Joint Center for Satellite Data Assimilation

CRTM: Subversion Repository Guide

Paul van Delst[a]
JCSDA/EMC/SAIC

November, 2008

---

[a]paul.vandelst@noaa.gov

Change History

Date	Author	Change
2008-08-29	P.van Delst	Initial release.
2008-08-30	P.van Delst	Added Build Conventions chapter.
2008-11-06	P.van Delst	Added Commit Log Messages chapter. Added bibliography.

Contents

List of Figures

List of Tables

1 Introduction

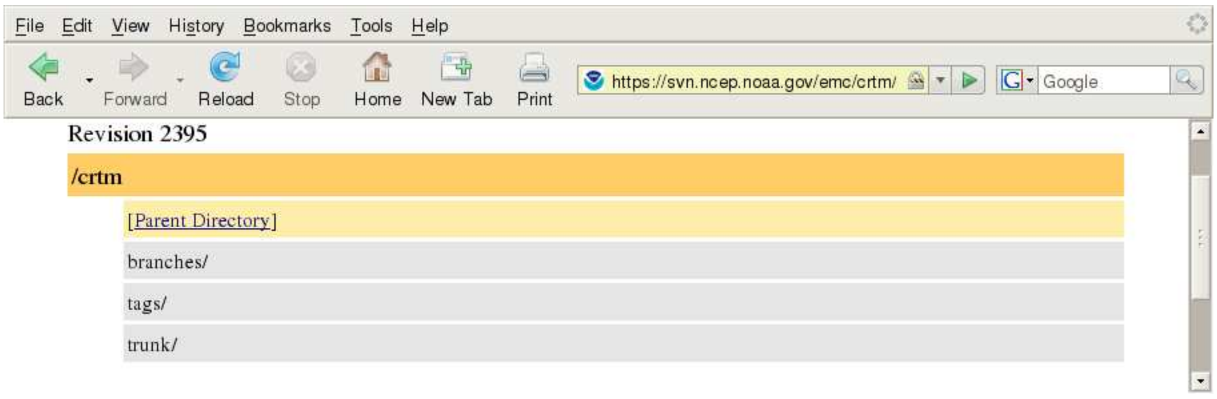
This document describes the CRTM subversion repository and how to set up your environment to allow you to build the CRTM library, and any associated programs, directly in your working copy. It is assumed the user is familiar with Subversion.

The CRTM is one of many projects in main EMC repository on the subversion server `svn.ncep.noaa.gov`. The location of the CRTM project repository is `https://svn.ncep.noaa.gov/emc/crtm`. Note that, as of November 2008, the EMC repository is still behind the NCEP firewalls and access is available only to users on the NCEP network, or via a VPN connection into the NCEP network.

A read-only mirror of the EMC repository is available on the vapor supercomputer. The location of the CRTM repository mirror on vapor is `file:///gpfs/v/svn/emc/crtm`. This mirror is automatically updated from the subversion server every night so users should be aware that any changes to the repository will not be available on vapor until the next day.

2 Repository Organisation

If you access the repository via your browser, you should see something like figure 2.1, where the repository is organised into the usual `trunk`, `branches`, and `tags` subdirectories.


File Edit View History Bookmarks Tools Help
Back
Forward
Reload
Stop
Home
New Tab
Print
https://svn.ncep.noaa.gov/emc/crtm/
Google

Revision 2395

/crtm

[Parent Directory]

branches/

tags/

trunk/

Figure 2.1: The root of the CRTM repository organised into the typical `trunk`, `branches`, and `tags` subdirectories.

2.1 `trunk` subdirectory

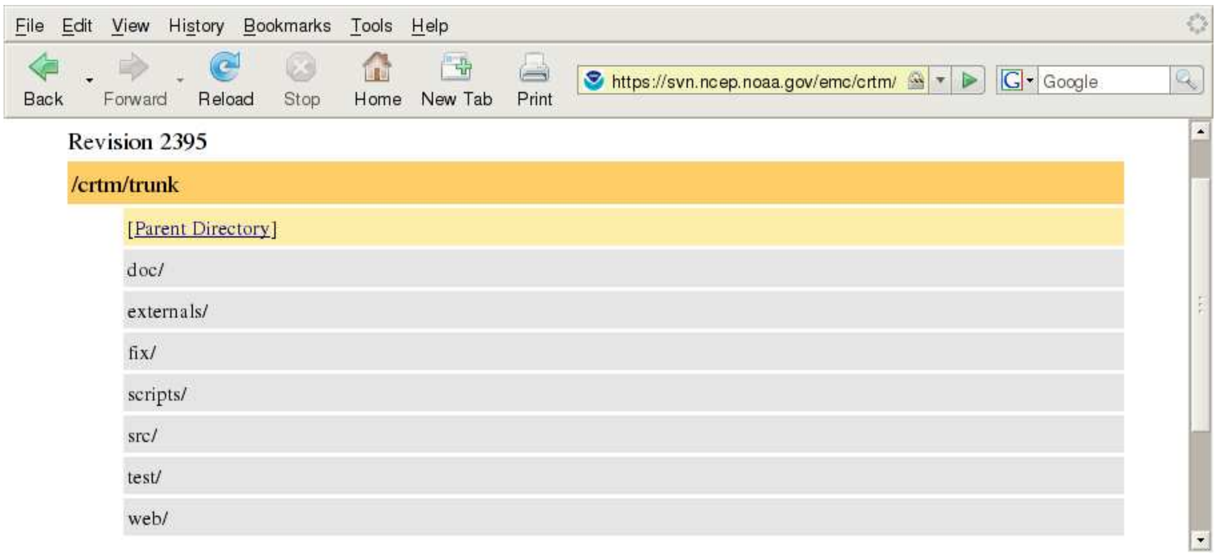
Mainline development of the CRTM is done in the trunk. Navigating the `trunk` link of the web page shown in figure 2.1, displays the various categories of the CRTM repository as shown in figure 2.2. A short description of the trunk subdirectories are shown in table 2.1

As indicated, the `src` subdirectory is the one that contains the actual CRTM source code. This and the `fix` directory, which contains all of the spectral, transmittance, aerosol, cloud, and surface emissivity coefficient datafiles, are the two main parts of the CRTM repository.

2.2 `branches` subdirectory

Development independent of the main CRTM trunk is done in the branches subdirectory. Currently, the CRTM contains only `src` category branches and, of those, there are two types:

1. Experimental developmental branches where wholescale changes to the CRTM may result in instability. The naming convention is `EXP-`*desc* where *desc* is a short description of the experiment. For example, a branch named `EXP-DISORT` has been created to implement the DISORT radiative transfer solver in the CRTM for validation.



Revision 2395

/crtm/trunk

[Parent Directory]

doc/

externals/

fix/

scripts/

src/

test/

web/

Figure 2.2: The trunk of the CRTM repository, showing the various categories.

Category	Description
`doc`	CRTM documentation
`externals`	Library of third party software used in the CRTM and/or support software
`fix`	Coefficient datafiles used by the CRTM.
`scripts`	Hierarchy of script software, for various languages, used in CRTM build, testing, visualisation, etc.
`src`	Main CRTM Fortran95 source code directory. Contains the core CRTM modules as well as support software.
`test`	CRTM testing. Contains all the tests (unit, component) used to validate the CRTM.
`web`	The CRTM webpage source.

Table 2.1: Description of the contents of the CRTM repository trunk categories.

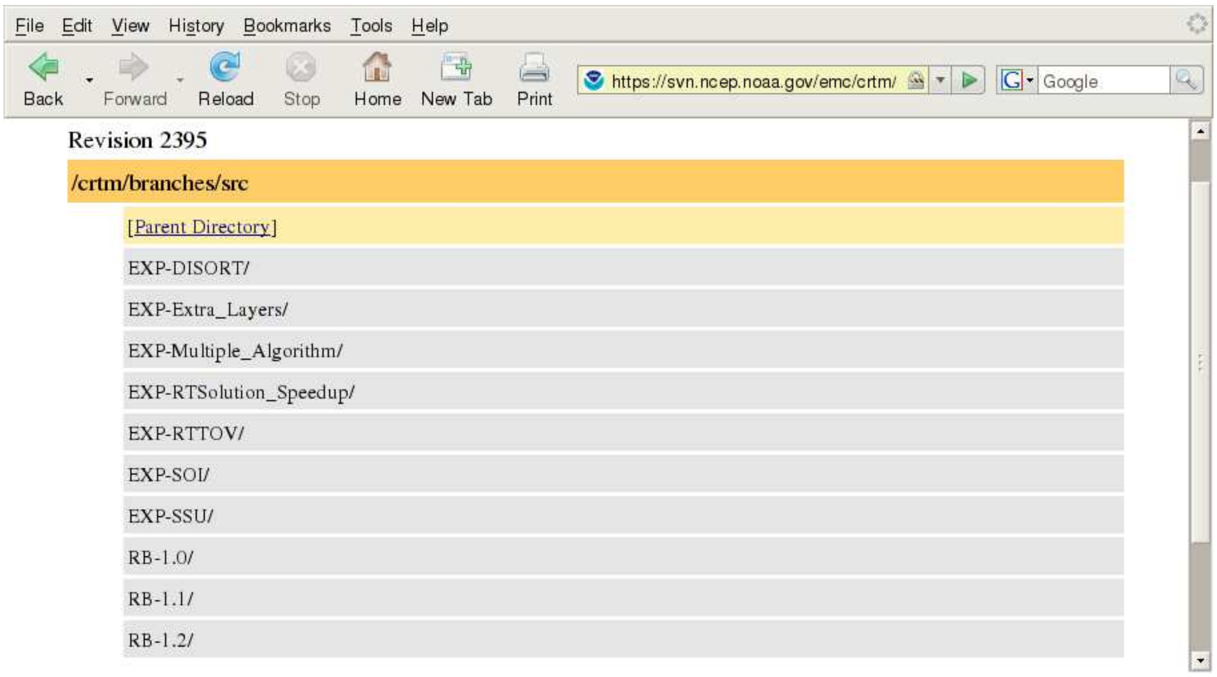
2. Code release branches where the code is tested and "tweaked" prior to a release. The naming convention here is `RB-`*rel* where *rel* is the planned release version number. An example would be the v1.2 release branch, `RB-1.2`.

The current state of the CRTM `branches/src` subdirectory is shown in figure 2.3

2.3 `tags` subdirectory

If a snapshot of development is wanted, or if development has been completed on a trunk or branch revision, a copy is made and placed in the `tags` subdirectory. There are three tag naming conventions in current use:

1. For official software releases, `REL-`*rel*; where *rel* is the software relase number. For example, the first official CRTM release has the tag `REL-1.1`.
2. For pre-release snapshots, `REL-`*rel*_*stage*`.rev`*RN*`.`*YYYY-MM-DD*; where *stage* is the release stage, typically `alpha` or `beta`; *RN* is the repository revision number from which the tag was created; and *YYYY-MM-DD* is the date on which the tag was created. An example of this is `REL-1.1_beta.rev1855.2008-02-25`.
3. For experimental branch snapshots, `EXP-`*desc*`.rev`*RN*`.`*YYYY-MM-DD*; where *desc* is a short description of the experimental branch. An example of this is `EXP-Extra_Layers.rev1738.2008-02-12`.


File Edit View History Bookmarks Tools Help
Back
Forward
Reload
Stop
Home
New Tab
Print
https://svn.ncep.noaa.gov/emc/crtm/
Google

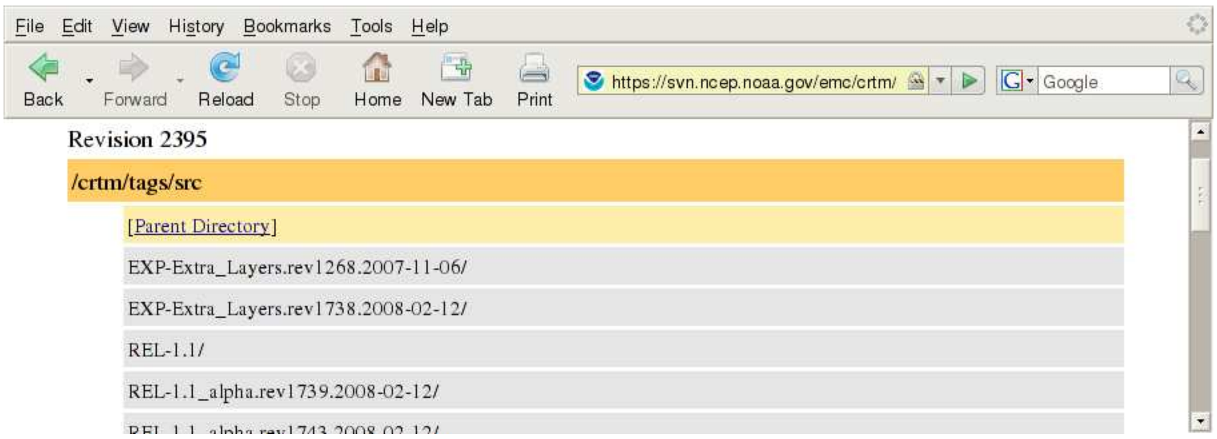
Revision 2395

/crtm/branches/src

[Parent Directory]
EXP-DISORT/
EXP-Extra_Layers/
EXP-Multiple_Algorithm/
EXP-RTSolution_Speedup/
EXP-RTTOV/
EXP-SOI/
EXP-SSU/
RB-1.0/
RB-1.1/
RB-1.2/

Figure 2.3: Snapshot of the `branches/src` subdirectory of the CRTM repository, showing the current branches.

Some examples of the current tags in the CRTM **`tags/src`** subdirectory are shown in figure 2.4. Note that there is no development in a tag directory - it is strictly a snapshot of a trunk or branch revision.


File Edit View History Bookmarks Tools Help
Back
Forward
Reload
Stop
Home
New Tab
Print
https://svn.ncep.noaa.gov/emc/crtm/
Google

Revision 2395

/crtm/tags/src

[Parent Directory]
EXP-Extra_Layers.rev1268.2007-11-06/
EXP-Extra_Layers.rev1738.2008-02-12/
REL-1.1/
REL-1.1_alpha.rev1739.2008-02-12/
REL-1.1_alpha.rev1743.2008-02-12/

Figure 2.4: Snapshot of the `tags/src` subdirectory of the CRTM repository, showing some current tags.

3 Build Conventions

This sections details the environment setup to enable the CRTM library, or any support software, to be compiled in a user's working copy. For the purposes of explanation we will assume that the entire CRTM trunk working copy has been checked out using something like the following commands,

```
cd $HOME/CRTM
svn checkout https://svn.ncep.noaa.gov/emc/crtm/trunk trunk
```

where a user's home directory is referred to by the environment variable `$HOME`, and the root directory of a user's working copy of the CRTM is `$HOME/CRTM` and reflects the same directory structure as the repository.

Additionally, it is assumed there exists a user directory, `$HOME/bin`, which is defined in a user's `$PATH`. Compiled executables and scripts will be placed in this directory.

3.1 Macro Definitions

All of the makefiles in the CRTM repository use environment variables as required to locate the particular category subdirectories described in table 2.1. The environment variable names, along with example definitions for a working copy are shown in table

Environment Variable Name	Example Definition
`CRTM_SOURCE_ROOT`	`$HOME/CRTM/trunk/src`, or `$HOME/CRTM/branches/src/RB-1.2`
`CRTM_FIXFILE_ROOT`	`$HOME/CRTM/trunk/fix`
`CRTM_TEST_ROOT`	`$HOME/CRTM/trunk/test`
`CRTM_SCRIPTS_ROOT`	`$HOME/CRTM/trunk/scripts`
`CRTM_EXTERNALS_ROOT`	`$HOME/CRTM/trunk/externals`
`CRTM_DOC_ROOT`	`$HOME/CRTM/trunk/doc`

Table 3.1: Environment variables used by CRTM makefiles.

Ideally, the environment variables of table 3.1 should be defined in a user's environment definition file to ensure they will be defined in any shell invocation.

For now, just note the multiple examples for the `CRTM_SOURCE_ROOT` macro. The reason for this will be explained later (see section 3.4).

3.2 Install of script files

The simplest way to build a library is to have all the source code in a single directory. The CRTM source code modules in the `src` directory, however, are organised into separate subdirectory hierarchies according to their application. It is expected that this organisational structure will change over time. Rather than create makefiles that need to know what the directory structure is to find all the various source files, a shell script (`linkfiles`) is used to link all the necessary files into the CRTM library build subdirectory, `src/Build`.

So, the second step in setting up the CRTM build environment is to install the necessary scripts. The current method for doing this is through unsophisticated use of makefiles. The sequence of commands for the script install are,

```
cd $CRTM_SCRIPTS_ROOT/shell/Utility
make install
```

This installs all the scripts currently used in the CRTM build process. Note there is also an uninstall target that removes all the scripts from a user's local `bin` directory.

3.3 Master Make Include Files

All of the makefiles in the CRTM repository use three standard include files: `make.macros`, `make.common_targets` and `make.rules`. These files reside in the `CRTM_SOURCE_ROOT` subdirectory and their function is described in table 3.2.

Include File Name	Description
`make.macros`	Defines macros for all the compiler and linker flags for the supported compiler/platform combinations, as well as commonly used operating system commands and utilities, e.g. `cp`, `rm`, `ar`, etc.
`make.common_targets`	Defines the common targets used in builds, e.g. `all`, `install`, `clean`, etc.
`make.rules`	Defines the suffix rules for compiling Fortran source code.

Table 3.2: Include files used by CRTM makefiles.

3.4 Building the CRTM Library

Having setup the environment on a system, the sequence of commands to build and install the CRTM library in a checked out working copy is,

```
cd $CRTM_SOURCE_ROOT
make create_links
make
make install
```

The first target, `create_links`, searches for all the required CRTM source code starting at `$CRTM_SOURCE_ROOT` and links it all into the `Build/src` subdirectory[1].

The second make does the actual source code compilation and library creation.

The last target, `install`, moves the created CRTM library, `libCRTM.a`, into the `Build/lib` subdirectory and all of the associated `*.mod` module files into the `Build/include` subdirectory.

If you wish to build a particular branch or release (tag) of the CRTM library, all you need to do is redefine the `CRTM_SOURCE_ROOT` environment variable to your working copy location for that branch or release. The

[1]For some systems, notably the IBM systems at NCEP, this can take several minutes. For linux desktop systems, it should only take a few seconds. A ruby version of the script does exist that is quite a bit faster.

redefinition can be system-wide (e.g. if you're working solely on a release branch in preparation for the release) or for a single shell session (e.g. if you're building an older or experimental version alongside the current release).

If the build is a final one (e.g. you're not testing the CRTM), the `Build/lib` and `Build/include` subdirectories are typically copied or moved to a generic location outside of the working copy, e.g. `$HOME/local/lib` and `$HOME/local/include`, or `$HOME/local/CRTM/lib` and `$HOME/local/CRTM/include`

As mentioned in section 3.3, compiler flags for varous platforms (or in the case of linux, for various compilers) are defined in the `make.macros` file. Instructions on how to modify the `make.macros` for different compilers on a linux systems can be found in the `Build/README` file.

3.5 Cleaning up

There are three targets that tidy up after a CRTM build. Depending on your needs they clean up intermediate files to varying degrees. A description of the clean targets is shown in table 3.3.

Target Name	Description
`clean`	Removes all the `*.o`, `*.mod`, `*.a` files from the `Build/src` subdirectory.
`distclean`	Same as `clean` but also deletes the `Build/lib` and `Build/include` subdirectories.
`realclean`	Same as `distclean` but also deletes the source code symbolic links in the `Build/src` subdirectory.

Table 3.3: Cleanup targets in the CRTM library build makefiles.

If you invoke the `realclean` target and want to subsequently rebuild the CRTM libarry, you will have to recreate the links as detailed in section 3.4. And, remember, creating the links can take some time on some systems.

4 Commit Log Messages

The purpose of this section is to describe the convention for commit log message formats. This may seem overly meticulous, but the goal is to use the repository commit messages to form the change log for CRTM releases. The change log should show the history of the devleopment of the CRTM and as more developers contribute to the CRTM directly by committing to the repository, the log message format should not differ from one developer to the next.

It is conceivable that at some point in the future the subversion log outputs will be automatically processed via a script to create the change log file for distribution with a CRTM release–or posting on a web page–so developers should endeavour to adhere to this formatting standard so as make parsing the log output easier.

Nearly all of the advice and format descriptions in this section are either taken directly or paraphrased from either the Change Logs section of the GNU Coding Standards, FSF [2008(a)], or the Change Log Guidelines section of the GNU guile project FSF [2008(b)].

Some generic points for good log messages (taken from [FSF, 2008(b)]) are:

1. Log messages should consist of complete sentences, not fragments. Sentence fragments can be ambiguous. Fragments like "Initial commit" for a new file, or "Added function" for a new function are acceptable, because they are standard idioms.
2. Log messages should mention every file changed, as well as mention by name every function and/or subroutine changed. Some common sense exceptions,
   - For trivial changes (e.g. renaming a variable), all affected procedures do not have to be listed.
   - For a complete rewrite of a file, a log entry description such as "Rewritten" is acceptable.
3. Group log message entries in "paragraphs", where each paragraph describes a set of changes with a single goal.
4. Do not abbreviate filenames or procedure names. It makes the log message output difficult to search for changes to these files and procedures.

Specific formats requirements with examples follow.

4.1 Log message format

An example of a log message format for a CRTM commit is shown in figure 4.1, starting with a header line that describes the CRTM category (in this case `src`, but see table 2.1 for all the current categories) and the relative source file location (here `Utility/InstrumentInfo/SpcCoeff`), followed by descriptions of the changes being committed.

Each entry is bulleted using the "*" character, followed by the filename (or list of filenames). Functions and subroutines are surrounded by parentheses. Always use the specific procedure name in the source code, not the

```
  src:Utility/InstrumentInfo/SpcCoeff subdirectory.
  * SpcCoeff_Define.f90 (Associated_SpcCoeff): Removed Skip_AC optional argument.
    (Assign_SpcCoeff): Removed Skip_AC actual argument in call to Associated_SpcCoeff.
```

Figure 4.1: Commit log message format for a commit to the `trunk`.

generic (or overloaded) procedure name. Additionally, if similar changes were made to many procedures such that the list doesn't fit on a single line, close the parentheses before the line break and reopen them on the next line continuing with the procedure list. This makes the modified procedures easier to search for in the log messages[1]

An example of a multiple entry log message is shown in figure 4.2. Note the separate <*category*>:<*directory*> header lines

```
  src:Statistics/FitStats subdirectory.
  * FitStats_Define.f90: Made the maximum number of predictors a public entity.
  * FitStats_netCDF.f90: Major rewrite. The various netCDF utility modules are no
    longer used.

  src:Statistics/FitStats/Test_FitStats subdirectory.
  * Makefile, make.dependencies: Updated to reflect changes to the FitStats_netCDF
    module.
  * Test_FitStats.f90: Decreased the number of loops used in the memory leak checks for
    use with valgrind.

  src:Statistics/FitStats/FitStats_ASCII2NC subdirectory.
  * FitStats_ASCII2NC.f90: Modified for use with microwave statistics files where there
    is no ozone component.
  * Makefile, make.dependencies: Updated to reflect changes in the main FitStats modules.
```

Figure 4.2: Multiple entry commit log message format for a commit to the `trunk`.

4.2 Branch creation log message format

When a branch is initially created the log message should state the branch name, and also identify the revision and source from which it was created. The log message for the creation of the CRTM v1.2 release branch is shown in figure 4.3.

```
RB-1.2 branch. Created from r2376 trunk.
```

Figure 4.3: Commit log message format when creating a branch.

As is clear, the name of the branch is `RB-1.2` and it was created from revision 2376 of the trunk. It is useful to list the source since a branch *may* be created from another branch, although this practice is generally discouraged.

[1]A lesson the author learned the hard way as you will undoubtedly encounter log messages that do not do this and these cases tend to break simple searching commands or scripts.

4.3 Branch commit log message format

When committing to a branch, the log message format is the same as for the trunk, *except* that the branch name should *always* be listed first. Doing this allows searching of the log messages for all instances of commits to a particular branch. An example of a branch commit log message is shown in figure 4.4.

```
RB-1.2 branch.
  src:Surface subdirectory.
  * CRTM_Surface_Binary_IO.f90 (Read_Surface_Record, Write_Surface_Record): Updated I/O
    statements that contained references to the SensorData structure components to be
    consistent with the structure definition updates from r2572.
```

Figure 4.4: Commit log message format for a commit to a branch, in this case the RB-1.2 branch.

4.4 Merge log message format

When merging changes from a branch to the trunk (or vice versa), the range of revisions merged should be specified in the log message. An example of a merge log message format is shown in figure 4.5

```
Merged RB-1.2 branch r2377:2444 into the trunk.
```

Figure 4.5: Commit log message format for a merge from a branch, in this case the RB-1.2 branch, to the trunk

Thus, the log message contains a record of what was merged, what revisions were merged, and what they were merged into. Inspection of the log messages informs developers at what revisions future merges should begin (in the example of figure 4.5, that would be r2445).

Bibliography

FSF. GNU Coding Standards, 2008(a). URL http://www.gnu.org/prep/standards. Last accessed 2008-11-06.

FSF. Guile, Project GNU's extension language, 2008(b). URL http://www.gnu.org/software/guile. Last accessed 2008-11-06.