bootstrap`.

14 Compiling, Linking, and Running Test and Example Programs

In order to install the Theme-D testing environment change to the directory where you want the environment to be installed and give command

```
setup-theme-d-test-env
```

This directory shall be called *theme-d-test-dir* in the sequel. The test programs are located in subdirectory `test-env/theme-d-code/tests` and the example programs in `test-env/theme-d-code/examples`. Subdirectory `tools` contains scripts to run tests.

The example programs are built by giving command `make -f user.mk` in subdirectory `test-env/theme-d-code/examples`. If you want to use modular linking use command

```
LINK_MODULES=1 make -f user.mk
```

The example programs are run with command `run-theme-d-program` *program*`.go`.

If `testX` is a program compile it with command

```
theme-d-compile -m ..:  testX.thp
```

and link with command

```
theme-d-link -m ..:  testX.tcp
```

or

```
theme-d-link --module -M ..:  testX.tcp
```

in directory *theme-d-test-dir*`/test-env/theme-d-code/tests`.

If `testX` is a module compile it with commands

```
theme-d-compile -m ..:  testX.thi
theme-d-compile -m ..:  testX.thb
```

in directory *theme-d-test-dir*`/test-env/theme-d-code/tests`. If you use modular linking you also have to link the units with commands

```
theme-d-link --module -M ..:  testX.tci
theme-d-link --module -M ..:  testX.tcb
```

Note that some test programs import test modules in which case you must compile the modules before the program that uses them. When a test program imports several test modules compile first all the interfaces of the imported modules and then all the bodies of the imported modules. Compile the interfaces in the order they are numbered. Note also that some test programs require the examples to be built.

In order to run a test `testX` give commands

```
run-theme-d-program testX.go
```

or

```
run-theme-d-program-m -g ..:  testX.go
```

in directory *theme-d-test-dir*`/test-env/theme-d-code/tests`.

If you want to build all the tests at once build the examples first. Then change to the directory *theme-d-test-dir*`/test-env/testing`. Compile the tests

with command

```
./compile-tests.scm
```

Then you can link the programs monolithically with command

```
./link-test-programs.scm
```

or link all units modularly with command

```
./link-to-modules.scm
```

Then run the linked programs with command

```
./run-test-programs.scm
```

in case of monolithic linking and

```
./run-test-programs-m.scm
```

in case of modular linking. The compilation results can be checked with command

```
./check-test-compilation.scm
```

Results of monolithic linking and running can be checked with

```
./check-test-program-linking.scm
./check-test-runs.scm
```

and for modular linking and running with

```
./check-test-module-linking.scm
./check-test-runs-m.scm
```

All these scripts are located in directory *theme-d-test-dir*`/testing`.

You can generate the test output into the subdirectory `output` with command

```
./run-test-programs-w-output.scm
```

or

```
./run-test-programs-w-output-m.scm
```

in case of modular linking. Use command `./compare-output.sh` to compare

the output files with the correct ones. The correct outputs of the tests can be found in subdirectory `tests` in files `test*.out`. The outputs of tests `test450` and `test756` may vary because of output buffering. The computed hash values in test `test587` and the order of elements in hash tables in tests `test826` and `test827` may also vary. The backtraces in `test579` and `test764` and the path in `test598` may be different in different runs. The values of environment variable `HOME` printed by `test820` are different in different systems. For split linking, the output of `test115` is different because of different program name. The internal variables of `test472` may be different in different linker backends and linking styles (monolithic, modular, or split). For modular linking, different outputs are reported for test cases `test132`, `test135`, `test136`, `test173`, and `test940` due to different internal variable names. Monolithic linking with Guile0 backend and split linking may give different outputs for test cases `test132` and `test173` for the same reason. The Tree-IL backend with split linking generates backtraces for scripts `test606`, `test607`, `test716`, `test717`, and `test720`. Other backends with split linking do not generate backends for these scripts because of the way Guile handles side effects in module importing.

If you want to build the examples with the bootstrapped compiler and linker set the environment variables `THEME_D_COMPILE` to

```
run-theme-d-program MYPATH/theme-d-in-theme-d/theme-d-compile-b.go
```

and `THEME_D_LINK` to

```
run-theme-d-program MYPATH/theme-d-in-theme-d/theme-d-link-b.go
```

or in case of modular linking to

```
run-theme-d-program-m -g MYPATH \
  MYPATH/theme-d-in-theme-d/theme-d-compile-b.go
```

and

```
run-theme-d-program-m -g MYPATH \
  MYPATH/theme-d-in-theme-d/theme-d-link-b.go
```

Here `MYPATH` is either directory `build1` or `bootstrap` in the bootstrapped environment. If you want to use the bootstrapped compiler give option `-b MYPATH/theme-d-in-theme-d/theme-d-compile-b.go` or `-B MYPATH` to command `compile-tests.scm`. If your bootstrapper compiler uses split linking give option `-s MYPATH/theme-d-in-theme-d/theme-d-compile-b.build`.

Programs `compile-tests.scm`, `link-test-programs.scm`, and `link-to-modules.scm` accept the following options:

- `-b` *bootstrap-path*: Use the bootstrapped compiler or linker. The argument is the path to file `theme-d-compile-b.go` or `theme-d-link-b.go`.

- `-B` *bootstrap-dir*: Use the modularly linked bootstrapped compiler or linker. The argument shall be either directory `build1` or `bootstrap` in the bootstrapped environment.
- `-s` *split-dir*: Use the compiler or linker linked with split linking.

Programs `link-test-programs.scm` and `link-to-modules.scm` accept also the following options:

- `-i` *backend* : Select the linker backend. The value of *backend* has to be either `tree-il`, `guile`, or `guile0`. Value `guile0` means the nonoptimized Guile backend. The default value is `tree-il`.
- `-k` : Do not delete the intermediate file (`.tree-il` or `.scm`).

Command `compile-tests.scm` passes the contents of environment variable `EXTRA_COMP_OPTIONS` to the compiler. Command `link-test-programs.scm` passes the contents of environment variable `EXTRA_LINK_OPTIONS` to the linker. Programs `link-test-programs.scm`, `run-test-programs.scm`, `run-test-programs-w-output.scm`, and `check-test-runs.scm` accept option `--split-linking` in order to use split linking.

If you want to clean the testing environment in order to rebuild and rerun the examples and the tests give command `./clean-test-env.sh` in the subdirectory `testing`. If you want to keep the compilation output but clean the other files use command `./link-clean-test.env.sh`.

15 Computing Makefile Dependencies

The software includes three scripts to compute makefile dependencies for Theme-D files: `compute-theme-d-pcode-deps` for Theme-D pseudocode files (`*.tc?`), `compute-theme-d-program-deps` for monolithically linked programs (`*.go`), and `compute-theme-d-module-deps` for modularly linked modules (`*.go`). Each of these commands takes two arguments: the source code file (`*.th?`) for which to compute the dependencies and the unit path to specify the units included in the dependency computation. Only dependencies with the specified unit path are included. The unit path is computed from the unit name by dropping the last symbol. For example, set the second argument to `examples` for the example programs.

16 Other Things

An Emacs mode for Theme-D can be found at `tools/theme-d.el`. There are some example programs in subdirectory `theme-d-code/examples` in the Theme-D source package. You can compile, link, and run them following the instructions given in sections 8, 9, and 10. If you install the Theme-D Debian package twice the configuration file `theme-d-config` may not be installed. This problem is solved by uninstalling Theme-D and installing it again.

The software includes command `pretty-print-scheme`, which can be used to pretty-print Scheme code. The program can also be used to pretty-print Theme-D source code files to make them readable, typically for autogenerated code. It uses the Guile pretty printer.

Theme-D translator uses the following notation for printing pair and tuple types: `(:pair` r s`)` is printed as { r . s } and `(:tuple` t_1 ... t_n`)` is printed as {t_1 ... t_n}. Note that this notation is not accepted in Theme-D code.

17 Comments

The linker requires that the compiled modules are placed in a proper subdirectory hierarchy under some directory among the module search directories. This condition is fulfilled if you define the module search directories to include all the unit root directories used by your source files and put the compiled files into same directories with the source files.