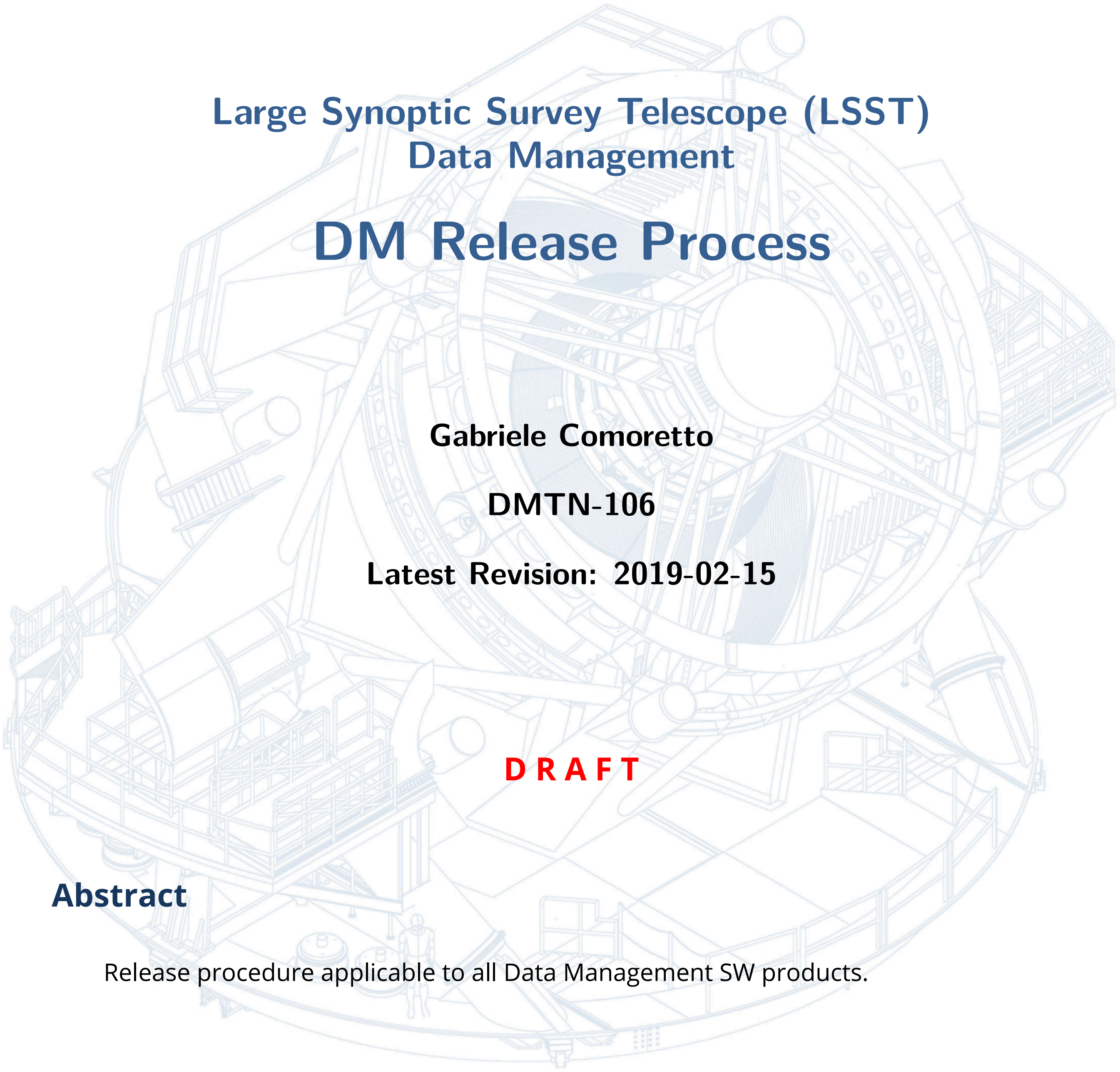
LARGE SYNOPTIC SURVEY TELESCOPE



Large Synoptic Survey Telescope (LSST)
Data Management

DM Release Process

Gabriele Comoretto

DMTN-106

Latest Revision: 2019-02-15

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

DM Release Process

1 Introduction

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

1.1 Applicable Documents

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 Software Product is a component of the subsystem (DM) product tree. A release is made on a SW product.

A SW Product should correspond to a single repository (git package). In the case of DM a SW Product is implemented in multiple git packages. A Github *metapackage* is used to identify a SW Products. All git packages of a SW product will be dependencies in a Github *metapackage* and released at the same time.

A Software product *lives* in a software repository. The LSST software repository is github.

2.2 Dependencies

Considering that a SW Product is identified by a *metapackage*, each package can depend from:

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

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

The dependency information is provided in each git package, in specific files, that may vary depending on the build system. It is also considered part of the source code.

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 release candidate.

The tag in the Github repository and the software release note should be sufficient for a developer to build the binaries and execute the software.

2.4 Binary Package

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

Binaries in general should be generated only once, and made available for their use by dependent software or for deployment.

The majority of DM software is using Eups for binary packaging. Other platforms can be considered, like for example *conda* or *pypi*. Software products implemented in java are using jar/war binary packaging. [1]

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 or commissioning
- test (integration, validation, operation rehearsals) software releases
- provide software releases to external collaborators.

DM software products are distributed using *docker*.

[1] So far there are a few technologies used to handle binary packages. It is recommended to assess and optimize them, converging to use only one. Conda seems to be so far becoming a standard.

2.6 Versioning and Naming

DM software is versioned using two digits separated by a ".". For example:

```
16.0
```

has been the last science pipelines release in 2016. The first number is the major version and the the second number is the minor version.

It is recommended to switch to Semantic Versioning, in order to improve dependency management.

2.6.1 Branch Naming

In order to clearly distinguish release branches from tags and other branches, following naming is suggested:

```
b.<M>.<m>
```

where **M** is the major version and **m** is the minor version.

3 Change control

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

Changes to the release plan need to be proposed to the DMCCB 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 fixes, DM issues, are requested to be included in the CCB.

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

3.1 Issue Management

The release is identified in Jira using a release issue.

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

It is recommended to use the field *Fix in Version(s)* provided by Jira. 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 person responsible for the release, usually the product owner.
- List of jira issues included in a release. This information can be extracted from Github. Completed epics will be highlighted in the first place. Other issues to be listed for documentation purpose.
- Narrative section describing the content of the release. To be provided manually be the person responsible for the release, usually the product owner.
- Technical information like tag in github, dependencies, binary packages, etc. To be extracted automatically from Github or other tools.

Once the field *Fix in Version(s)* has been introduced in the Jira DM project, it shall be made consistent with the release note (automatically).

5 Software Release Procedure

This release procedure has been derived from the *Stack release playbook* SQR-016.

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 follow the (LSST Data Management) developer guide. All changes are done on ticket 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, the 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.

At the moment, continuous integration is done prior to merge the changes into master, using the ticket branch. Binary packages (eups) and docker images for distribution are made available with the nightly and weekly builds (see next section 5.2).

5.2 Daily and Weekly builds

Daily and weekly builds are performed in order to have fixed references in time. They ensure periodically that no breaking changes have been introduced. 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 Github repositories. When a weekly build is done, a corresponding tag in the Github repository is created. Despite weekly builds are considered releases for Github, 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 release activity starts.

In this way, all contributors will be able to provided feedback, like for example additional issues expected to be included in the release. Base on the received feedback, DMCCB can take corrective actions such as delaying the release.

5.4 First Release Candidate

The first release candidate on a release is created when:

- all issues that are suppose to be included in it have been implemented and merged into master.
- a weekly build has been completed successfully.
- the completed weekly build has been proved to be a valid starting point for the release. Criteria for this can include regression, scientific performances, etc.

The weekly build is the starting point for the release process.

The creation of the first release candidate is done using the Jenkins job:

```
releases/official-release
```

with the following parameters:

- **SORUCE_GIT_REFS**: <weekly tag> (for example *w.2018.52*)
- **RELEASE_GIT_TAG**: <Release Candidate> (for example *v.17.0.rc1*)
- **O_LATEST**: false

If the Jenkins job is not available, the release candidate should be created 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

As of today, the majority of these steps are integrated in Jenkins jobs. This makes it impossible to complete the release process without the use of the continuous integration system.

5.5 Other Related Artifacts

Together with the creation of the release candidate, 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.
- environment, for example conda environment definition
- others; still referring to the science pipelines, the package where *newinstal.sh* is developed, need to have a corresponding branch

5.6 Release Candidates Validation

The release candidate needs 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 therefore the forthcoming release, is behaving as expected.

Practically, the validation in many cases can 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
- ask downstream users to act as beta testers.

For science pipelines, a characterization report is produced, in order to document the release outputs. This is a new document created for each major release.

5.7 Resolving Problems

In case problems are found during the validation, a DM issue needs to be opened in Jira. This issue shall follow the usual development process as described in the developer guide. The fix will be first implemented in a ticket branch, reviewed and merged to master. Once the fix has been proved to work, it 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)
- given a release branch created based on a release candidate, or on a weekly tag
- a specific backporting ticket DM-YYY has to be created, and a corresponding ticket branch based on DM-XXX issue
- backport is done from the backporting ticket branch, using the following command:

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

  - Note that this will rebase the backporting branch to the release branch.
- 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
- remove the ported branch and the original branch

Note that porting mechanism is a development activity, under the responsibility of the development team, and therefore needs to be documented 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.

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 backporting of issues to the release branch, and take corrective actions if needed. Backporting may require a considerable use of development resources or delay in the final release availability.

5.8 Additional Release Candidates

Once one or more issues have been fixed on the release branch, a new release candidate has to be generated using the Jenkins job *official-release* using following parameters:

- **SOURCE_GIT_REFS**: <BranchId> <Previous RC>. (for example *b.17.0 v17.0.rc1*)
- **RELEASE_GIT_TAG**: <new RC> (for example *v17.0.rc2*)
- **O_LATEST**: false

5.9 Final Release

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

This has to be done using the Jenkins job *official-release* as previously done for the release candidates, with the following parameters:

- **SOURCE_GIT_REFS**: <Final RC>. (for example *v17.0.rc2*)
- **RELEASE_GIT_TAG**: <Release Tag> (for example *17.0*)
- **O_LATEST**: true

6 Patch Releases

In the case that the DMCCB approves a proposed patch release, the process shall be:

- create the release branches from the available release tags on the Github packages impacted by the fixes
- backport the requested issues
- create a release candidate, for example *v17.0.1.rc1* using the the same approach explained above (5.8)
- validate the release candidate
- fix eventual problems found in the release candidate and repeat the last 2 steps until a valid release candidate is found
- create a final release as described above (5.9).

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-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
DM	Data Management
DMCCB	DM Change Control Board
DMTN	DM Technical Note
LDM	LSST Data Management (document handle)
LSST	Large Synoptic Survey Telescope
RFC	Request For Comment
SW	Software (also denoted S/W)