

Joint Center for Satellite Data Assimilation

CRTM: Subversion Repository Guide

Paul van Delst^a
JCSDA/EMC/SAIC

September, 2008

^apaul.vandelst@noaa.gov

Contents

1	Introduction	1
2	Repository Organisation	2
2.1	trunk subdirectory	2
2.2	branches subdirectory	2
2.3	tags subdirectory	3
3	Build Conventions	5
3.1	Macro Definitions	5
3.2	Install of script files	5
3.3	Master Make Include Files	6
3.4	Building the CRTM Library	6
3.5	Cleaning up	7

List of Figures

2.1	The root of the CRTM repository organised into the typical trunk , branches , and tags subdirectories.	2
2.2	The trunk of the CRTM repository, showing the various categories.	3
2.3	Snapshot of the branches/src subdirectory of the CRTM repository, showing the current branches.	4
2.4	Snapshot of the tags/src subdirectory of the CRTM repository, showing some current tags.	4

List of Tables

2.1	Description of the contents of the CRTM repository trunk categories.	3
3.1	Environment variables used by CRTM makefiles.	5
3.2	Include files used by CRTM makefiles.	6
3.3	Cleanup targets in the CRTM library build makefiles.	7

DRAFT

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 September 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.

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.

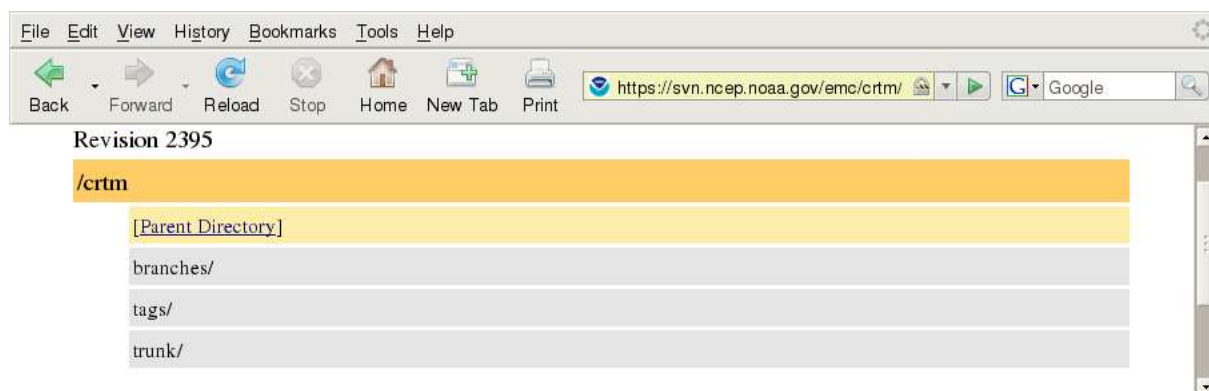


Figure 2.1: The root of the CRTM repository organised into the typical trunk, branches, and tags subdirectories.

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 wholesale 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.

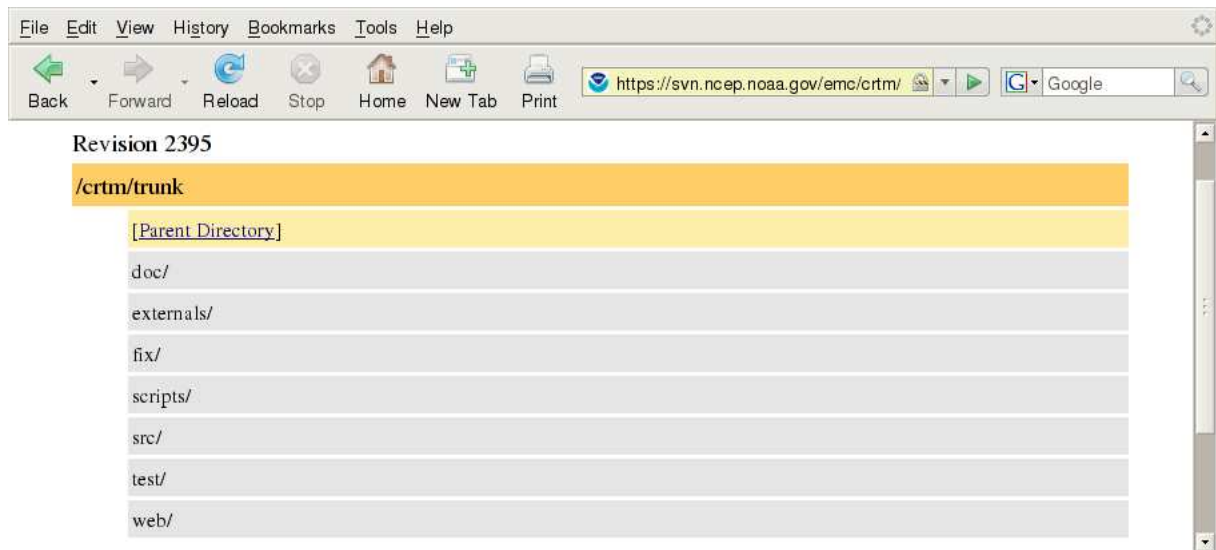


Figure 2.2: The trunk of the CRTM repository, showing the various categories.

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 release number. For example, the first official CRTM release has the tag `REL-1.1`.
2. For pre-release snapshots, `REL-rel_stage.revRN.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.revRN.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`.

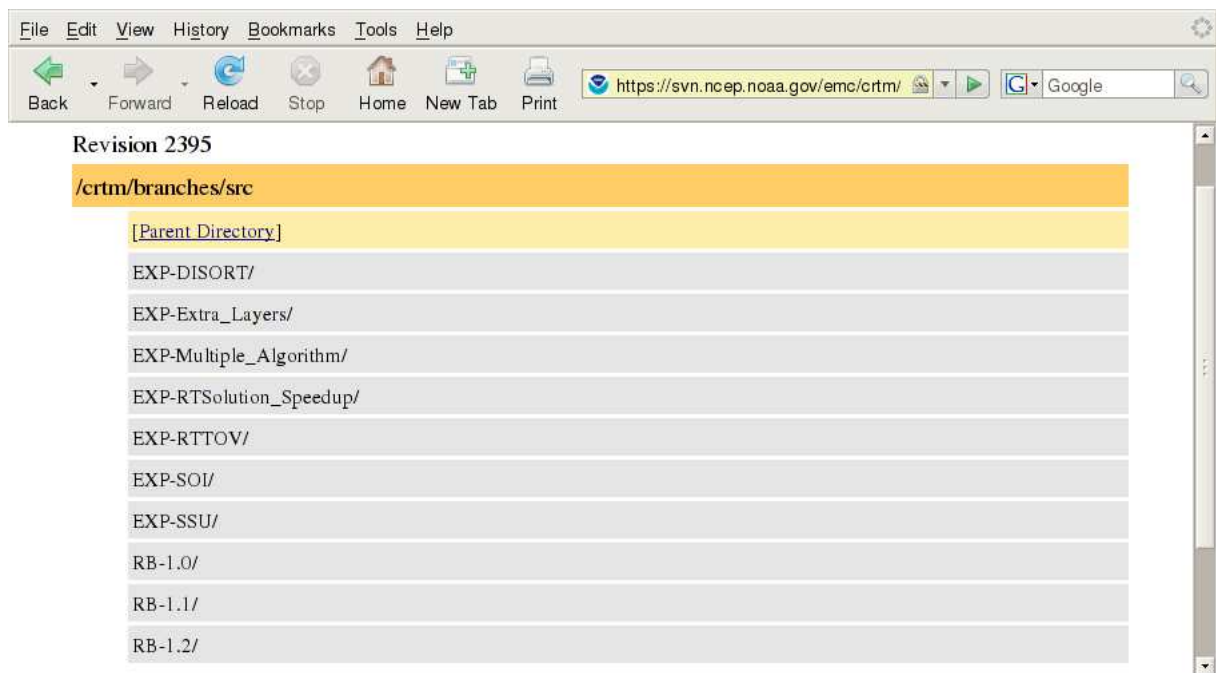


Figure 2.3: Snapshot of the branches/src subdirectory of the CRTM repository, showing the current branches.

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.

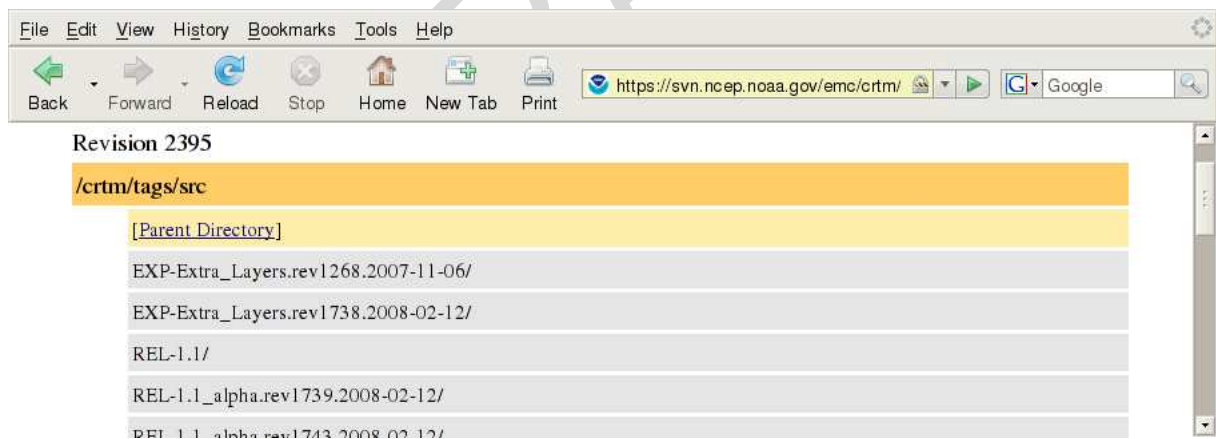


Figure 2.4: Snapshot of the tags/src subdirectory of the CRTM repository, showing some current tags.

Build Conventions

This section 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	<code>\$HOME/CRTM/trunk/src</code> , or <code>\$HOME/CRTM/branches/src/RB-1.2</code>
CRTM_FIXFILE_ROOT	<code>\$HOME/CRTM/trunk/fix</code>
CRTM_TEST_ROOT	<code>\$HOME/CRTM/trunk/test</code>
CRTM_SCRIPTS_ROOT	<code>\$HOME/CRTM/trunk/scripts</code>
CRTM_EXTERNALS_ROOT	<code>\$HOME/CRTM/trunk/externals</code>
CRTM_DOC_ROOT	<code>\$HOME/CRTM/trunk/doc</code>

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
<code>make.macros</code>	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. <code>cp</code> , <code>rm</code> , <code>ar</code> , etc.
<code>make.common.targets</code>	Defines the common targets used in builds, e.g. <code>all</code> , <code>install</code> , <code>clean</code> , etc.
<code>make.rules</code>	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` subdirectory¹.

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

¹For 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 various 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
<code>clean</code>	Removes all the <code>*.o</code> , <code>*.mod</code> , <code>*.a</code> files from the <code>Build/src</code> subdirectory.
<code>distclean</code>	Same as <code>clean</code> but also deletes the <code>Build/lib</code> and <code>Build/include</code> subdirectories.
<code>realclean</code>	Same as <code>distclean</code> but also deletes the source code symbolic links in the <code>Build/src</code> 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 library, 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.