from Baseline Software Management Plans

DRAFT last revised 14 January 2010

LSST Data Challenge 3 Software Configuration Management Plan V 0.1

1. Introduction

This Software Configuration Management Plan realizes the policies and procedures defined in the LSST Software Configuration Management Guidelines for the Data Challenge 3 (DC3) baseline.

1.1 Purpose of the document

In particular, the DC3 Configuration Management Plan moves from generalities to the concrete specification of the CM Items managed during DC3.

1.2 Scope

The DC3 baseline will be implemented based on the DC3 objectives specified in the DC3 Software Release Project Plan. The DC3 lifecycle will be managed by the set of DC3 Software Release Plans.

1.3 Definitions, acronyms and abbreviations

Additional Software Configuation Management definitions, acronyms and abbreviations specific to DC3 are specified below

- **svn branch**: a line of development that exists independently of another line, yet still shares a common history. A branch always begins existence as a copy of another branch and thence progresses independently. Branches are frequently used to maintain separate maintenance lines for release products from the current development line.
    - **trunk** - contains mainline development for Product Releases.
    - **tags** - contains release versions for both Product Releases and any Maintenance Product Releases.
    - **maint** - contains maintenance development for an existing Product Release.
    - **tickets** - contains private development on new features or Problem resolution which will be merged back into 'trunk' or 'maint'
- **revision number**: unique identifier associated with all changes made to the Subversion repository on successful completion of a 'commit' request. The 'commit' saves all modifications in a local working copy of a Subversion source tree into the Subversion repository.
- **tag**: is svn nomenclature for an alias to a specific svn revision which can then be used to extract the repository contents as they were at that revision.

- **trac**: software code and modification tracking tool.
    - **changeset number**: 1 for 1 correspondence with a Subversion revision number.
    - **ticket**: an integer number assigned to a Change Request.

- **twiki**: a window based document management system embedded in the **tract** tool which maintains change history.

1.4 References

No additional references.

1.5 Management

1.5.1 Chain of Responsibility

No deviation from the Lsst Software Development Plan

1.5.2 Timeline

Refer to DC3 Schedule for the DC3 project timeline.

1.5.3 Applicable Policies, Standards and Procedures

See LSST Software Configuration Management Guidelines

1.6 Support Tools

See LSST Software Configuration Management Guidelines

---

2. Configuration Management (CM) Control

2.1 CM Policy

See LSST Software Configuration Management Guidelines

2.2 CM Procedure

The procedure is defined in the Technical Change Management document's section on 'Initiating a Software Policy/Procedure/Design? Change'.

---

3 Module (Package) Item Configuration Management

The overarching Lsst Software Configuration Plan labels the smallest configurable unit as a 'Module'. However, The Data Management Subsystem has consistently used the label 'Package' for that entity throughout all their documentation. Since this is a Data Management Subsystem Plan, the term 'Package' will hereafter be used.

Packages are uniquely identified based on the branch in which they exist:

- in **trunk** branch: package name and svn revision number;
- in **tickets** branch: package name, ticket number, and svn revision number;
- in **maint** branch: package name, module version number, and svn revision number;

- in **tags** branch: package name and module version number or package name and svn revision number.

For purposes of Configuration Management, we are only concerned with a package's **tags** directory and uniquely identifying it using the package name and module version number.

3.1 Specification

3.1.1 Contents

The Package Configuration Management Item, known hereafter as Package, is composed of:

- primary software
- test software
- build configuration software

and optionally:

- documentation
- demonstration software
- data.

The comprehensive list of packages is available in the DC3 Package List.

3.1.2 Identifier

The syntax for Package identifiers, also known as Package version numbers, is defined in On Version Numbers.

Given a Package identifier of the form: <version>**.**<revision>**.**<interim>,

- **<version>** marks a major overhaul of a Package.
  On update, <pkgVersion> reset to: <version+1>.0[.0];
- **<revision>** marks an upgrade to major version.
  On update, <pkgVersion> reset to: <version>.<revision+1>[.0];
- **<interim>** marks an upgrade from a buggy revision.
  On update, <pkgVersion> reset to <version>.<revision>.<interim+1>.

The Package identifier is updated when the configuration source file (i.e. <package>/ups/<package>.table) containing the Package version number is committed to the Subversion repository. The Package Item is created when the Package's 'svn' directory is copied into the **tags** directory.

The Package identifier is also represented by <pkgVersion> in this document.

3.1.3 Modification Authorization

The Package's Responsible Developer, or delegate, is the only person authorized to update the Package version number and archive the new version. The list is maintained in DC3 Package list.

3.1.4 Conditions To Satisfy Prior to Modification

The Package to be modified in response to a Change request is located either:

- in the repository within the **<package>/trunk** branch for current development; or
- in the repository within the **<package>/tags/<pkgVersion>** branch for maintenance on released baselines.

Development, either new enhancements or bug fixes, is associated with a Trac ticket and is developed in a Subversion branch identified by that ticket number, i.e. **DMS/<dir>/<package>/tickets/<ticket>**.

Three exceptions to the above Rule allow changes to be made directly on the Trunk branch:

- initial module development may occur directly within the **trunk** directory;
- trivial changes, taking < 30 lines and not modifying the API, may be done directly within the Trunk branch without Module Guru permission;
- trivial changes, taking < 50 lines and modifying the API, may be done directly within the Trunk branch if the Module Guru approves.

3.1.5 Update Procedure

Refer to Releasing LSST Packages for information on when, how, and by whom a new Package version is created.

3.2 Control

3.2.1 Build Procedures

The Module contains **scons** build configuration files which are used to build the derived products (executables, libraries, documentation, etc) of the individual module.

3.2.2 Release Procedures

Individual packages may be individually released to developers during the development cycle.

Packages are generally not released to non-developers individually; they are usually bundled together in the Release Product baseline for distribution.

Refer to Releasing LSST Packages for information on release procedure for a new Package version.

3.2.3 Archiving

The Package is archived when it is committed to the **tags** Subversion branch. As long as the repository exists, any and all changes to the repository endure and can be retrieved.

3.2.4 Disaster Recovery Plan

Recovery of an archived Package simply requires an **svn checkout** of the appropriately tagged package from the **tags** branch.

3.2.5 Traceability of Package Modification

Trac maintains a perpetual record of changesets acting on the Subversion source repository. Given any source file, its lifelong change history can be unrolled. Since Package versions are maintained within the source configuration file, Package change history can also be unrolled.

Trac provides a convenient interface to view changeset history on a per file basis.

---

4 DC3 Release Item

The DC3 Release Item, known hereafter as the DC3 Release, includes the source and configurations necessary to reproduce the DC3 distribution products and the logs and reports generated during the testing and build of those products.

4.1 Specification

4.1.1 Contents

The version id, aka <relVersion>, for the DC3 Release is initialized at '3.0'; any software updates to the initial DC3 Release will increment the version id as appropriate.

The DC3 Release CI contains the following items; their physical locators are also included:

- Within DMS SVN archive
  - ♦ Metadata identifying each tagged package included in the DC3 Product.
    - ◊ DMS/release/<relVersion>/pkg/<tag>
  - ♦ Metadata identifying the 3rd party tools used when building the DC3 Release on each supported HW/SW platform.
    - ◊ DMS/release/<relVersion>/<platform>/<toolList>
  - ♦ Metadata identifying the hardware setup on each supported HW/SW platform.
    - ◊ DMS/release/<relVersion>/<platform>/<hardware setup>
  - ♦ Metadata identifying the operating system characteristics on each supported HW/SW platform.
    - ◊ DMS/release/<relVersion>/<platform>/<os setup>

- Within a mass storage system
  - ♦ The Integration Test (IT) Configuration Item
    - ◊ <loc>/Release/<relVersion>/IT/<pkg>/<pkgVersion>/<platform>/<pkg>_<pkgVersion>_<tes
    - ◊ impl detail: here shown as a tar bundle
    - ◊ use no debug options on build
  - ♦ The final build logs
    - ◊ <loc>/Release/<relVersion>/build/

TBD Where's Ray Plante's doco?

- Within the Lsst Software Distribution archive
  - ♦ The executables, libraries, and documentation comprising the DC3 software distribution.
    - ◊ These products will be packaged for distribution using Ray Plante's DM Distribution process.

- Within LSST Docushare system
    - The DC3 Release Report describing the status of the DC3 Release and, possibly, the authorization to distribute the derived executables, libraries and documentation.
        - Collection-1339, also known as 'Data Challenge 3' will contain, on DC3 completion, the document: 'Release Report'.

Note that the derived products of the packages: libraries, executables, and documentation, will be regenerated should recovery of an installed Release Product derivative be necessary so they do not require permanent archival.

Note that System Tests and Acceptances Tests will not be done for DC3.

4.1.2 Identifier

DC3 Release CI identifiers are of the form **version.revision.interim**
where

- **revision** and **interim** are optional.

and where

- **version** marks a major benchmark baseline.
  On update, <relVersion> reset to: <version+1>[.0[.0]]
- **revision** marks a planned upgrade to major version.
  On update, <relVersion> reset to: <version>.<revision+1>[.0]
- **interim** marks an upgrade from a buggy version or revision.
  On update, <relVersion> reset to <version>.<revision>.<interim+1>

The components of the <relVersion> triplet must each be integers. Optionally, use _ instead of **.** as separators.

version.revision.interim is also known as <relVersion> when referencing a DC3 Release in this document. The release CI identifier is created when the build configuration metadata is copied to **DMS/releases/<relVersion>/baseline.cfg**.

4.1.3 Modification Authorization

The Release CI will be created by the System Release guru after the appropriate fitness validation has successfully completed.

4.1.4 Conditions to Satisfy Prior To Acceptance

- All packages comprising the Release Product must pass their unit and integration tests successfully;
- The Release guru must sign-off on the baseline verification and validation.

4.1.5 Update Procedure

- Create the new baseline <relVersion> SVN branch: **DMS/releases/<relVersion>**.
- Capture metadata defining
    - the packages and their versions which comprise the new baseline Release Product;
    - the tools and their versions necessary to build the new baseline Release Product;
    - the hardware platform(s) used to build the Release Product;

- Ensure baseline is ready for release by
    - ♦ creating the validation test CIs (i.e. run the tests);
    - ♦ ensuring the validation tests successfully complete within defined error tolerance; and
    - ♦ having the Release guru verify the release's adequacy and formally acknowledge that acceptance.
- Notify the Data Management System Manager that a new baseline as been generated and is ready for distribution.

4.2 Control

4.2.1 Build procedures

4.2.1.1 Create new Product Release baseline from "trunk" development

Assumption

- the tagged package directories are waiting for Integration testing

Procedure

- If the **release** branch has not been created for the new baseline, create it

```
svn mkdir DMS/release/<relVersion>
```

- Create the configuration table (baseline.cfg) defining all packages and their versions used in new Release baseline.
- Save the table in the **releases** archival directory

```
cd <localWorkingDirectory>
svn co DMS/release/<relVersion>
cp baseline.cfg <relVersion>/baseline.cfg
svn commit –m <add baseline configuration file> DMS/release/<relVersion>/baseline.cfg
```

- Setup environment to test complete Release Product build
- For each package in the baseline:
    - ♦ Perform unit test;
    - ♦ If failure,
        - ◊ exit this procedure,
        - ◊ fix package,
        - ◊ tag it, then
        - ◊ resume this procedure from beginning.
- Do Integration test using procedure in Section 4.2.1.2. "Perform Test on proposed baseline"
- Ensure baseline is ready for release by
    - ♦ verifying the multitude of tests successfully complete within defined error tolerance; and
    - ♦ ensuring the Release guru formally signs-off on the release.
- If this procedure was used to create a maintenance Release Product baseline, determine if bug fixes made in the **maint** branch should be migrated into the development **trunk** branch.

4.2.1.2 Perform Test on Proposed baseline

Perform the appropriate test on the <relVersion> baseline

- If test fails,

- create Problem Ticket;
- exit procedure until Problem resolved; then
- resume from the beginning of the 'Create Product Releases baseline' procedure.

- Else
  - save test build log and reports as per the Test Item CI; and
  - capture Test CI locator metadata.

4.2.2 Release procedures

TBD Where's Ray Plante's doco?

The Release Product distribution will be generated using the DM Distribution process described in Ray Plante's DM Distribution procedure. The new DC3 distribution will be announced in the LSST-data mail group.

4.2.3 Archiving Validation Artifacts

The following validation artifacts should be retained:

- Integration Test CI; and
- Release Authorization statement completed by Release Guru.

The Test CI should be archived as specified in this document under the individual <test type> Test CI sections. The Release Authorization should be archived in LSST Docushare Collection-1339.

All validation artifacts should be retained for life of the project.

4.2.4 Disaster Recovery Plan

TBD Where's the LSST 3rd party software repository for current and obsolete 3rd party products?

Recovery of previous Release Products is based on the premise that the derived components of Release Products will be regenerated using the configuration metadata, source, and supporting 3rd party tools originally used to build the Release Products. Those components of a build are all archived, either in the subversion repository or the LSST 3rd party tool archive (ToBeCreated?), so the recovery of Release Products is straightforward.

4.2.5 Traceability of Release Products Modifications

The individual packages which are combined to build a Release Product are maintained under **svn** which maintains a full modification history. Any changes to the baseline release branch are thus traceable.

5. Test Items

5.1 Unit Test Item

Unit Tests validate that a discrete component, or an aggregation of discrete components into a subsystem, work according to specifications.

For DC3, Unit Test Items will not be archived. This does not mean, though, that Unit Tests will not be run nor that their successful completion is not required.

5.2 Integration Test Item

Integration tests validate that multiple subsystems work together correctly. For DC3, which operates totally within the Data Management domain, the Integration Tests occur between subsystem-level components in the Domain model.

5.2.1 Specification

5.2.1.1 Contents

5.2.1.2 Identifier

- IT_<integrationPackage>_<tag>_<datetime>

5.2.1.3 Modification Authorization

5.2.1.4 Conditions to Satisfy Prior to Modification

5.2.1.5 Update Procedure

5.2.2 Control

5.2.2.1 Build procedures

5.2.2.2 Release procedures

5.2.2.3 Archiving Validation Artifacts

- Describe the procedures for archiving test output and status.
- Define the retention period
    - ♦ trunk releases until next DC or major production release?
- List all artifacts to be retained

5.2.2.4 Disaster Recovery Plan

- Describe the procedure to recover the Item from either archival storage or regeneration.

5.2.2.5 Traceability of Item Modification

- Describe the procedures for keeping an audit trail of changes.

5.3 System Test Item

System Tests validate that software works end-to-end through all system paths.

For DC3, which operates totally within the Data Management domain, the Release Software System Tests are identical to the Integration Tests between subsystem-level components in the Domain model.

5.4 Acceptance Tests

For DC3, which operates totally within the Data Management domain, the Release Software Acceptance Tests are identical to the Integration Tests between subsystem-level components in the Domain model.

6 Documentation

Documentation included in this section is not maintained within the Subversion repository.

6.1 Specification

6.1.1 Contents

6.1.2 Identifier

6.1.3 Modification Authorization

6.1.4 Conditions to Satisfy Prior to Modification

6.1.5 Update Procedure

6.2 Control

6.2.1 Build procedures

6.2.2 Release procedures

TBD

- Describe the procedures for the release of documentation.
    - ♦ Currently, all documents relating to release products are maintained within the **subversion** repository. Only documents relating to LSST software development are maintained outside **subversion** in either **twiki** or **docushare**.
    - ♦ possibly some of these documents are relevant to new sites wanting to implement contributed code.

6.2.3 Archiving Validation Artifacts

- Describe the procedures for archiving test output and status.
    - ♦ For each baseline, copy all the twiki-based documentation to LSST Docushare.
        - ◊ possibly use twiki macro 'Combine' to build single documents out of document trees.
        - ◊ possibly convert to 'pdf' format (via twiki output macro)
- Define the retention period
    - ♦ Forever
- List all artifacts to be retained
    - ♦ Plans: SQA, CM, V&V, baseline EA model...
    - ♦ Policies: SDP, ...
    - ♦ Standards: C++ and Python Language, Doxygen Standard, CCB decisions,...
    - ♦ Procedures:

6.2.4 Disaster Recovery Plan

Describe the procedure to recover the Item from either archival storage or regeneration.

TBD

- First line of defense: Docushare backups
- second line of defense: Twiki backups

6.2.5 Traceability of Item Modification

- Describe the procedures for keeping an audit trail of changes.

All documents maintained in **twiki** have complete changeset histories; as do all documents maintained in **docushare**.

Trac maintains a perpetual record of changesets acting on the Subversion source repository. Given any source file, its lifelong change history can be unrolled. Since Package versions are maintained within the source configuration file, Package change history can also be unrolled. Finally, since Release products are built from a specific tag which is copied from a specific branch, the lifelong change history of a tagged branch is also available.

Trac provides a convenient interface to view changeset history on a per file basis.

7 Model CM

Covered in LSST Software Configuration Management Plan