



Vera C. Rubin Observatory
Data Management

Docsteady Usecases for Rubin Observatory Constructions

Gabriele Comoretto

DMTN-178

Latest Revision: 2021-04-26



Abstract

How to use docsteady in the context of Rubin Observatory Construction project, including continuous document generation using Jenkins.

Change Record

Version	Date	Description	Owner name
1.0	2021-01-20	Document Issued	Gabriele Comoretto

Document source location: <https://github.com/lst-dm/dmtn-178>

Contents

1 Introduction	1
2 Installation	1
3 Execution	1
3.1 Test Specification Generation	2
3.2 Test Plan and Report Generation	3
3.3 Verification Element Baseline Generation	4
3.3.1 Sub-Components	5
3.4 Verification Control Document Generation	6
3.5 Components - Sub-systems	7
4 Development	7
5 Documentation Procedure	8
6 Continuous Integration	9
6.1 Authentication	10
6.2 Deployment of New Versions	10
A References	10
B Acronyms	10

Docsteady Usecases for Rubin Observatory Constructions

1 Introduction

This is intended to be a short guide to the installation and use of docsteady. See DMTN-140 for a high level overview of the process and the logic implemented here.

This documents it superceded by `docsteady.lsst.io`

2 Installation

Create a conda environment based on the **docsteady** conda package:

```
conda create --name docsteady-env docsteady -c lsst-dm -c defaults --override-channels
```

Ensure that the conda configuration file `.condarc` does not include the conda-forge channel.

To use docsteady, activate the environment as follows:

```
conda activate docsteady-env
```

This environment will provide all dependencies that are required to run docsteady.

It is recommended to use the provided conda environment also for development activities, see section 4.

3 Execution

It is recommended to provide credentials by setting up the following environment variables:

- `export JIRA_USER=<jira-username>`
- `export JIRA_PASSWORD=<password>`

otherwise, it is required to specify them from the command line using the `--username` and `--password` options. In case credential options are omitted and no environment variables are defined, username and password will be prompted interactively.

Personal Jira credentials can be used. For CI purposes, a general set of credentials are available, as specified in section 6.1.

In order to execute any of the docsteady commands described in the following subsections, a conda environment providing docsteady shall be activated, as described in section 2.

Use `--help` to get the list of available options for each of the docsteady commands. In addition, existing documents generated using docsteady have a `.docugen` file with the command to be executed in CI, which can be used as an example.

3.1 Test Specification Generation

Test specifications are extracted using the Jira REST API. All tests cases included in a TM4J folder, including subfolders, are rendered in the same extraction. The folder organization in Jira should correspond to the major subsystem components and the MagicDraw model.

The syntax to extract a test specification is the following:

```
docsteady generate-spec "</tm4j_folder>" jira_docugen.tex
```

where `</tm4j_folder>` shall be replaced by the exact folder where the test cases are defined in the Jira Test Cases tab. For example, the command to extract the DM Acceptance test specification, LDM-639, is the following:

```
docsteady generate-spec "/Data Management/Acceptance|LDM-639" [jira_docugen.tex]
```

Note that:

- the output file *jira_docugen.tex* is optional in the execution of docsteady but required in this context in order to include the extracted information in a \LaTeX document (i.e. LDM-639.tex). If omitted, the docsteady output will just be printed in the terminal;

- the folder name in Jira includes at the end the test specification document handle. This is a best practice to use when organizing test cases in Jira since it helps to orient the user in the folder structure;
- an appendix with the traceability to the requirements is produced in the file *jira_docugen.appendix.tex*, to be included in the test specification TeX file.

See LDM-639 Git repository as an example.

3.2 Test Plan and Report Generation

Important: before extracting a test plan and report using docsteady, the corresponding document handle has to be added in the Document ID field in the Jira test plan object. This ensures that the Verification Control Document will include this information.

The following command extracts a Test Plan and Report using Jira REST API:

```
docsteady generate-tpr <LVV-PXX> <file.tex> [--trace true]
```

Where:

- LVV-PXX is the TM4J object that describes the test campaign, for example LVV-P72;
- file.tex is the Test Plan and Report tex file where the document will be rendered, for example DMTR-231.tex;
- --trace true (optional) generates an appendix with the traceability information.

Each TM4J test plan and related information in Jira is rendered in a different Test Plan and Report document, the filename of which usually corresponds also to the document handle in DocuShare.

The generated file can be built directly into the corresponding pdf, however additional files are required:

- Makefile

- appendix.tex
- history_and_info.tex

When creating the Git repository using **sqrbot-jr**, all the required files should already be present. See SQR-006 for more information regarding sqrbot-jr.

In case you want to generate a Test Plan and Report for a different subsystem, not DM, you can use the namespace global option:

```
docsteady --namespace <NS> generate-tpr <LVV-PXX> <file.tex> [--trace true]
```

Valid namespaces are:

- SE: system Engineering
- DM: Data Management
- T&S: Telescope & Site

See SCTR-14 or DMTR-231 Git repositories as an example.

3.3 Verification Element Baseline Generation

Verification Elements (VE) are Jira issues in the LVV Jira project, of type Verification. They are categorized into Components (DM, SITCOM, etc) and Sub-Components.

A VE baseline document is extracted using REST API. All VE associated with a Jira Component or Sub-Component, if specified, are rendered in the same extraction.

The syntax to extract a VE baseline information is the following:

```
docsteady [--namespace <CMP>] baseline-ve [--subcomponent <SUBC>] jira_docugen.tex [--details true]
```


The information is saved in the specified `jira_docugen.tex` file. This file has to be included in a \LaTeX document, where the corresponding context about the Component and Sub-Component is provided.

The `--namespace <CMP>` option identifies the Jira component from which to extract the information. The parameter `CMP` shall correspond to the Rubin Observatory sub-systems. See subsection 3.5 for the complete list of components. If omitted, the DM component is selected by default.

The `--subcomponent <SUBC>` is optional. If omitted all verification elements of the specified component will be extracted. See 3.3.1 for the description of the DM subcomponents.

If the option `--details true` is provided, an extra technical note is generated, including all test case details.

See LDM-732 Git repository as an example.

3.3.1 Sub-Components

Ideally, Sub-Components are matched to the major products of a LSST/Rubin subsystem. They should also be mapped to the product tree defined in the MagicDraw model.

In DM, trying to find a good balance between details and practice, the following components have been defined, in agreement with the DM scientist leader:

- Science
- Service
- Network
- Infrastructure

For each of these subcomponents, a different VE baseline document is extracted.

3.4 Verification Control Document Generation

The extraction of the Verification Control Document is done using direct access to the Jira database and not using REST API access, like for all other test documents described above.

Since the access to the Jira database is possible only from the Tucson network, it is required to be connected via VPN. A direct access to the Jira database implies also that the username and password to use are different since credentials to access the Jira web interface or the REST API are not enabled to access the database. They are two different authentication systems. Therefore personal Jira credentials will not work with this docsteady command.

A special read-only user has been enabled in the Jira database, **jiraro**. Section 6.1 explains where to find the full credentials details.

For your convenience, the credentials can be specified in the following environment variables:

- `export JIRA_VCD_USER=jiraro`
- `export JIRA_VCD_PASSWORD` (see section 6.1)
- `export JIRA_DB=140.252.201.12`

otherwise, it is required to specify them from the command line using the options `--vcduser`, `--vcdpwd`, and `--jiradb`. In case credential options are omitted and no environment variables are defined, they will be prompted interactively. Note also that the Jira database IP address may change. Updated information are maintained in the vault specified in section 6.1.

The following command extracts all VCD information regarding **DM** and generates the file `jira_docugen.tex`:

```
docsteady [--namespace <COM>] generate-vcd --sql True jira_docugen.tex
```

When no `--namespace` is provided, the DM component is selected by default. The generated file **jira_docugen.tex** is meant to be included in `LDM-692.tex`.

In case you want to generate the VCD for a different LSST/Rubin Observatory subsystem, just

use the corresponding subsystem code configured in the Jira **component** field. See next subsection 3.5 for the complete list.

3.5 Components - Sub-systems

Follows the list of components configured for the Jira LVV project. Each component corresponds to a Rubin Observatory Construction subsystem.

- **CAM:** Camera
- **DM:** Data Management, the default component for all docsteady commands.
- **EPO:** Education and Public Outreach
- **OCS:** Observatory Control System
- **PSE:** Project System Engineering, used for Commissioning (SitCom)
- **T&S:** Telescope and Site

In case the subcomponent specified is "None", all VE without subcomponents will be extracted.

4 Development

Despite docsteady being a pure python tool, it depends on **pandoc**, which is a c++ compiled library available only as a conda package. It has been observed that any small change in the version of pandoc may lead to unexpected changes in the resulting \LaTeX format.

Therefore, in order to ensure the expected pandoc behavior, it is important to set-up the conda environment corresponding to the latest docsteady working version. The environment set-up is explained in section 2.

The docsteady source code is available at <https://github.com/lsst-dm/docsteady>.

To test changes done locally in the source code, use the following procedure:

- (if not already available) create the environment as specified in section 2
- activate the environment: `conda activate docsteady-env`
- clone docsteady repository and checkout a ticket branch
- do your changes
- install the updates in the docsteady-env environment: `python setup.py install`
- activate the same docsteady-env environment in a different terminal to test the new changes
- once the changes are OK, commit them in the repository and open a PR for merging the branch to master

5 Documentation Procedure

The autogeneration of documents may become very confusing if not done in a programmatic way. Please consider the DM documentation approach as a guideline, summarized here.

- Create a document handle in DocuShare
- Use the document handle to create a repository in GitHub using `sqrbot-jr`, which will also create the corresponding landing page in `lsst.io`
- Configure the continuous integrations described in the next section 6.
- Render the document to a ticket branch, or to the **jira-sync** special branch. Never auto-generate the document directly to master
- Ensure that the document is correctly published in the corresponding LSST The Docs landing page and that everybody who is interested can access it.
- Create a GitHub Pull Request to let contributors and stakeholders comment on the changes.
- When a set of activities are completed, and all comments have been addressed, merge the branch/PR to master.

- In case the special **jira-sync** branch is used, after merging it to master, delete it and recreate from the latest master. Documentation tags corresponding to official issues of the document in DocuShare can also be done in the jira-sync special branch.

6 Continuous Integration

The real added-value of this approach, is the capability to auto-generate continuously the documents from Jira. This is done using the Jenkins service available at:

<https://lsst-docs-ci.ncsa.illinois.edu/jenkins/>

This service is at the moment behind the NCSA firewall, and therefore it requires a VPN to access and monitor the jobs. Authentication to this Jenkins instance is managed via OAuth. See next sub-section 6.1.

The rendered documents will be available in the corresponding GitHub repositories and LSST The Docs landing pages.

The docugen jobs are created using the seeds script defined at the following location:

https://github.com/docs-ci/docs-ci-seeds/blob/master/jobs/docugen_jobs.groovy

There should be no need to change this script, unless problems are found in its logic, or improvements need to be implemented The jobs are configured in the following YAML file:

<https://github.com/docs-ci/docs-ci-seeds/blob/master/etc/docugen.yaml>

The seeds job in Jenkins:

<https://lsst-docs-ci.ncsa.illinois.edu/jenkins/job/Service/job/seedling/>

will detect the changes in the YAML configuration file and create/update the the corresponding Jenkins docugen jobs.

6.1 Authentication

The access to <https://lsst-docs-ci.ncsa.illinois.edu/jenkins> is granted adding users to the **docs-ci** GitHub organization, managed by the DM Architecture team.

Two generic set of credentials to access the Jira REST API and the Jira database have been defined. These credentials are available at 1password.com, in the LSST-IT architecture vault, but not yet integrated into docsteady. In order to use these credentials, they have to be configured using environment variables, added as options from the command line, or entered when prompted, as specified in this technical note.

In addition, Jenkins DSL scripts use the `rubinobs-dm-admin` credential-ID defined in the *Manage Credentials* Jenkins section.

6.2 Deployment of New Versions

The `docs-ci` Jenkins application is hosted on NCSA Kubernetes cluster. The deployment files are available at <https://github.com/docs-ci/docs-ci-kubekt1>.

Continuous generation of documents is implemented by the `docugen` worker service, which is deployed as a `docsci/docugen-worker` docker image. This docker image is defined in the <https://github.com/docs-ci/docs-ci-workers-docker> repository in the `docs` folder.

A References

[DMTN-140], Comoretto, G., Guy, L.P., et al., 2020, *Documentation Automation for the Verification and Validation of Rubin Observatory Software*, DMTN-140, URL <https://dmtn-140.lsst.io/>

B Acronyms

Acronym	Description
API	Application Programming Interface
CAM	CAMera
CI	Continuous Integration
DM	Data Management
DMTN	DM Technical Note
DMTR	DM Test Report
EPO	Education and Public Outreach
IP	Internet Protocol
IT	Information Technology
LDM	LSST Data Management (Document Handle)
LSST	Legacy Survey of Space and Time (formerly Large Synoptic Survey Telescope)
LVV	LSST Verification and Validation
LaTeX	(Leslie) Lamport TeX (document markup language and document preparation system)
NCSA	National Center for Supercomputing Applications
OCS	Observatory Control System
PR	Pull Request
PSE	Project System Engineering
REST	REpresentational State Transfer
SE	System Engineering
SQR	SQuARE document handle
T&S	Telescope and Site
VCD	Verification Control Document
VE	Verification Element
VPN	virtual private network
YAML	Yet Another Markup Language