Theme-D User Guide

Tommi Höynälänmaa

November 15, 2020

Contents

1 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`.

2 Installation

2.1 Debian-based Systems

These instructions apply to Debian-based Linux distributions such as Debian and Ubuntu. 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.

Symbol *rev* in the package names means the Debian revision of the packages. It is typically 1.

Install first one of the packages `guile-2.2-dev` or `guile-3.0-dev` (or both) Use command

```
apt-get install guile-2.2-dev
```

or

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

as root. You can check if these packages have already been installed with commands

```
dpkg -s guile-2.2-dev
dpkg -s guile-3.0-dev
```

If you have both `guile-2.2-dev` and `guile-3.0-dev` installed you may choose which platform to use for Theme-D.

2.1.1 Package `guile-2.2-dev` and Amd64 / Intel 64-bit x86 Processor Architecture

1. If your home directory contains file `~/.theme-d-config` delete the file.
2. Give the following commands as root:

```
dpkg -i th-scheme-utilities_3.1.0-rev_amd64.deb
dpkg -i libthemedsupport_3.1.0-rev_amd64.deb
dpkg -i theme-d-rte_3.1.0-rev_amd64.deb
dpkg -i theme-d-translator_3.1.0-rev_amd64.deb
dpkg -i theme-d-stdlib_3.1.0-rev_all.deb
```

in the directory where you have the Debian files.

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

```
dpkg -i theme-d-doc_3.1.0-rev_all.deb
```

as root.

If you want to rebuild the Debian packages follow the instructions in section **??**.

2.1.2 Other Debian-based Systems with Guile 2.2

1. If your home directory contains file ∼/.theme-d-config delete the file.
2. Change to the directory where you want to unpack the Theme-D source code
3. Copy files theme-d-3.1.0.tar.xz and theme-d_3.1.0-*rev*.debian.tar.xz into that directory.
4. Give command

```
ln -s theme-d-3.1.0.tar.xz theme-d_3.1.0.orig.tar.xz
```

5. Unpack Theme-D source code with command

```
tar xvf theme-d-3.1.0.tar.xz
```

6. Change to the subdirectory theme-d-3.1.0.
7. Give command

```
tar xvf ../theme-d_3.1.0-rev.debian.tar.xz
```

8. 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.

9. Give command

```
dpkg-buildpackage -uc -us -ui
```

10. Give commands

```
cd ..
dpkg -i th-scheme-utilities_3.1.0-rev_arch.deb
dpkg -i libthemedsupport_3.1.0-rev_arch.deb
dpkg -i theme-d-rte_3.1.0-rev_arch.deb
dpkg -i theme-d-translator_3.1.0-rev_arch.deb
dpkg -i theme-d-stdlib_3.1.0-rev_all.deb
```

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

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

```
dpkg -i theme-d-doc_3.1.0-rev_all.deb
```

as root.

2.1.3 Other Debian-based Systems with Guile 3.0

1. If your home directory contains file ~/`.theme-d-config` delete the file.
2. Change to the directory where you want to unpack the Theme-D source code
3. Copy files `theme-d-3.1.0.tar.xz` and `theme-d_3.1.0-`*rev*`.debian.tar.xz` into that directory.
4. Give command

```
ln -s theme-d-3.1.0.tar.xz theme-d_3.1.0.orig.tar.xz
```

5. Unpack Theme-D source code with command

```
tar xvf theme-d-3.1.0.tar.xz
```

6. Change to the subdirectory `theme-d-3.1.0`.

7. Give command

   ```
   tar xvf ../theme-d_3.1.0-rev.debian.tar.xz
   ```

8. Give command

   ```
   dch -v 3.1.0-newguile
   ```

   and write some comment into the changelog.

9. Change the value of the variable `GUILE_VERSION` to 3.0 in file `debian/rules` (6th line).

10. Change the package names `guile-2.2` and `guile-2.2-dev` to `guile-3.0` and `guile-3.0-dev` in the Build-Depends field in file `debian/control` (5th line).

11. 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.

12. Give command

    ```
    dpkg-buildpackage -uc -us -ui
    ```

13. Give commands

    ```
    cd ..
    dpkg -i th-scheme-utilities_3.1.0-newguile_arch.deb
    dpkg -i libthemedsupport_3.1.0-newguile_arch.deb
    dpkg -i theme-d-rte_3.1.0-newguile_arch.deb
    dpkg -i theme-d-translator_3.1.0-newguile_arch.deb
    dpkg -i theme-d-stdlib_3.1.0-newguile_all.deb
    ```

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

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

    ```
    dpkg -i theme-d-doc_3.1.0-newguile_all.deb
    ```

as root.

2.2 Other UNIX Systems

1. If your home directory contains file ~/.theme-d-config delete the file.
2. Install Guile 2.2 or 3.0 if you don't have it already. Check the version of the Guile development environment with commands

   ```
   pkg-config --modversion guile-2.2
   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-3.1.0.tar.xz`

   The subdirectory `theme-d-3.1.0` 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. The version has to be either 3.0 or 2.2.
   - `--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.

2.3 Using the Software without Installation

This sofware may also be used without installing it. This is useful if you develop Theme-D itself. It is recommended that you should not use Theme-D simultaneously with installed version and local mode.

1. Install guile 2.2 or 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-3.1.0.tar.xz`

3. Go into the the subdirectory `theme-d-3.1.0` of the directory created in the previous step. Give commands

   ```
   ./configure
   make
   ```

   See section **??** 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.

2.4 Support for Racket

If you want to use Racket as the target platform (i.e. the platform where you run your Theme-D programs) you have to install Racket package `theme-d-racket`. To do this obtain the file `theme-d-racket.zip` and give command

```
raco pkg install theme-d-racket.zip
```

3 Removing the Software

3.1 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.

3.2 Other Systems

Give command

```
make uninstall-complete
```

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

3.3 Support for Racket

Give command

```
raco pkg remove theme-d-racket
```

4 Theme-D Environment

5 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

6 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.

7 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.

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.

8 Linking a Theme-D Program

Give command

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

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

- `--module-path=` *paths* or `-m` *paths* : Module search paths separated with :'s
- `--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.
- `-y` *module*: Link (load) the module into the target program using a relative path. This option is available only for Racket.
- `--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
- `--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.
- `--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.

The available intermediate languages are:

- `tree-il` : Guile Tree-IL. The Tree-IL version shall be the Guile version for which Theme-D has been configured.
- `tree-il-3.0` : Guile 3.0 Tree-IL. This is currently identical to Guile 2.2 Tree-IL.
- `tree-il-2.2` : Guile 2.2 Tree-IL.
- `guile` : Guile Scheme. The Guile version shall be the version for which Theme-D has been configured.
- `guile-3.0` : Guile Scheme 3.0.
- `guile-2.2` : Guile Scheme 2.2.
- `racket` : Racket Scheme.

The option `--no-optimization` has no effect for target platforms for target platforms `--tree-il` and `--tree-il-xxx`. They are always optimized. If you use Racket options `--keep-intermediate` and `--no-final-compilation` are assumed implicitly. 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 module names in the `-x` and `-y` options depends on the intermediate language. It is `"(mod1 ... modn)"` for Guile and `mod1/.../modn` for Racket. If you use the `-y` option for linking a module (available only for Racket) the module is imported with a `require` form so that the

module name is enclosed in double quotes. See the Racket documentation how this works.

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

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

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

If you have a big program it is useful to split the linker output to several intermediate files so that the Guile bytecode compiler does not begin to use the swap memory. 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.

9 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 use Racket as the target platform you can run Theme-D programs with command

```
racket -t myprog.rkt -m programarg ...
```

or

```
racket -t myprog.rkt 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.

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.

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

10 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.
- `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 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.

11 Distributing Linked Theme-D Programs

If your target environment has Theme-D installed it is sufficient to distribute only the linked `.go` file. Otherwise, the easiest way to ensure that all the necessary Theme-D files are present is to install the `theme-d-rte` Debian package in the target system. If you use Racket as the target platform it is sufficient to install package `theme-d-racket.zip` in the target system, see section **??**.

If you are using Guile and you don't want to install `theme-d-rte` into the target system you have to ensure that the following files are present in the Guile library path:

- `theme-d/runtime/params.go`
- `theme-d/runtime/runtime-theme-d-environment.go`
- `theme-d/runtime/theme-d-stdlib-support.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. 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` in the target system. These files 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.

12 Compiling, Linking, and Running Test and Examples 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`. 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
```

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`.

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
```

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
```

and link them with command

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

Then run the linked programs with command

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

The test results can be checked with commands

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

for compilation, linking and running, respectively. 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
```

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 backtrace in `test764` may be different in different runs.

If you want to use `scheme` (Guile) as the intermediate language use command

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

for linking. If you want to use `racket` as the intermediate language use scripts

```
./link-test-programs-racket.scm
./check-test-program-linking-racket.scm
./run-test-programs-racket.scm
./check-test-runs-racket.scm
./run-test-programs-w-output-racket.scm
```

for linking and running the test programs. Test `test450` does not work for Racket. The following tests give different output with Guile and Racket: 115, 235, 273, 277, 450, 556, 598, 606, 607, 673, 716, 717, 720, 729, 743, and 756. Note that the Racket-related scripts omit the tests using features not available for Racket, such as GOOPS or the extra math functions in the support library.

13 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 **??**, **??**, and **??**. 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.

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.

14 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.