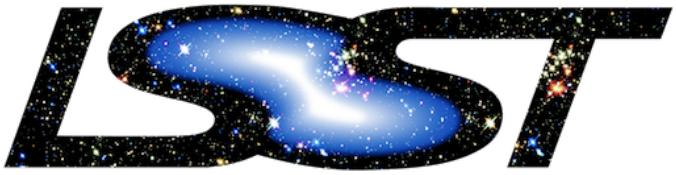

LARGE SYNOPTIC SURVEY TELESCOPE

## Large Synoptic Survey Telescope (LSST)
## Data Management

# DM Release Process

**Gabriele Comoretto**

**DMTN-106**

**Latest Revision: 2019-02-05**

**DRAFT**

## Abstract

Release procedure applicable to all Data Management SW products.

# Change Record

| Version | Date | Description | Owner name |
|---|---|---|---|
| | 2019-02-04 | DM Release Process | Gabriele Comoretto |

# Contents

**1 Introduction** **1**
1.1 Applicable Documents . . . . . 1

**2 Definitions** **2**
2.1 Software Product . . . . . 2
2.1.1 Software Metackages . . . . . 2
2.2 Dependencies . . . . . 2
2.3 Software Release . . . . . 2
2.4 Software Binary Package . . . . . 3
2.5 Distribution . . . . . 3
2.6 Versioning . . . . . 4

**3 Change control** **5**
3.1 Issue Management . . . . . 5

**4 Release Note** **6**

**5 Software Release Procedure** **7**
5.1 Development . . . . . 7
5.2 Daily and Weekly builds . . . . . 7
5.3 Announcement . . . . . 7
5.4 Consolidation . . . . . 8
5.5 Release Candidates Validation . . . . . 9
5.6 Final Release . . . . . 10

**A References** **11**

**B Acronyms used in this document** **11**

# DM Release Process

## 1 Introduction

The scope of this document is to provide a release procedure technicalities valid for all Software Products in the Data Management LSST subsystem. The procedure as presented here can be tailored accordingly to specific SW products needs.

### 1.1 Applicable Documents

When applicable documents change a change may be required in this document.

LDM-148 DM Architecture
LDM-294 DM Project Management Plan

# 2 Definitions

Following definitions are considered in the scope of this document.

## 2.1 Software Product

A release is made of a SW product documented in the product tree. A SW Product should correspond to a single repository (git package). In the case of lsst a SW corresponds to multiple git packages but the single repository can be mimicked using a *metapackage*.

### 2.1.1 Software Metackages

The *metapackage* depends from the git packages that are considered part of the SW product.

In order to clearly identify all git packages included in a SW product, they have to be listed as direct dependencies in the metapackage.

Each git software package has to be included only in one metapackage.

## 2.2 Dependencies

Considering the fact that a SW Product is identified with a metackage, there are two types of dependencies in a git package:

- dependencies to metapackage, to resolve Github packages released in other SW products (metapackages)
- dependencies to a Github package contained in the same metapackage, and therefore part of the same SW product

## 2.3 Software Release

A software release is identified by a TAG in the SW repository and it is documented with a software release note. The TAG is created on a release branch after manual checks on the last candidate.

All metapackages from which a SW product depends, need to be released before the SW product release is done.

## 2.4 Software Binary Package

A software binary package is a package containing executable binaries created building the SW contained in the release Tag. Binary packages can be created to support multiple platforms (such as linux, osx, windows) if required.

## 2.5 Distribution

A distribution is a collection of binary packages to be deployed together. A distribution can be used for different purposes:

- make available software releases for operations
- test (integration, validation, operation rehearsals) software releases
- provide software releases to external collaborators.

## 2.6 Versioning

Major, Minor, Patch.

Semantic Versioning.

# 3 Change control

The DMCCB is in charge of the major and minor release planning. The DM Release Plan, LDM-564, will provide the expected reschedule, based on P6 milestones.

Changes to the release plan need to be proposed to the CCB using RFC Jira issues.

The DMCCB is also in charge of approving patch release requests. A patch release has to be requested to the DMCCB using RFC Jira issues, specifying which which fixed, DM issues, are requested to be included in the CCB.

This process is documented in LDM-294, section 7.4.

## 3.1 Issue Management

Issues to be included in a release shall be added as blocking to the release issue.

The field *Fix in Version(s)* should be used. This requires changes in the DM Jira project.

# 4 Release Note

The release note documents the content of a release.

Following information are provided:

- Installation instructions. To be provided manually by the release responsible, usually the product owner.
- List of jira issue included in a release. This information can be extracted from Github. Completed Epics will be highlighted.
- Narrative section describing the content of the release. To be provided manually be the release responsible, usually the product owner.
- Technical information like tag in github, dependencies, binary packages, etc. To be provided automatically.

# 5 Software Release Procedure

This release procedure has been derived generalizing the *Stack release playbook* **?**.

## 5.1 Development

Development activities are not part of the release process, but are the starting point for a stable and reliable master branch in all Github software packages.

Development activities follows the (LSST Data Management). All changes are done on tickets branches and reviewed using the Pull Request mechanism before merging to master. Ticket branches are removed once merged.

Each time a change is merged into master following activities should be performed:

- continuous integration build of the Github software package (SW product)
- if unit tests pass, generate binary packages
- build downstream dependencies: CI build on SW products that depend from the newly build SW product.

## 5.2 Daily and Weekly builds

Daly and weekly are done in order to have fixed reference in each package. They can be used as a starting point for a development activity or, more relevant for this document, as starting point for a release.

Daily builds, do not generate a tag in the git repository. Weekly builds corresponds to a tag in the Github repository. Despite for Github these builds are considered releases, from a release management point of view they are not.

## 5.3 Announcement

The preparation of the next release is announced using a community post.

This has to be done few days before the planned starting of the release. All contributors can provided feedbacks and the DMCCB can take corrective actions in case there are still

outstanding issues to be included in the release.

### 5.4 Consolidation

This is the first step of the release process. It is done when:

- all issues that are suppose to be included in the next release have been implemented and merged into master.
- a weekly build has been completed successfully.

Last weekly build is the starting point for a release. In other words: the first release candidate of a release is produced starting from a weekly build.

The creation of the first release candidate is done using the Jenkins job (if available) or executing manually:

- creating the release branch from the last weekly build (codekit, functionality still to be added, branches can be created manually)
- creating a release candidate (codekit)
- creating the binaries packages
- creating the distribution package

At today, the binaries packages are stored using eups in the eups binaries repository. Eups package is also used to orchestrate dependencies between packages.

This imply that the manual sequence to use is slightly different and more complex. it is therefore recommendable to use an automatic process implemented in Jenkins.

Together with the creation of the release branch on the git packages included in the software product, other artifacts may need to be branched, and possibly have a first release candidate also:

- documentation: release note are usually part of a documentation endpoint, similar to pipelines.lsst.io (for science pipelines). In order to consolidate all information relevant for the new release, a corresponding branch need to be created.

- conda environment, is relevant
- others. Still referring to the science pipelines case, the *newinstal.sh* Github package need to have a corresponding branch

## 5.5 Release Candidates Validation

The release candidate need to be validated, ensuring that it is consistent with what expected.

In an ideal case, a test campaign following a specific test plan should be conducted, demonstrating that the release candidates, and therefor the forthcoming release, is behaving as expected.

Practically speaking, the validation in many cases can just be:

- installation/configuration of the binaries packages, or distribution image; usually done manually
- inspect of the installation, that all expected files and configuration are there
- execution of some demo package available case by case depending of the SW product
- try some use cases in order to prove that the release candidate is behaving properly

In case problems are found, a DM issue need to opened in Jira. This issue shall follow the proper development cycle, been first implemented in a ticket branch, reviewed and merged to master. Once the issue has been proved to work on master can be backported to the release branch. Backporting mechanism has to be documented in the developer guide, however it is here summarized:

- given an issue DM-XXX fixed on a ticket branch *tickets/DM-XXX* and merged to master (the ticket branch shall not be deleted in this case, until the backporting is concluded)
- backport using

```
git rebase --onto <RELEASE_BRANCH>
```

  - Note that this will move the original branch based on master to be based on the the release branch. In case you want to keep the old branch, a new branch has to be created from the original ticket branch. Ideally this can be done using a new DM Jira issue, where the porting is requested.

- open a PR from the ported ticket branch to merge into the release branch
- merge the ported branch into the release branch when the PR is approved

Note that porting mechanism is a development activity, under the responsibility of the development team, and therefor need to be documented properly in the developer guide and removed from this document.

In a special case, that an issue cannot be fixed on master, the ticket branch can be opened based on the release branch.

Once the fixes have been implemented in the release branch, a new release candidate can be created, and then verified.

The porting can be applied also in case that an issue, implemented on master after the release branch has been cut, has to be included in the release.

The DMCCB has to overview the issues backported on the release branch, and take corrective actions if needed, since backporting may required a considerable use of development resources or delay in the final release availability.

## 5.6 Final Release

Once a final release candidate has been identified, the final release can be created.

# A References

## References

**[LDM-148]**, Lim, K.T., Bosch, J., Dubois-Felsmann, G., et al., 2018, *Data Management System Design*, LDM-148, URL https://ls.st/LDM-148

LSST Data Management, LSST DM Developer Guide, URL https://developer.lsst.io/

**[LDM-564]**, O'Mullane, W., Economou, F., Jenness, T., Loftus, A., 2018, *Data Management Software Releases for Verification/Integration*, LDM-564, URL https://ls.st/LDM-564

**[LDM-294]**, O'Mullane, W., Swinbank, J., Jurić, M., DMLT, 2018, *Data Management Organization and Management*, LDM-294, URL https://ls.st/LDM-294

# B Acronyms used in this document

| Acronym | Description |
|---|---|
| CCB | Change Control Board |
| CI | Configuration Item |
| DM | Data Management |
| DMCCB | DM Change Control Board |
| DMTN | DM Technical Note |
| LDM | LSST Data Management (document handle) |
| LSST | Large Synoptic Survey Telescope |
| SW | Software (also denoted S/W) |
| TCAM | Technical Control Account Manager |
| TN | Technical Note |