RPW Operation Centre

ROC Engineering Guidelines For External Users

ROC-GEN-SYS-NTT-00019-LES
Iss.02, Rev.00

Prepared by:	Function:	Signature:	Date
Xavier Bonnin	RPW Ground Segment Project Manager		12/10/2017
Verified by:	**Function:**	**Signature:**	**Date**
RPW RCS teams	Team Member #2		Dd/mm/yyyy
Approved by:	**Function:**	**Signature:**	**Date**
N/A	N/A		Dd/mm/yyyy
For application:	**Function:**	**Signature:**	**Date**
Name	Team Member #4		Dd/mm/yyyy

CLASSIFICATION **PUBLIC** ☒ **RESTRICTED** ☐

l'Observatoire de Paris LESIA
Laboratoire d'Études Spatiales et d'Instrumentation en Astrophysique

CNRS-Observatoire de PARIS
Section de MEUDON – LESIA
5, place Jules Janssen
92195 Meudon Cedex – France

Change Record

Issue	Rev.	Date	Authors	Modifications
01	00	08/10/2015	X.Bonnin	First issue
01	01	18/11/2015	X.Bonnin	Update the RCS general conventions Modify the Acronym list
02	00	12/10/2017	X.Bonnin	Second issue with major changes.

Acronym List

Acronym	Definition
ANT	Antenna
BASH	Bourne-Again SHell
BIA	BIAS unit
CDF	Common Data Format
ESA	European Space Agency
ESAC	European Space Astronomy Centre
ESOC	European Space Operation Centre
GIGL	Groupe Informatique Générale du LESIA
HFR	High Frequency Receiver
ID	Identifier
ISTP	International Solar Terrestrial Physics
LDPA	Lightweight Directory Access Protocol
LESIA	Laboratoire d'Etudes Spatiales et d'Instrumentation en Astrophysique
LFR	Low Frequency Receiver
NFS	Network File System
OS	Operating System
RCS	RPW Calibration Software
ROC	RPW Operation Centre

RODP	ROC Operations and Data Pipeline
RPW	Radio and Plasma Waves experiment
RSS	ROC Software System
SCM	Search-Coil Magnetometer
SSH	Secure SHell
SVN	SubVersioN
S/W	Software
TC	Telecommand
TDS	Time Domain Sampler
TM	Telemetry
TNR	Thermal Noise Receiver
URL	Uniform Resource Locator
VCS	Version Control System

Table of Contents

List of figures

List of figures

Aucune entrée de table d'illustration n'a été trouvée.

1 GENERAL

1.1 Scope of the Document

The present document addresses guidelines for the external users involved in the engineering activities of the RPW Operation Centre (ROC) [RD1].

In the framework of this document, the definition of external users covers the people that do not belong to the ROC engineering team in the LESIA site (Meudon, France). Especially, it includes the teams in charge of delivering to the ROC the RPW Calibration Software (RCS).

These guidelines are a tailored version of the ROC Engineering Guidelines (REG) defined in [RD10], and which concerns more specifically the ROC engineers of the LESIA.

1.2 Applicable Documents

This document responds to the requirements of the documents listed in the following table:

Mark	Reference/Iss/Rev	Title of the document	Authors	Date
AD1				
AD2				

1.3 Reference Documents

This document is based on the documents listed in the following table:

Mark	Reference/Iss/Rev	Title of the document	Authors	Date
RD1	ROC-GEN-SYS-PLN-00002-LES/1/3	RPW Operation Centre Concept and Implementation Requirement Document (CIRD)	Y.de Conchy, X.Bonnin	20/12/2016
RD2	SOLO-RPWSY-IF-55-CNES_0401(=EID-B).pdf/04/01	Experiment Interface Document Part B (EID-B) for RPW	RPW Team	21/12/2012
RD3	SOL.EST.RCD.0050/03/00	Experiment Interface Document Part A (EID-A)	Erik de Witt	03/08/2012
RD4	ROC-PRO-DAT-NTT-00006-LES/1/0	RPW Data Products	X.Bonnin	23/12/2016
RD5	ROC-TST-GSE-SPC-00017/2/1	Data format and metadata definitions for the ROC-SGSE data	X.Bonnin	14/10/2016
RD6	ROC-PRO-PIP-ICD-00037-LES/01/01 (draft)	RPW Calibration Software ICD	M.Duarte, X.Bonnin	12/10/2017
RD7	ROC-GEN-SYS-PLN-00015-LES/02/02	ROC Software Development Plan	X.Bonnin	27/09/2016
RD8	http://nvie.com/posts/a-successful-git-	A Successful Git Branching Model	Vincent Driessen	05/01/2010

	branching-model/			
RD9	https://git-scm.com/	Git	Git developer community	30/05/2017
RD10	ROC-GEN-SYS-NTT-00008-LES/1/3	ROC Engineering Guidelines	X.Bonnin	09/11/2016
RD11	https://pypi.python.org/pypi/maser4py	maser4py: Python 3 module for the MASER portal	X.Bonnin	20/03/2017
RD12	cdf364ug.pdf	CDF User's Guide	SPDF-GSFC	20/03/2017
RD13	ROC-TST-SFT-SUM-00027-LES/1/1	ROC-SGSE calibration software validation tool user manual	M.Duarte	06/05/2016
RD14	https://about.gitlab.com/	Gitlab	Gitlab team	10/2017
RD15	TBD	ROC Software User Manual	ROC developer team	
RD16	https://pypi.python.org/pypi/maser4py	Maser4py library	Maser developer team	10/2017

2 GUIDELINES FOR THE RPW CALIBRATION SOFTWARE (RCS)

2.1 Context

The RCS play a key role in the processing of the RPW science data. They are delivered to the ROC with a dedicated interface, described in the RCS Interface Control Document (ICD) [RD6], and which allowing the ROC pipelines to produce calibrated RPW data files at the LESIA site. In practice two pipelines: the ROC-SGSE and the ROC Operations and Data Pipeline (RODP) will run the RCS.

The general convention and the way the RCS must be delivered are presented in the next sections.

2.2 General convention

2.2.1 RCS descriptor file

According to [RD6], each RCS must be delivered with descriptor files, which contain information about the software and its inputs/outputs.

There must be one file per pipeline datasets. Especially, it means that the RODP and ROC-SGSE must use different RCS descriptor files.

2.2.2 RCS naming

RCS must be named using alphanumerical characters only. If required, only the underscore "_" or hyphens "-" must be used as separators.

The RCS name must be given in the `"identification.name"` attribute of the descriptor file, as described in [RD6].

2.2.3 RCS versioning

The RCS version must be a unique number sequence identifier "X.Y.Z", where "X" is an integer indicating the release (major changes, not necessarily retro-compatible), "Y" is an integer indicating the issue (minor changes, necessarily retro-compatible) and "Z" is an integer indicating a revision (e.g., bug correction).

The first stable release of S/W must have its major number "X" equals to 1, its minor number "Y" equals to 0 and its revision number "Z" equals to 0 (i.e., "1.0.0").

Software (S/W) preliminary versions (e.g., alpha, beta, etc.) must have their version number "X" equals to 0 and must not have a character as a prefix/suffix ("0.Y.Zb" for the 0.Y.Z beta version for instance).

In all cases, any change in the S/W must lead to update the version number.

The S/W version must be reported into the attribute `"release.version"` of the RCS descriptor, as described in [RD6].

2.2.4 RCS identification

Each RCS must be identified in the ROC pipelines by a unique identifier (ID). The ID naming convention must comply the definition of the attribute `"identification.identifier"` of the RCS descriptor, as described in [RD6].

The allocation must be done before the first delivery with the support of the ROC. Especially, the ROC must validate the ID, in order to avoid ID naming duplication in the pipeline database.

2.2.5 RCS input/output data identification

The teams in charge have the responsibility to define and report into the RCS descriptor, the list of the RPW datasets read/produced by their S/W. The way these datasets must be given is presented in [RD6].

Especially, the allocation of the corresponding RPW dataset ID [RD4] must be done in collaboration with the ROC, in order to avoid ID naming duplication in the pipeline database.

2.3 RCS delivery mechanisms

The teams in charge are responsible to deliver their RCS, following the procedure given in this section.

2.3.1 Context and convention about RCS-related files storage at the LESIA site

2.3.1.1 RCS Git repository

The ROC team uses Git [RD9] as a Version Control System (VCS) to store its S/W source codes on a dedicated Gitlab [RD14] server at the LESIA.

This system will be also used in order to ensure the delivery and archiving of the RCS instances run at the LESIA. Especially, each RCS team will have its own space - called "project" in Gitlab – on the ROC server to deliver and host its S/W.

The RCS project page, which includes a Git repository, must be created by a ROC system administrator only, inside the dedicated "ROC/RCS" Gitlab sub-group, and must be named as the RCS.

The URL of a given RCS project page on the ROC Gitlab server will hence be:

https://gitlab.obspm.fr/ROC/RCS/[NAME_OF_RCS]

Where [NAME_OF_RCS] is the name of the RCS.

NOTE: according to the Gitlab role definition, the ROC system administrators have the "manager" role permissions on the RCS repository, people of the RCS team have the "developer" role permissions and people from other RCS teams the "observer" role permissions.

The teams are free to use or not this project page to develop and test their RCS, however the following rules must be applied:

- **The "master" branch of the Git repository must only store versions of the S/W to be delivered to the ROC**. In another word, the "master" branch must not be used to commit development and/or test versions.
- **Any S/W version committed on the "master" branch of the Git repository must be tagged**. The name of the tag must be the version of the S/W release without the "V" prefix (e.g., "2.1.3"). In the very special case where a modification in the "master" branch is required, but does not need to update the version of the S/W, a fourth integer can be appended as a suffix to the tag name (i.e., "X.Y.Z.k", where "X.Y.Z" is the S/W version number, and "k" is the integer that can be incremented by

1, starting at 1). The latter case can typically occur when the RCS team needs to upload master CDF files, without upgrading the S/W source files.

If teams also use the Git repository to perform the S/W development and testing, the ROC developer team strongly recommends to protect the "master" branch, and to apply the Vincent Driessen's branching model [RD8].

In the case where teams use another VCS, or a different Git server, to store its RCS source code, a specific interface needs to be set up with the support of the ROC, in order to transfer the S/W into the ROC Gitlab server, with a minimal human intervention.

The expected content of the RCS Git repository is given in 2.3.3.

2.3.1.2 ROC DataPool Git repository

In addition to the RCS main Git repository, the teams needs an access to the ROC "DataPool" Git repository:

https://gitlab.obspm.fr/ROC/DataPool

This repository is used by the ROC to archive, among others, the CDF skeleton tables of RPW science datasets (see section 2.7.2), but must also be used by the teams to deliver the so-called "RCS E2E data package" (see section 2.5.2.3).

NOTE: the access to the ROC "DataPool" Git repository is restricted, and the "master" branch is protected (i.e., only the "master" branch owners can modify its content on the Gitlab server). Nevertheless the teams have the possibility to use the "rcs" branch to modify the content of the repository. Be careful since the "rcs" branch is shared between the RCS teams.

To get an access to the "DataPool" Git repository, people has to send an email to the roc.support@sympa.obspm.fr mailing list. The access is also possible using SSH-Key mechanism on demand.

2.3.2 Procedure to deliver a new RCS version

Each time a new version of RCS is available, the team in charge must:

1. Upload the content of this new version, also called "RCS delivery package", in the dedicated Git repository in the ROC Gitlab server. In practice, it should only consist of performing a push on the "master" branch (make sure that the pushed commit is tagged).
2. Inform the ROC and RCS teams, by sending a message to the roc.rcs@sympa.obspm.fr mailing list. The object of the mail must contain the S/W name, version and the purpose of the message as a prefix (e.g., "NAME -- VERSION -- NEW RELEASE DELIVERED:").

The ROC team will then launch first the verification tests, as explained in the section 2.5. If at least one test has failed, then the ROC will inform the team in charge. If all of the tests are passed successfully, then the RCS is integrated into the ROC pipelines and all the teams involved are informed via the roc.rcs@sympa.obspm.fr mailing list.

Note that if the changes in the new version of the RCS have impacts on the data products, the integration of the S/W might require upgrading also other RCS that use these data as inputs. In this case, the procedure might be longer if discussions are needed.

2.3.3 Expected content of a RCS delivery package

Any RCS release must be at least delivered with the following items:

- The S/W source code files
- The S/W descriptor file(s), as defined in [RD6].
- Any script or executable file used to run S/W on the ROC servers. This executable must comply the specification defined in [RD6].
- If required, a single S/W configuration file
- Any additional files required to run the S/W without error, i.e., master CDF files, calibration tables... And, if required, the scripts to activate/deactivate the execution environment [RD6].
- At least context files (e.g., README.rst, CHANGELOG.rst, etc.), placed at the root of the S/W directory
- If required, any additional libraries required to compile and/or to run S/W
- If required, any script or program used to compile S/W on the ROC servers (e.g., makefile or ant build file for instance).
- If it exists, any script or executable file that could permit to test the S/W execution.
- If required, any shell script required to set/unset the S/W environment variables.
- The corresponding up-to-date documentation. Especially a user manual describing in details the S/W in terms of organization, installation and use. This document must be compliant with the ROC documentation conventions defined in [RD10].

The organization of the S/W root directory may look like to the ROC proposal in [RD10].

2.3.4 Verifications to be done by teams before delivery

Before delivering a RCS, the team in charge shall ensure that:

- The RCS delivery package has the expected content (see previous section)
- S/W can be compiled and run on the ROC servers
- The interface with the ROC pipelines complies the specification in [RD6].
- The “RCS E2E test data package” - delivered separately (see the section 2.5.2.2) - is still useable with the new S/W delivery package.

In order to test and run the compilation/execution, the RCS teams can ask for opening dedicated accounts on the ROC development server. In the same way, they can use the validation tool [RD13] provided by the ROC - also available in the ROC development server - in order to test the interface compliance with the pipelines.

2.4 RCS deployment mechanism

2.4.1 RCS deployment procedure overview

When a team has informed the ROC that a new RCS version is available on the Gitlab server, a ROC system administrator will then retrieve, deploy and run an instance of this RCS version on the ROC production server, but using a dedicated testing version of the pipeline.

If the testing phase has successfully ended, the same deployment procedure will be repeated but with the production instance of the ROC pipeline.

For both the testing and production pipelines, the following common steps must be performed:

1. Installation of the RCS on the ROC server. It mainly consists of downloading/upgrading (i.e., cloning/pulling) the new version from the Git repository on the remote Gitlab server
2. Interface validation tests, where the compliance with the RCS ICD [RD6] is checked. This step is required to perform the RCS registration step.
3. Registration of the RCS inside the pipeline database. This step is required to run the RCS data products validation step
4. Data products validation tests. It mainly consists of running the end-to-end (E2E) verification tests.

The following sections detail each of these steps.

2.4.2 RCS installation

As soon as the ROC team has been informed of a new RCS release, it must download the RCS content into the ROC server. In practice it should only consist of downloading/upgrading (i.e., cloning/pulling) the local S/W Git repository. Before proceeding, the ROC system administrator in charge shall always be sure that the branch of the repository is “master” and that the version is correctly tagged.

The installation of a RCS must not require any compilation or configuration process. It means that the teams in charge must ensure that the delivered S/W are ready to be run on the ROC servers. Especially, it means that the delivery package must contain all of the files required to run the program on the ROC servers, as highlighted in the section 2.3.3.

Note that if RCS require compilations to work, the teams in charge must also include in the delivery package, all source files, scripts and libraries required for the installation and compilation. In case of anomalies, these files may be used by the ROC for investigations.

2.4.1 RCS interface validation test

The RCS interface validation test allows the ROC to check the RCS compliance with the interface specified in [RD6]. This test is run by a dedicated tool [RD13], which analyses the S/W interface against the expected specification.

In practice, this tool is a tailored version of the ROC pipeline. The way to set up and use this tool is given in [RD13].

Note that the RCS teams can also use this tool from the ROC development server, in order to validate their S/W interface prior to the delivery.

2.4.2 RCS registration

Once the installation is ended, the registration process must be launched.

This step consists to read information in the RCS descriptor file, check its integrity and its consistency, and then insert the content into the pipeline database.

The registration is performed by the pipeline it-self, with a dedicated command. In the case where the descriptor file is not valid, or if the content inside is inconsistent with the database, the pipeline must abort the registration[1].

In all cases, the pipeline always keeps track in its database, of the previous versions of the registered RCS.

The registration step is required before running RCS into the pipeline.

2.4.3 RCS data products validation step

After the registration, the pipeline will perform the validation of the RCS data products. It will consist of

- Verify the master CDF structure
- Running end-to-end (E2E) tests with the dedicated "RCS E2E data package" delivered by the teams

These two tests are described in the section 2.5.

2.5 RCS data products validation tests description

2.5.1 RCS master CDF verification test

During this test, the structure of the RCS master CDF will be checked to ensure that its content is as expected.

In practice, it will be performed using the "cdf.validator" module of the maser4py Python package [RD16]. This package requires a template file as a model to check a CDF binary file content.

2.5.2 RCS end-to-end (E2E) verification test

2.5.2.1 Concept

The RCS E2E test consists of checking the S/W data production, by comparing a set of known output files with the same files, but generated lately in the same or in a different environment. Especially, this should allow the ROC to verify that the RCS output files produced by the pipelines at the ROC site, are as expected.

To achieve this goal, each team has to deliver to the ROC a so-called "RCS E2E test data package", which must contain a set of input data files and their corresponding expected output files for each of the S/W functions to test. This set may not have to be exhaustive, but will have to be representative enough to be fully confident about the S/W data production behaviour.

2.5.2.2 RCS E2E test data package content description

The RCS E2E data package must be organized as shown in the figure below.

[1] For instance, if the software version in the descriptor has changed compared to the database content, but the release date is the same, then an error is returned. Otherwise, information about the new version has been inserted.

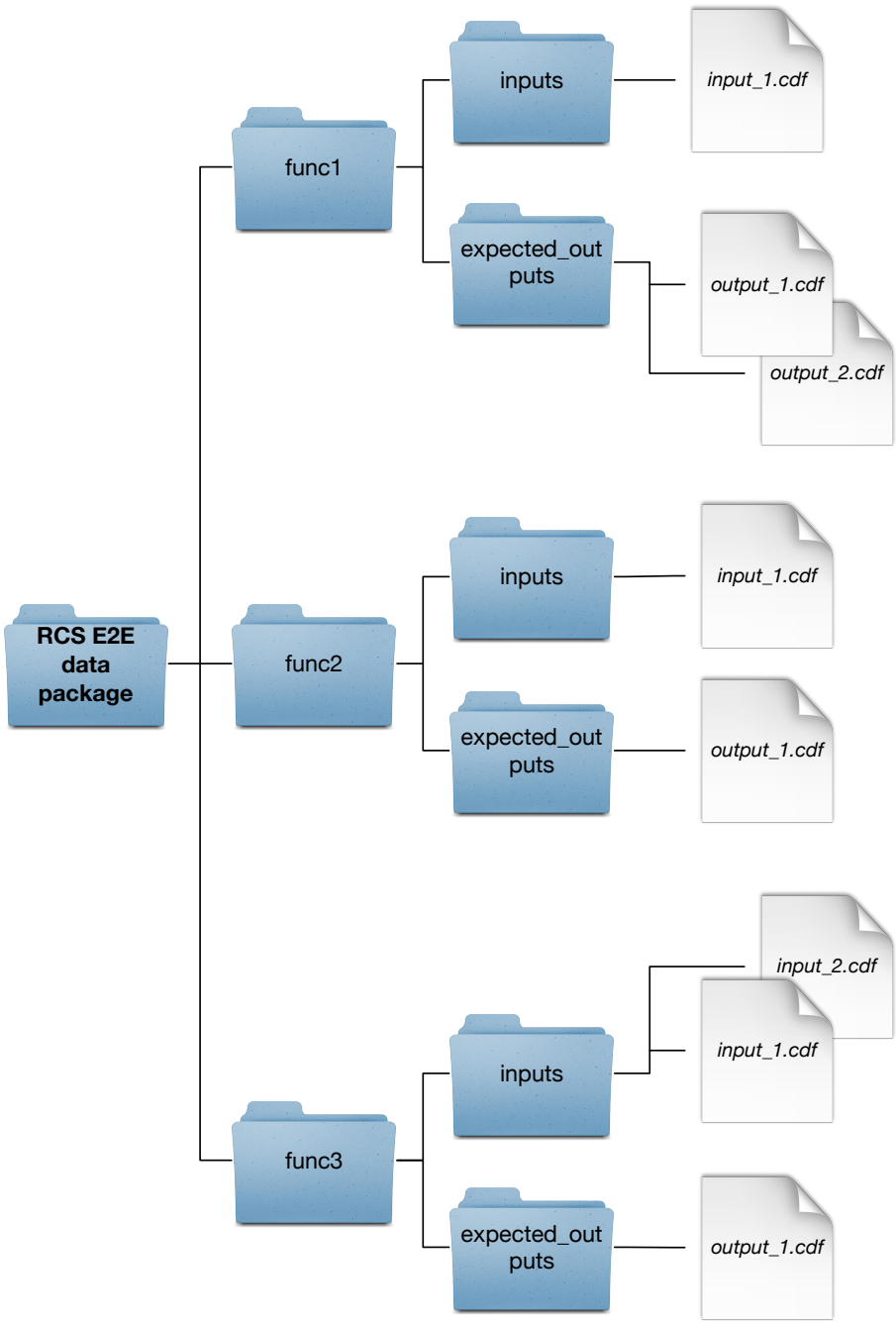

Figure 1. RCS E2E test datapackage content.

There must be one directory per S/W function (i.e., mode) to test. The name of the directory must be the value of the `"modes.name"` attribute in the descriptor file (see the descriptor structure in [RD6] for more details). Each directory must contain the following items:

- An "inputs" sub-directory, containing the input RPW data files to be used to run the E2E tests for the given function.
- An "expected_outputs" sub-directory, containing the expected output RPW data files. These files will be used by the ROC, in order to compare with the files generated by the RCS inside the ROC pipelines.

The name of the E2E test data files - inputs and expected outputs – must follow the convention:

RCS_E2E_[PIPELINE_ID]_[RPW-DATASET-ID].cdf

Where [PIPELINE_ID] is the pipeline ID (“RODP” or “RGTS”) and [RPW-DATASET-ID] is the corresponding RPW dataset ID.

Note:

- A same file can be used to test several functions. In this case, there must be as many as copies of this file than the number of the functions to test (i.e., one copy per function directory).
- When comparing files, the ROC does not compare the meta-data that are not stateless relative to the time and software instance (e.g., “Generated_by”, “Generation_date” and “FILE_UUID”).

2.5.2.3 RCS E2E test data package delivery mechanism

The RCS E2E data package must be saved by teams, inside dedicated folders of the ROC “DataPool” Git repository. The path of this folder relative to the repository main directory must be:

Tests/E2E/RCS/[NAME_OF_RCS]

Where [NAME_OF_RCS] is the name of the RCS.

If a new package is available, the team in charge will just have to upload the new package into the new “DataPool” Git repository on the Gitlab server. In practice, it should only consist of upgrading the folder above, but in the local copy of the “DataPool” repository. Then commit and push the new version toward the remote repository on the Gitlab server.

As explained in the section 2.3.1.2, the teams have to use the “rcs” branch to commit and push their package.

The ROC team must be informed each time a new package has been released. It will ensure that the ROC always uses the right package in order to perform its RCS E2E verification tests.

2.6 Procedures relative to the RCS already deployed inside the ROC pipelines

2.6.1 Maintenance of the RCS

The ROC team plans to monitor the behaviour of the RCS called by its pipelines during the mission. It will alert the teams in case of anomalies, however it is outside of its scope to modify the S/W source code or to perform upgrades, e.g., in case of bugs or new calibration tables.

It is hence the role of the teams in charge to ensure the maintenance of their S/W, especially they must be sure that the compliance with the ROC interface [RD6], and the RCS data production is as expected.

2.6.2 Upgrading the RCS

If an upgrade of the RCS is needed, then it must be considered as a new delivery.

Since the outputs of given S/W can be the inputs of another one, the team in charge must inform the other teams, including the ROC, any time the list of outputs for its S/W has been changed, by sending an email to the roc.rcs@sympa.obspm.fr mailing list.

If this modification leads to change the source code, the S/W must be delivered with a new S/W version "X.Y.Z.

In the case where the change only concerns files, such as master CDF or calibration tables, a new S/W version is not required, but the "master" branch (containing the new files) must be tagged with the extra "k" integer convention, as explained in the section 2.3.1.1.

2.6.3 RCS bug tracking

If a team or the ROC finds a bug in a given RCS that impact the pipeline and/or the other RCS, it must create a new JIRA issue in the ROC JIRA server. This issue must be saved in the dedicated ROC-DATAPROD JIRA project.

Additionally, the ROC and the RCS team must ensure that all people concerned by the bug are informed.

When the bug is fixed, the issue must be closed and people informed. The team in charge must then follow the procedures described in the section 2.6.2 to upgrade its S/W.

2.6.4 Maintenance of the RODP

If a maintenance impacting the RCS is performed on the RODP, the teams are informed by an email sending to the roc.rcs@sympa.obspm.fr mailing list.

2.7 Procedures relative to the RCS-related data

2.7.1 Delivering master CDF binary files

The master CDF binary files, required to produce the output CDF files, must be delivered with the RCS. The master CDF files can be generated by the teams them-selves (see 4.1 for more details about how to generate the master files), or copied from the `ROADS/RODP/CDF/Master` folder of the "master" branch of the ROC "DataPool" Git repository.

2.7.2 Delivering CDF skeleton table files

The CDF skeleton table files, used to generate the master CDF files, must be also archived in the dedicated ROC "DataPool" Git repository. The delivery file format to be used must be the Excel 2007 (i.e., .xlsx), as required by the "maser4py" Python library [RD16] in order to generate CDF skeleton tables and master binary CDF files.

To deliver their CDF skeleton files, the teams in charge must upload (i.e., push) them into the `ROADS/RODP/CDF/Excel` folder of the ROC "DataPool" Git repository. Since the "master" branch has a read-only access, the teams must use the dedicated "rcs" branch.

The read/write access to the "rcs" branch can be requested using the roc.support@sympa.obspm.fr mailing list. The access to the Git repository is also possible from SSH key mechanism.

Once files have been uploaded, the teams must inform the ROC team using the roc.lesia@sympa.obspm.fr list. The ROC team will then generate the new skeleton tables (.skt) and CDF binary files (.cdf), and merge the "rcs" branch into the "master". At the ROC team will then inform all of the teams via the roc.rcs@sympa.obspm.fr mailing list.

3 GUIDELINES FOR THE USAGE OF THE ROC ENGINEERING INFRASTRUCTURE

Depending of the need, a part of the ROC infrastructure is accessible to external collaborators. The accessibility concerns more specifically the collaboration tools, but also servers and data disks.

This section gives the rules to be followed by external users relative to this infrastructure.

All requests/questions concerning these tools must be done using the roc.support@sympa.obspm.fr list.

3.1 ROC team collaboration tools

3.1.1 ROC Gitlab server

The ROC uses git as a main VCS to store its source codes. The ROC git repositories are managed using a Gitlab server:

https://gitlab.obspm.fr/ROC

The ROC can provide an access to its Gitlab server on demand.

If the demand concerns software to be implemented in the RODP, the Vincent Driessen's branching model [RD8] shall be applied, and only tagged master branch will be used by the ROC.

3.1.2 ROC SVN repository

The ROC can provide an access to its SVN repository on demand. Nevertheless, it must be noticed that the ROC team does not use the SVN repo. for its software development and execution.

3.1.3 JIRA Issue tracker tool

The ROC team uses JIRA as an issue tracker tool. This tool is accessible on demand to external users.

3.1.4 ROC Documentation Manager System

External users can ask for an access to the ROC Documentation Manager System (DMS), which contains the documentation of the ROC.

Especially, any document relative to the RCS must be archived on the ROC DMS.

3.1.5 ROC Wiki page

The ROC project Wiki page is available in:

https://confluence-lesia.obspm.fr/display/ROC/ROC

This Wiki page is run under an Atlassian Confluence server, in order to share information concerning the RPW ground segment project.

Note that the restricted area of the site is currently limited to 25 users.

3.1.6 RPW Web portal

There is no specific rule concerning the RPW Web portal, which is publicly accessible from Internet at https://rpw.lesia.obspm.fr.

3.1.7 ROC intranet site

The ROC intranet Web site is only accessible by the ROC team at LESIA.

3.1.8 ROC mailing lists

External users can use one of the two following mailing lists to communicate inside the ROC project:

- roc.teams@sympa.obspm.fr, must be used for sending messages for all people involved in the RPW ground segment activities.
- roc.lesia@sympa.obspm.fr, must be used for sending messages only to the ROC team at the LESIA.
- roc.cal@sympa.obspm.fr, must be used for sending messages related to the RPW calibrations activities
- roc.sgse@sympa.obspm.fr, must be used for sending messages related to the ROC-SGSE engineering activities
- roc.rcs@sympa.obspm.fr, used for discussions related to the RCS development and integration activities.
- roc.support@sympa.obspm.fr, must be used for sending messages if an assistance is needed when using ROC infrastructure, data products or software.

1.1 ROC servers

The description of the ROC servers is given in the "ROC Software Development Plan" document [RD7].

3.1.9 Access policy

External collaborators can ask to open a user account on the ROC development server *roc-dev.obspm.fr*.

The ROC development server is only accessible from the Observatoire de Paris intranet and using the SSH protocol only. This requires users to have both a valid LDAP account at the Observatoire de Paris and a user account on the development server.

Note that the users must connect to the *styx.obspm.fr* server first with their LDAP login and password in order to reach the intranet.

Any request concerning an access to the ROC development server must be addressed to the roc.support@sympa.obspm.fr mailing list.

1.1.1 Usage policy

The default user account access privileges are defined by the ROC team, but can be changed if required.

In the same, the space quota per user must be limited to **20 Gigabytes**, but can be also extended on demand. Especially, large data might not be stored directly in the ROC server. They shall be read/saved from/to the dedicated data disks visible on the server (see next section). This requirement avoids exceeding the disk quota on the server, which is dedicated to data processing.

3.2 ROC data disks

The description of the ROC data disks is given in the "ROC Software Development Plan" document [RD7].

3.2.1 Access policy

The ROC has 16 Terabytes data disk mounted on the ROC servers, and accessible through the "/volumes" local directory.

Each user on the ROC development server has a dedicated space on this disk, reachable at the path:

`/volumes/plasma/rpw/roc/data/https/private/users/[name_of_user]`

Where `[name_of_user]` is the name of the user account of the server.

Note that this space can also be visible from Internet via the URL:

https://rpw.lesia.obspm.fr/roc/data/private/users/[name_of_user]

Where [name_of_user] is the name of the user account of the server. The access is restricted and will require to login using the LDAP info.

1.1.1 Usage policy

Each user has the right to read/write inside its dedicated space only. The total amount of data stored in the space must not exceed **50 Gigabytes**. This default configuration can be changed on demand.

4 APPENDIX

4.1 RODP CDF file production mechanism

All of the RODP CDF data files, including the RCS, must be produced using master CDF binary files [RD12].

Figure below presents the concept of the CDF file production mechanism.

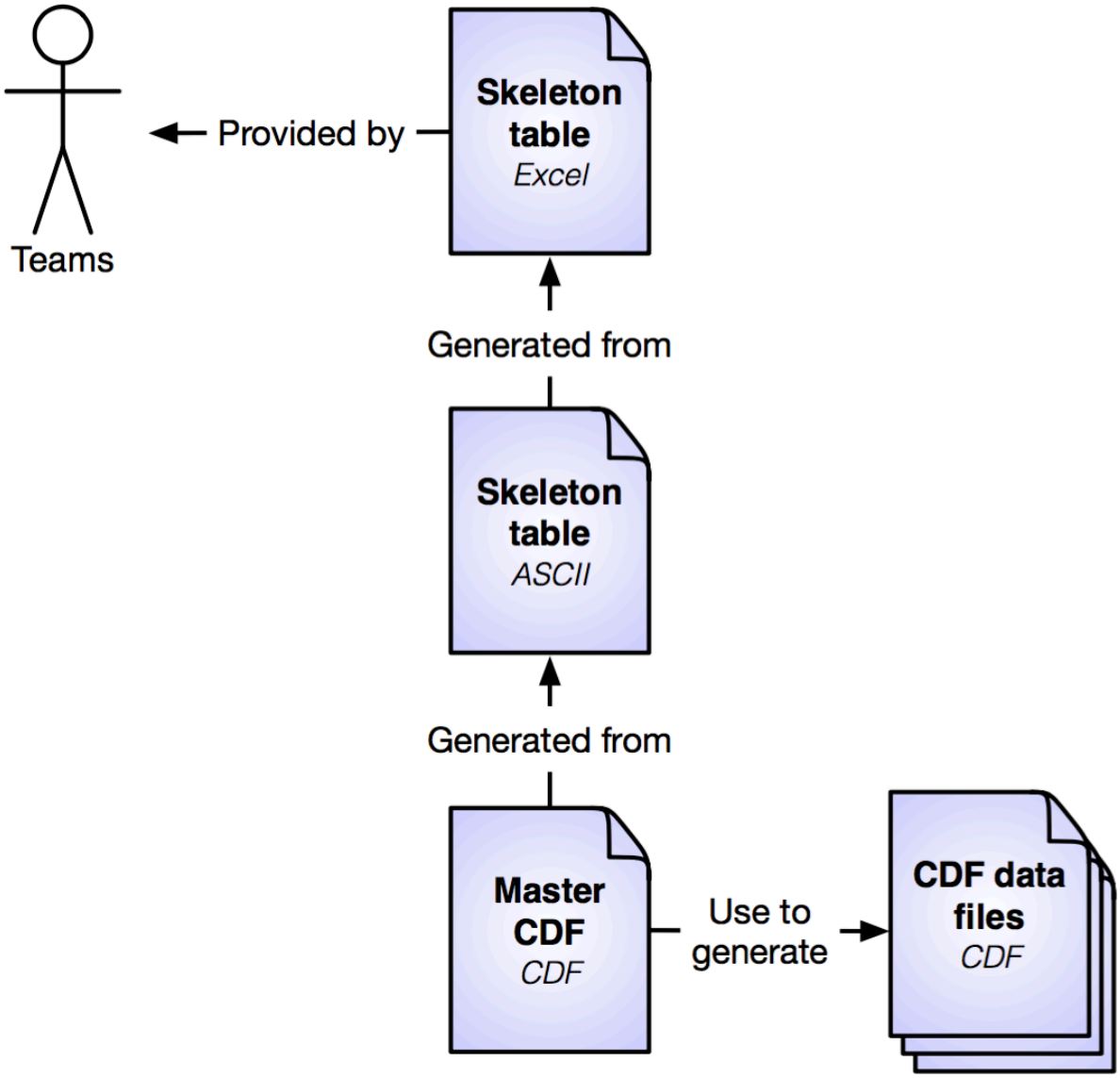

Figure 2. RODP CDF file production mechanism.

The main steps are:

1. Each team writes the CDF skeleton files for their RPW sub-system data sets. It must be one skeleton file per data set. The format uses to save these files is the Excel 2007 format (i.e., .xlsx).
2. From these skeleton Excel format files, the corresponding CDF skeleton tables in the ASCII format can be generated using the maser4py Python package [RD11].
3. The master CDF binary files are then created using the "skeletoncdf" tool of the NASA CDF software [RD12]
4. The CDF master files can be then used by the RODP to produce the CDF data files.

5 LIST OF TBC/TBD/TBWS

TBC/TBD/TBW			
Reference/Page/Location	Description	Type	Status

6 DISTRIBUTION LIST

LISTS See Contents lists in "Baghera Web": Project's informations / Project's actors / RPW_actors.xls and tab with the name of the list or NAMES below		
		Tech_LESIA
		Tech_MEB
		Tech_RPW
		[Lead-]Cols
		Science-Cols

INTERNAL

LESIA CNRS	x	M. MAKSIMOVIC
	x	Y. DE CONCHY
	x	X. BONNIN
	x	QN. NGUYEN
	x	**S. LION**
		L. GUEGUEN
		P. PLASSON
		A. BOUBACAR AMADOU

LESIA CNRS		

EXTERNAL (To modify if necessary)

CNES		C. FIACHETTI
		C. LAFFAYE
		R.LLORCA-CEJUDO
		E.LOURME
		M-O. MARCHE
		E.GUILHEM
		J.PANH
		B.PONTET
IRFU		L. BYLANDER
		C.CULLY
		A.ERIKSSON
		SE.JANSSON
	x	A.VAIVADS
LPC2E		P. FERGEAU
	x	G. JANNET
		T.DUDOK de WIT
	x	M. KRETZSCHMAR
		V. KRASNOSSELSKIKH
SSL		S.BALE

AsI/CSRC		J.BRINEK
		P.HELLINGER
		D.HERCIK
		P.TRAVNICEK
IAP		J.BASE
		J. CHUM
		I. KOLMASOVA
		O.SANTOLIK
	x	J. SOUCEK
	x	L.UHLIR
IWF		G.LAKY
		T.OSWALD
		H. OTTACHER
		H. RUCKER
		M.SAMPL
		M. STELLER
LPP	x	T.CHUST
		A. JEANDET
		P.LEROY
		M.MORLOT
	x	B.KATRA