Joint Center for Satellite Data Assimilation

CRTM: Coding Review and Acceptance Guidelines

Paul van Delst[a]
JCSDA/EMC/SAIC

January, 2008

---

[a]paul.vandelst@noaa.gov

1 Motivation

Manage the workload and potential for interruption due to ongoing development of the CRTM.

2 Code Review Process

The following details the proposed procedure for development and acceptance of source code for use in a CRTM release.

- Prior to any coding
  - Developer shall propose changes in advance. For significant changes, a design review should be held to ensure the proposed changes will integrate into the CRTM; or, alternatively, suggest alterations to the internal structure of the CRTM to accomodate the developer's code.
  - Developer shall adhere to CRTM Coding Guidelines[3].
  - Developer shall update their CRTM working copy to the repository head revision and ensure that tests pass. For significant changes, a repository branch will be created for the developer.
- Prior to any CRTM trunk commit
  - Developer shall make updated code available to other developers for optional review.
  - For a significant change (i.e. not just a simple bug fix), the developer shall make working code available to other developers with enough lead time to permit review before the desired commit date.
  - Developer shall add new tests to test suite for new features.
  - Interested developers may review the code and suggest modifications.
  - Developers who are responsible for a section of code may require changes.
  - Proposed changes are discussed. Meeting schedule for this needs to be determined (e.g. regular, or as required?). Changes that are ready to commit are ordered.
- Person responsible for a particular section of code has veto power of commit.
- After all commits are completed, JCSDA CRTM core developers shall run an extensive suite of tests.
- Any test failures exposed by the extensive suite shall be fixed by the original developer(s).

DRAFT

3 Code Selection Criteria

This section needs some quantitative guidance from operational users.Additional, or more useful criteria?

Currently, there are three categories of code acceptance criteria.

3.1 Review

- Clarity. Is the code understandable?
- Maintainability. Is the code designed for easy maintenance?

These criteria are somewhat subjective but, as stated, they can be satisfied via iterations of the code review process discussed in section 2.

3.2 Specification

- Speed. Is it fast enough?
- Accuracy. Is it accurate enough?
- Memory usage. Does it use too much memory?

To be useful, the specification criteria obviously need to be quantified but the relative importance of each is dependent on the application. A two-tiered approach is suggested.

Firstly, what are the limits that would cause current operational codes to fail? That is, if the code is too slow then much less satellite data will make it into the data assimilation system eventually degrading the forecast; or if the code uses too much memory, page faulting causes it to halt.

Secondly, what are the margins that developers can use to guide their development? Determining these numbers is a bit more difficult as it depends on the application and the user's hardware.

A chart of typical/suggested values for the different JCSDA partners?

3.3 Testing

3.3.1 Baseline model tests

Code should be provided that runs the CRTM module under development and compares it with supplied results that are known to be correct.

Baseline test for all components, i.e. forward, tangent-linear, adjoint, K-matrix (if appropriate)? Or just the forward model since the following should test for the others?

DRAFT

3.3.2 Forward/Tangent-linear test

Let $F(x)$ represent the forward model for input x, and $TL(\delta x)$ represent the tangent-linear model for perturbation input δx. The relationship

$$\frac{F(x+\alpha\cdot\delta x)-F(x)}{\alpha\cdot TL(\delta x)}=1+O(\alpha) \tag{3.1}$$

should be true, where $\alpha=0.1,0.01,0.001,\ldots$

3.3.3 Tangent-linear/Adjoint Test

Let $\mathbf{TL}(\delta\mathbf{x})$ represent the tangent-linear model result for perturbation input vector $\delta\mathbf{x}$, and $\mathbf{AD}\left(\mathbf{TL}(\delta\mathbf{x})\right)$ represent the adjoint model result where the tangent-linear model output is used as adjoint model input. The following relationship,

$$\mathbf{TL}(\delta\mathbf{x})^{\mathbf{t}}\cdot\mathbf{TL}(\delta\mathbf{x})=\delta\mathbf{x}^{\mathbf{t}}\cdot\mathbf{AD}\left(\mathbf{TL}(\delta\mathbf{x})\right) \tag{3.2}$$

must be true to within numerical precision.

3.3.4 Adjoint/K-matrix Test

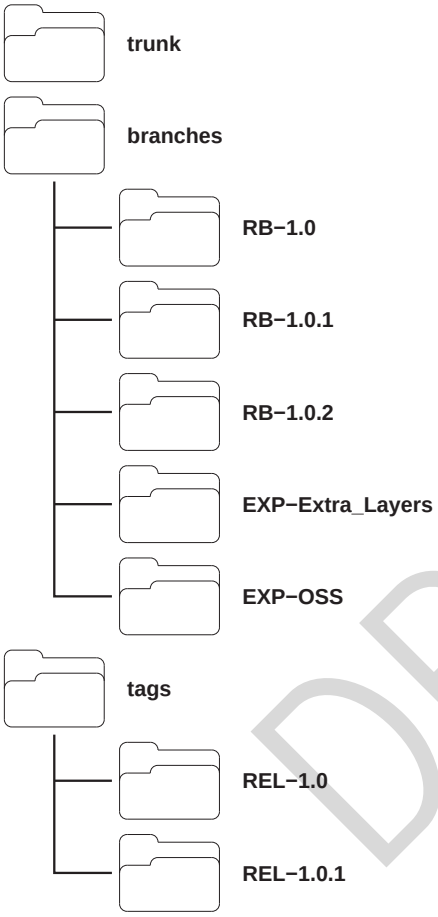
For specified adjoint input, δy^*, let $AD(\delta y^*)$ represent the adjoint model result, and $K_l(\delta y^*)$ the K-matrix result for sensor channel l. The following relationship

$$AD(\delta y^*)=\sum_{l=1}^{L}K_l(\delta y^*) \tag{3.3}$$

must be true to within numerical precision.

4 Repository Organisation

The top-level CRTM repository structure is shown in figure 4.1. This type of structure is standard for repositories in general[1, 2]. The naming of branches and tags as shown in figure 4.1 is also a *de facto* standard (see [2] for additional examples) that emphasises convention over configuration. That is, any developer should be able to inspect the repository and determine branch and tag information from their names.



- Three main directories
  - trunk
  - branches
  - tags
- trunk
  - The main line of development
  - Subdirectories here are based on logical association of what different CRTM components do (e.g. AtmOptics, SfcOptics, etc.)
- branches
  - This is where branches go.
  - Experimental development branches with name EXP-*desc*
  - Other branches related to code releases with name RB-*rel*
- tags
  - This is where snapshots and releases go.
  - Snapshots. REL-rev*num.YYYY-MM-DD*
  - Releases. REL-*rel*
  - No development (otherwise it would be a branch.)

DRAFT

Figure 4.1: Example and description of top-level directory structure of the CRTM Subversion repository.

5 Repository Access

5.1 Policy

- To start with, developers will have **read-only** access. It is acknowledged that this restriction works against agile development[1], but until there are sufficient test cases it's a necessary evil.
- Until write-access is granted, code to be considered for commit is provided as a tarball uploaded to CRTM ftp site.

[1] That's agile with a lower case "a". It should not be confused with the Agile Development framework for software engineering. See http://en.wikipedia.org/wiki/Agile_software_development

- CWG co-chairs responsible for repository branch creation. Developers request a branch for write-access.

5.2 Details

- Offsite Subversion repository.
- SCM software **this is important!**
- Website/wiki for online CRTM documentation.
- ftp site for release download, or code upload.
- User forum.

We need this capability to be *much* more flexible than it is now. IT security restrictions have hampered CRTM development. This is not an indictment of the current security protocols and procedures - they are there for a reason - but the current status quo is slowing development.

6 Code Release Policy

This section needs better organisation

DRAFT

- All releases will be made from the top of the main trunk of the repository. All developers will be notified in advance of any type of release.
- Prior to every release, a release branch will be created for pre-release testing. During this pre-release stage, the the main trunk of the repository will still be available for development. Bug-fixes in either the current release branch or trunk shall be merged.
- In order to allow time for developers to test modifications on their machines and contribute bug fixes in time for pre-release testing, all releases must be announced at least two months in advance.
- All components of the CRTM will be released together.
- A "major release" contains important new scientific and/or software engineering advancements that significantly alter the behavior and/or performance of the CRTM. Numbering for major releases increments the first element of the version number. Examples: "CRTM 2.0.0", "CRTM 3.0.0". Major releases will be synchronized across all development subgroups and scheduled well in advance. There will be no more than one major release per year.
- A "minor release" contains minor feature additions and enhancements. Model behavior is not fundamentally altered. Numbering for minor releases increments the second element of the version number. Examples: "CRTM 2.1.0", "CRTM 3.2.0". Minor releases will be scheduled well in advance and will be coordinated across all developers. However it is not required that every subgroup schedule minor releases at the same time. This allows for quicker turnaround of minor feature enhancements by individual development subgroups.
- A "patch release" contains bug fixes. No new features are added. Coordination and scheduling are identical to minor releases, except that less advance notice is required. This allows for quick turnaround of bug fixes by individual development subgroups. Numbering for patch releases increments the third element of the version number. Examples: "CRTM 2.1.1", "CRTM 2.1.2"
- Release numbering and designation of the type of releases is primarily the responsibility of the CRTM Working Group.
- For all releases, the full CRTM test suite must be run on all supported platforms.

- For major releases, the CWG will run the full CRTM test suite on all supported platforms.
- For minor and patch releases, the development subgroup that has requested the release is responsible for running the full CRTM test suite on all supported platforms.
- The CWG is responsible for content of the CRTM test suite.
- No policy is perfect. Variations from this policy will be allowed with prior consent from all affected subgroups.

7 Policy Consequences

It must be possible for a development subgroup to make a minor release or a patch release without seriously impacting development schedules in other subgroups. Therefore, the main trunk of the repository must always contain source code that is suitable for release, or nearly so.

Development that is not yet ready for release must be done in a repository branch or in a checked-out working copy. The use of repository branches for any non-trivial development is strongly encouraged.

DRAFT

Acknowledgement

Many of the guidelines and policies in this document were adapted from those for the original WRF Source Code Control Procedures available at:

http://box.mmm.ucar.edu/wrf/WG2/allreg/WRFSourceCodeControlPolicies.html

References

[1] Collins-Sussman, B., B.W. Fitzpatrick and C.M. Pilato. *Version Control with Subversion.* O'Reilly Media, 2004. Latest updates available online at http://svnbook.red-bean.com/.

[2] Mason, M. *Pragmatic Version Control using Subversion.* The Pragmatic Bookshelf, 2005.

[3] van Delst, P. CRTM Fortran95 Coding Guidelines. Technical report, JCSDA, 2008.