RPW Operation Centre

ROC Engineering Guidelines

ROC-GEN-SYS-NTT-00008-LES
Iss.01, Rev.03

Prepared by:	**Function:**	**Signature:**	**Date**
Xavier Bonnin	RPW Ground Segment Deputy Project Manager		09/11/2016
Verified by:	**Function:**	**Signature:**	**Date**
Name	Team Member #2		Dd/mm/yyyy
Approved by:	**Function:**	**Signature:**	**Date**
Name	Team Member #3		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 release
01	01	18/11/2015	X.Bonnin	Update the acronym list. Modify the recommended S/W structure. Update the S/W versioning and descriptor requirements.
01	02	07/04/2016	X.Bonnin	Rename main sections title Remove requirement sections Replace SVN guidelines by Git
01	03		X.Bonnin	

Acronym List

Acronym	Definition
AIT	Assembly Integration and Test
AIV	Assembly Integration and Validation
ASCII	American Standard Code for Information Interchange
BIA	BIAS
CDF	Common Data Format
CoI	Co-Investigator
DS	Data Sets
ESA	European Space Agency
ESAC	European Space Astronomy Centre
GIGL	Groupe Informatique Générale du LESIA
IAGC	International Association of Geophysical Contractors
ICD	Interface Control Document
ID	Identifier

IOR	Instrument Operation Request
ISTP	International Solar Terrestrial Physics
HK	HouseKeeping
LDAP	Lightweight Directory Access Protocol
LESIA	Laboratoire d'Etudes Spatiales et d'Instrumentations en Astrophysiques
MOC	Mission Operation Centre
NFS	Network File System
OS	Operating System
PI	Principal Investigator
PMP	Project Management Plan
ROC	RPW Operation Centre
RPW	Radio and Plasma Waves instrument
SGS	Science Ground Segment
SOC	Science Operation Centre
SVN	SubVersioN
TBC	To Be Confirmed
TBD	To Be Determined
TBW	To Be Written
URL	Uniform Resource Locator

Table of Contents

List of figures

1 GENERAL

1.1 Scope of the Document

This document addresses guidelines related to the software engineering activities of the RPW Operation Centre (ROC) [RD1]. Its main goal is to help the ROC developer team to:

- Ensure the efficiency and homogeneity of the software development, validation and application.
- Define common rules to optimize collaboration between
- Optimize the software delivery and deployment
- Ensure the required documentation is delivered with software

The convention concerning external software to be run at ROC - such as RPW Calibration Software (RCS) – are listed in the ROC Engineering Guidelines for External Users document [RD3].

The present document should be read in complement to the ROC Software Product Assurance Plan (SPAP) [RD8].

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/0/2	ROC Concept and Implementation Requirements Document	Y.de Conchy	24/06/2015
RD2	ROC-GEN-SYS-PLN-00015-LES/00/00	ROC Software Development Plan	X.Bonnin	25/06/2015
RD3	ROC-OPS-PIP-NTT-00018-LES/0/0	ROC Engineering Guidelines for External Users	X.Bonnin	29/01/2015
RD4	ROC-GEN-DAT-SPC-00006-LES/0/0	RPW Data Products	X.Bonnin	01/10/2014
RD5	ROC-TST-GSE-SPC-00017-LES/0/01	Data format and metadata definitions for the ROC-SGSE data	X.Bonnin	09/07/2015
RD6	ROC-TST-GSE-ICD-XXXX-LES/0/1	RPW Calibration Software ICD	M.Duarte	20/07/215
RD7	ECSS-Q-ST-80C	Space product assurance –	ECSS team	06/03/200

		Software product assurance		9
RD8	ROC-GEN-MGT-QAD-000XX-LES/0/2	ROC Software Product Assurance Plan	S.Papais	19/10/2016
RD9	https://git-scm.com/	Git – fast-version-control	Git Team	09/11/2016
RD10	http://nvie.com/posts/a-successful-git-branching-model/	A successful Git branching model	V. Driessen	05/01/2010
RD11	ROC-GEN-MGT-PLN-00013-LES/1/0	ROC Project Management Plan	X.Bonnin	04/01/2016

1.4 About this document

1.4.1 Access policy

This document must be accessible without any restriction to the teams involved in the RPW ground segment activities. Any other access requests to the document must be addressed to the ROC team for acceptance.

Any modification of this document must be approved by the RPW Ground Segment Project Manager before publication.

1.4.2 Terminology

In the framework of this document:

- The data processing and operation definitions refer to all of the procedures, software (S/W), data and documentations required to produce the complete ROC data sets, and to perform the instrument operations during the in-flight mission.
- The ROC project encompasses all of the activities to be supervised by the ROC.

Except if it is explicitly mentioned in the table below, the terms definitions provided in the document [RD8] are applicable.

Name	Definition
(Team) collaboration tools	Software or interfaces that can be used to collaborate on a document, file, or software development. (e.g., SVN, JIRA, etc.)
Data products	Generic term for data produced by software or stored in a database. They are typically files, entries in a database, images on a screen, or a given ROC data set.
(ROC) data set	A set of data products fully identified in the ROC software system (e.g., LFR level 1 data set). It shall be associated with a given support (e.g., CDF file format) and follow the convention defined in [RD4].
External team/person/user	Team/person/user that does not belong to the ROC team in LESIA.
File	Any type of file including database dump files, source codes, documents, etc.
Ground tests	All of the tests performed on the RPW system or sub-systems before the pre-launch phase. (e.g., performances/calibrations, SBM algorithm validations, etc.)
In-flight data	Data produced in space after the launch of the Solar Orbiter probe.
Integration	Integration of a given S/W into a given ROC pipeline.
(S/W) job	A time stamped S/W process run by a given S/W. A job should be unique and permit to retrieve which S/W creates it and when.
(Input) keyword	Keyword provided as an input argument in a S/W. It could be a boolean flag (e.g., '--help') or not (e.g., '-f filepath').
Mission	Refers to the in-flight phase of the project (i.e., after the launch of the Solar Orbiter probe).
Mission data	See "In-flight data" definition.
(Software) module	See Software unit definition.

On-ground data	Data produced on-ground before the launch of the Solar Orbiter probe.
Operations	Tasks relative to the instrument control and monitoring.
Pipeline	Set of S/W, files or databases that allows the production of the different levels of required data.
(S/W) process	A well identified type of operation that can be realized by a given S/W.
Production pipeline	Stable release of a pipeline used for the production of fully validated and ready to be distributed RPW data.
Release	A set of (version number, release date, responsible) for a given S/W or data products.
ROC Software System	The ROC software system (RSS) is the top-level system that regroups all of the software systems developed and run in the framework of the RPW ground segment activities. The RSS includes the ROC GSE and ROC Operations And Data System (ROADS).
ROC sub-system	A system that belongs to the ROC Software System
(S/W) task	See (S/W) process definition.
ROC team	People working in the ROC based at the LESIA (Paris Observatory, Meudon).
Root directory	The lower level directory in a given arborescence tree.
RPW ground segment team	People working on the RPW ground segment activities, including the ROC team, the RPW PI and the RPW sub-system Lead-CoIs.
Stable release	Release of S/W that has been tested and validated. A stable release is ready to be deployed in the ROC servers.
S/W directory	The main directory of a given S/W. It corresponds to the root directory of the S/W.
(Software) unit	Unit or equipment of a given ROC sub-system. It is typically software or database.

Table 1. Terminology.

2 GUIDELINES FOR THE ROC SOFTWARE SYSTEM (RSS)

The guidelines in this section concern hardware and software relative to the ROC Software System (RSS).

2.1 RSS environment convention

2.1.1 Usage and accessibility of the ROC servers

All of the *roc* servers must only by accessible with read-write privileges via the SSH protocol, from the Observatoire de Paris intranet and to authorized users.

All of the *roc* data disks must only by accessible using a NFS-like mounting system from the *roc* servers. Some parts of the disks can also be visible from Internet but always in read-only mode.

The *operation*, *low* and *web* servers must be used for operations, data productions and visualizations only. It must not be used for developments and must be accessible by the ROC team only.

The ROC team must ensure that the *development* and *operation* servers have the same hardware and software environment.

The ROC team must provide a dedicated space on its *development* server to the people involved in the RPW ground segment activities. The access must be only possible from the Observatoire de Paris intranet, and through a SSH using a LDAP user account delivered by the GIGL on the ROC team demand. By default, users must be not allowed to read/write outside their space. The size allocated by the ROC team for each user space must be limited, but can be extended on demand.

The ROC servers must not be used to store data produced by S/W. Data products, including log files, shall be saved on the dedicated data disks, which have to be accessible from the servers through NFS-like mounting systems.

2.1.2 Python software specific convention

A virtual environment must be used to run each instance of a Python S/W on the ROC servers. This rule must be applied for both producing and testing instances. It is highly recommended to apply also this rule for software development.

2.2 RSS software general convention

2.2.1 Software identification

All of the ROC S/W must be assigned a unique identifier (ID). This ROC S/W ID shall be a string in capital letters (e.g., ‘RPL’). Only alphanumerical characters, hyphens and underscore characters are authorized.

The ROC must be the only entity authorized to assign the S/W IDs.

2.2.2 Software versioning

S/W 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").

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 suffix ("0.Y.Zb" for the 0.Y.Z beta version for instance).

2.2.3 Software testing/debugging

S/W must include a debug mode that allows ROC developer team to test the S/W possible failures.

S/W must be able to produce a log file that reports the S/W activity.

S/W should also include unit testing to help ROC to

2.3 Software organization convention

2.3.1 Software structure tree

There is no specific convention concerning the ROC S/W structure. Nevertheless, it should contain at least the following items:

- The S/W source code files
- Any additional libraries required to compile and/or to run S/W
- Any script or program used to compile S/W (makefile or ant build file for instance).
- Any script or program used to run S/W, including executable binary files that shall be runnable in the ROC servers.
- The corresponding 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 convention.

For more clarity, it is recommended to apply the following conventions for the S/W directory organization, providing:

- A **/config** directory that can be used to store configuration files loaded by S/W.
- A **/doc** directory containing the S/W documentation.
- A **/data** directory containing additional files to be read as input arguments by S/W (e.g., master CDF binary files). This directory must not contain S/W data inputs/outputs (e.g., TDS/LFR/THR Level 1 or Level 2 data files).
- A **/lib** directory containing additional libraries required to run S/W.
- A **/scripts** directory that can be used to store scripts or batch files used to setup, run or manage S/W.
- A **/src** directory containing the S/W source code files. Note that in the case of a Python package software, the name of the Python package can be used as a /src directory.
- An optional **/tools** directory that can be used to provide any useful tool to test, debug, validate and/or manage S/W.
- If S/W executable binary files are provided, they should be saved in a **/bin** directory.

- If S/W is a Python package, the root directory must include a *requirements.txt* file, as defined for *pip* tool usage.

The following ***context files*** must be saved into the S/W root directory:

- *README.rst* A reStructuredText format file providing general information about S/W
- *CHANGELOG.rst* A reStructuredText format file providing history of modifications of the S/W releases

In addition the following optional files could be added:

- *howto.txt* An ASCII format file providing instructions to run S/W
- *install.txt* An ASCII format file providing explanations on how to install S/W

All of theses files shall be always up-to-date.

These context files are described in more details in the next sections.

Additions of any other directory or file are left to the S/W responsible initiative.

Figure 1 presents the minimal tree structure of the root directory, which is should be found for a S/W running on a ROC server.

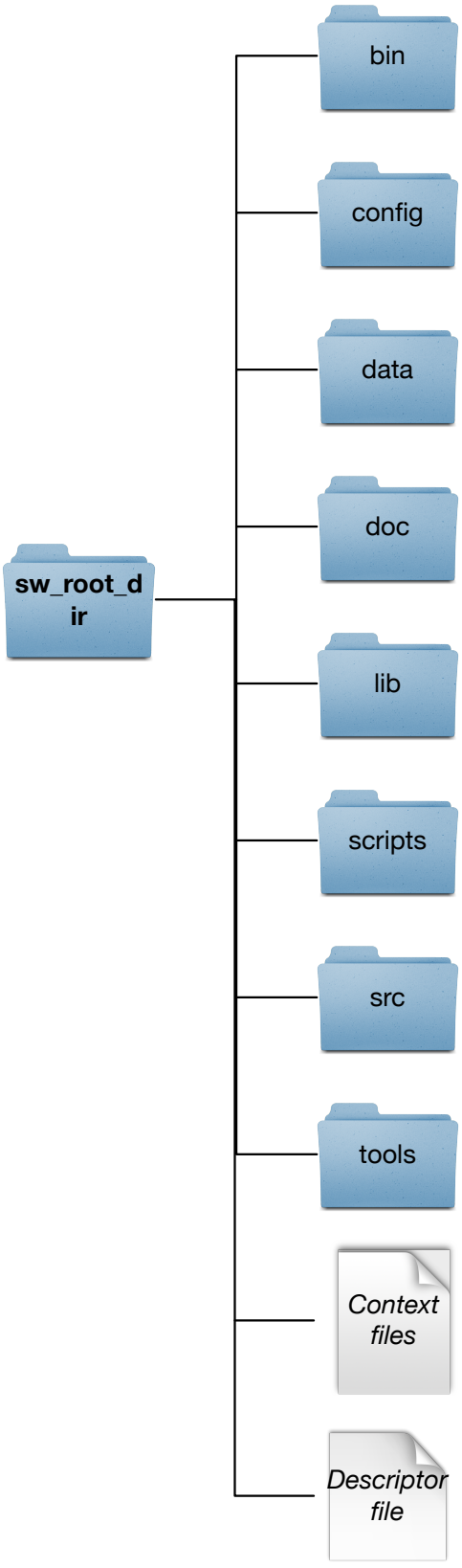

Figure 1. Recommended S/W directory structure.

2.3.2 Context files

The **README.rst** file must provides at least information about:

- The purpose of S/W
- A description of the root directory content
- Limitations or caveats of S/W use
- A reference to a more complete documentation
- A contact of the person or team in charge

The **CHANGELOG.rst** file must provide a history of the S/W releases, including:

- The version of the releases
- The date of the releases
- The responsible of the releases
- A summary list of what has been changed since the previous release.

The **howto.txt** file should explain how to run S/W. The use of additional tools use can be also described in this file.

The **install.txt** file should describe how to install S/W, including:

- The name and version of the languages and compilers to use.
- The name and version of additional libraries, and if possible where they can be retrieved.
- Instructions on how to compile the S/W source code.
- If available, instructions on how to test that the installation has been done correctly.

2.3.1 ROC S/W descriptor file

Any ROC S/W or POPPy plugin to be integrated into a ROC pipeline must be delivered with a corresponding descriptor file named "roc_sw_descriptor.[id_of_pipeline].json", where [id_of_pipeline] is the identifier of the ROC pipeline. The format of this file must be JSON. This file must be unique and placed in the S/W root directory for external S/W and in a /config sub-directory for POPPY plugin. It must be always up-to-date.

2.3.2 The S/W root directory naming

The name of the S/W root directory must contain alphanumeric, hyphen "-" and underscore "_" characters only.

2.4 ROC software life-cycle convention

This section describes all of the procedures to be applied concerning the delivery, storage, deployment, development, integration, execution and maintenance of S/W to be run on the ROC servers.

2.4.1 ROC software release

The delivery procedure must comply with the following steps:

- Before delivery, the ROC team must ensure that the S/W has a unique ROC S/W ID. This ID must be reported into the S/W descriptor file. This step is only required for the first delivery.
- The delivery must be done via the ROC Gitlab server

Any software release must be delivered with a given version number.

2.4.2 Software storage on the ROC servers

The root directory of the S/W run on the ROC servers must be archived in the dedicated ROC Git repository following the conventions described in this document.

Note that a local copy of the Git repository is expected to be found also in a dedicated directory on the ROC servers.

The path of this local repository must as much as possible remain unchanged, even if a new version is deployed.

2.4.3 ROC Software tests and validation

The tests of S/W must always be performed first on the roc development server, before integration.

The major S/W development tests and validations might be reported into a dedicated JIRA issue.

2.4.4 Software integration tests and validation

The S/W integration tests must be performed on the roc development server.

The S/W major integration tests and validations shall be reported into a dedicated issue in the appropriate JIRA project page.

Before being integrated into a given ROC production pipeline any new S/W release shall fully complete the following steps:

Any S/W shall be individually tested and validated by the person or team in charge before the integration processes.

Once S/W has been successfully tested and validated, a new S/W release shall be committed into the corresponding Git repository and with the appropriate (tagged) version.

Then people in charge shall test and validate that the S/W works correctly into the corresponding *integration test pipeline* on the *production* server.

If the S/W integration is validated, S/W can be finally integrated into the corresponding *production pipeline* by the people in charge.

Figure 2 summarizes the integration steps.

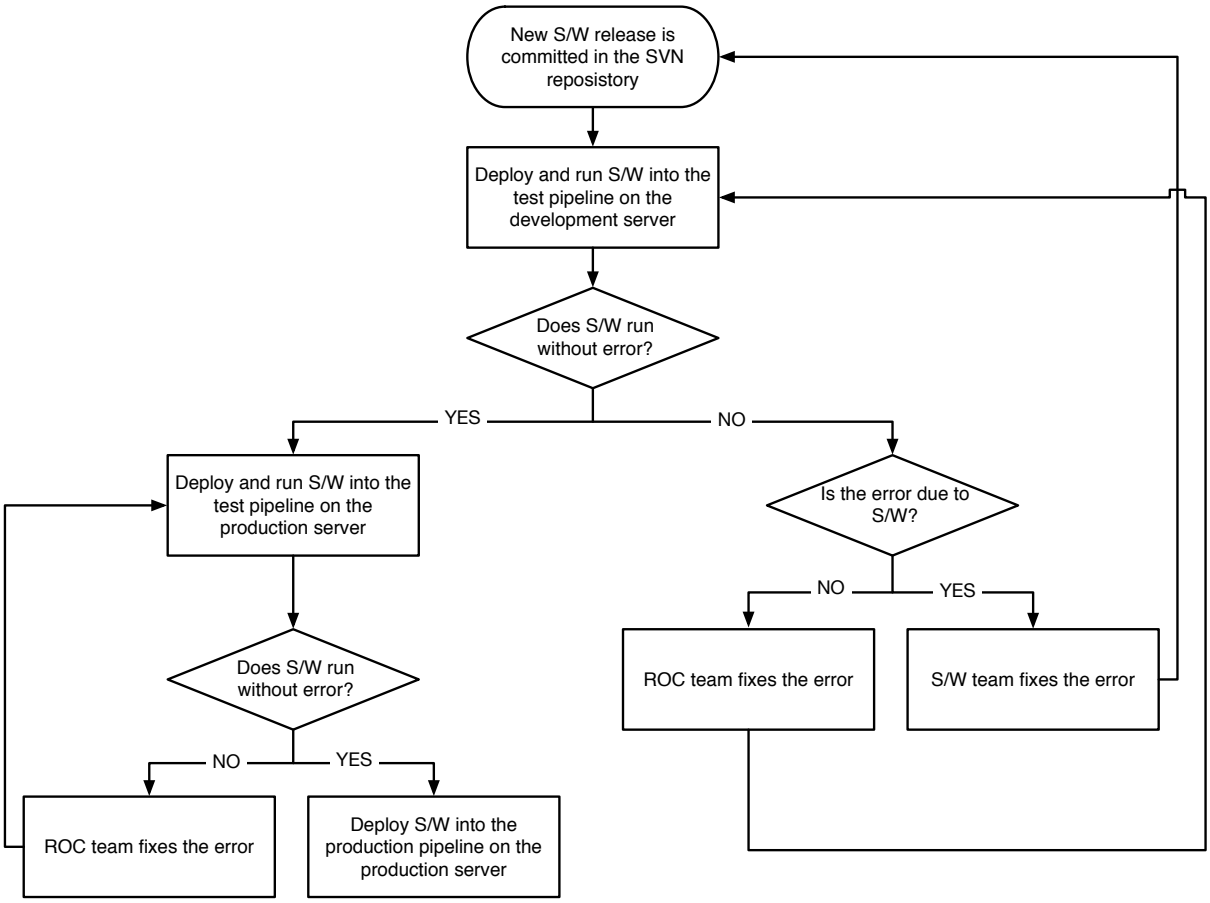

Figure 2. S/W integration procedure.

2.4.5 Software deployment on the ROC servers

Only people from the ROC team are authorized to deploy S/W on a *production* server. External teams can use the allocated space in the development server to test their S/W compilation and execution in the ROC programming environment.

Any S/W deployment on the development server shall be performed in the following order:

1. Once a new version '*X.Y.Z*' of S/W is available in the dedicated Git repository, then people in charge of deploying S/W on the ROC *production* server shall be informed.
2. This new version is then downloaded on the *development* server updating the local SVN repository only.
3. The integration of the new S/W release into the development or integration test pipeline shall then follow the procedure described by the requirement REQ-ROC-REG-0035.

Any S/W deployment on the production server shall be performed in the following order:

1. Once a new version '*X.Y.Z*' of S/W has been tested and validated on the integration server, an update of the local SVN repository can be done on the production server to retrieve this version.
2. The integration of the new S/W release into the production pipeline shall then follow the procedure described by the requirement REQ-ROC-REG-0035.

2.4.6 Software execution

S/W execution in the production pipelines shall be as much as possible automated with a given cadence.

2.4.7 Software failures / warnings

Any S/W error or warning shall be automatically archived and reported to the users using the dedicated modules of the pipeline.

2.4.8 Software maintenance / upgrades

Opening a dedicated issue in the appropriate JIRA project page shall start any S/W bug or issue resolution.

Any S/W upgrade shall be considered as a new release with its own tuple of (version number, release date, person in charge) and its own folder in the /tags directory.

The ROC team shall be informed each time a new release related to a S/W upgrade is available in the /tags directory.

As any new release, the integration of a S/W upgrade shall be tested and validated by the ROC team.

All support content including documentations, context and descriptor files shall be up-to-date with the upgraded S/W release.

2.4.9 Software calling

This section lists all of the requirements concerning the S/W calling.

S/W input keywords shall be callable using the two following formats only:

- A short format composed of one letter preceded by the hyphen prefix (e.g, “-v”, “-h”).
- A long format composed of a word preceded by the double hyphen prefix (e.g., “--version", “--help").

S/W input arguments shall at least contain the following keywords:

- ‘-h, --help’, which returns the help of the program.
- ‘-v, --version’, which returns information about the current release of the program (i.e., version number, release date, person in charge)

2.5 Convention related to the RPW calibration software

The convention concerning the RPW calibration S/W is defined in the “ROC Engineering Guidelines for External Users” document [RD3].

3 GUIDELINES FOR THE ROC DATA PRODUCTS

3.1 General convention

3.1.1 Data identification

Any data set (DS) produced by the ROC must be assigned a unique identifier (ID). This DS ID must be a string in capital letters.

The ID name must be composed of a tuple of three items of string type separated by underscore characters. The first item must correspond to the corresponding "*Source_name"* attribute value. The second item must indicate the "*level*" attribute value of the dataset. The third item must provide the "*Descriptor*" attribute value of the dataset. It can contain sub-strings separated by hyphens. It must be explicit enough to fully identify the data set (e.g., "RPW-TDS-BURST-RSWF" for Regular Snapshot Waveform level 1 data in burst mode for instance).

The ROC must be the only entity authorized to assign the DS IDs.

The ROC must ensure the integrity and validity of the DS ID listing.

3.1.2 Data versioning convention

The data file versioning must be compliant with the conventions defined in the "RPW Data Products" and "Data format and metadata definitions for the ROC-SGSE data" documents.

3.1.3 Data access policy

The ROC team must ensure an access to its RPW data file archive, including TM/TC packet data, HK, ancillary, and quick-look files, to the RPW and to the Solar Orbiter consortium. Two layers of accessibility mudt be planned:

- A restricted access for data products used only inside the RPW consortium for S/W testing, on-ground tests and data processing validation.
- A public access for final data products to be delivered to the ESAC archive centre. Access to the restricted area can be also envisaged for specific users such as other Solar Orbiter instrument teams. Demands will have to be addressed to the ROC team for acceptation.

3.2 Data file formats & organization convention

The RPW on-ground data format must be compliant with the conventions defined in the "ROC-TST-GSE-SPC-00017-LES" document [RD5].

The format of the RPW data produced during the Solar Orbiter mission must be compliant with the conventions defined in the "RPW Data Products" document.

4 GUIDELINES FOR THE ROC PIPELINES

4.1 Pipeline general convention

4.1.1 ROC pipeline identification

Each ROC pipeline must be assigned a unique identifier (ID). This ID must be a 4-characters string, containing alphanumeric in upper case only. Hyphens and underscores can be used as a separator if required.

Here is the list of IDs for the main pipelines:

- "RGTS" for the ROC-SGSE pipeline
- "RODP" for the ROC Operations and Data Pipeline (RODP)

4.1.2 ROC pipeline environment

Each pipeline must be executed on a dedicated environment. This environment must be completely isolated from the others to avoid possible ambiguity. Especially, each pipeline must:

- Have its own instance name, which must be unique on a given server
- Have its own database
- Generate its own data products, which must be saved in separated directories with the associated access privileges.
- Be installed on a specific Python virtual environment. There must be one instance by virtual environment.

4.1.3 ROC pipeline instance naming

There can be several instances of a ROC pipeline, required for different purposes (e.g, dev/test instances, prod. instance, etc.) or installed on different servers (e.g., roc-dev, roc, laptop, etc.).

The name of the instance must:

- Be unique
- Contain only alphanumerical characters and underscore(s)
- Do not exceed 64 characters
- Must be the same than the name of the virtual environment

The name should be explicit enough to identify the pipeline and its function.

4.1.4 ROC pipeline versioning

The pipeline version must be a unique number sequence identifier “X.Y.Z”, where “X” is an integer indicating the release (major changes), “Y” is an integer indicating an issue (minor changes) and “Z” is an integer indicating a revision (e.g., bug correction).

4.2 Pipeline software unit execution

ROC pipelines must manage the S/W executions using jobs. Each S/W execution shall generate a new job identified by a Universal Unique Identifier (UUID). The list of executed jobs must be stored in a dedicated table.

4.3 Pipeline validation procedures

Depending on the stage of development, each pipeline can be found in one of the three following phases:

- A *development* state, which means that the pipeline is unstable and must be used for development purposes only. Data produced in this phase are unreliable and must not be distributed at all. There can be several instances of the dev. pipelines, running on the roc-dev.obspm.fr server.
- An *integration (test)* state, which means that the pipeline can be unstable and must be used to test and validate new integrations only. Data produced in this phase are potentially unreliable and must be only distributed to the RPW team for testing.

There must be only one instance of the *integration* pipeline, running on both the roc-dev.obspm.fr and roc.obspm.fr servers.

- A *production* state, which means that the pipeline is fully operational, stable and ready to be used to produce deliverable data. There must be only one instance of the production pipeline running on the roc.obspm.fr server.

Notes:

- All of the pipelines must pass by the *development* then *integration* states before being fully operational, including for updates.
- A *backup* pipeline instance might be deployed on both the roc-dev.obspm.fr and roc.obspm.fr servers, in the case where the *production* pipeline is crashed.
- Each pipeline comes with its own front-end and tools that allow to visualize and validate data products. Additional tools should be also developed to help ROC to maintain their production pipelines always operational.

5 GUIDELINES FOR THE ROC DATABASES

5.1 Database convention

5.1.1 ROC database identification

Any database must be assigned a unique ID. This ID must be a uppercase unique 3-characters name preceded by the "ROC-" prefix (e.g., "ROC-MDB", 'ROC-TDB").

This ID must be used to identify the database in the system.

5.1.2 ROC database versioning

The database version must be a unique number sequence identifier "X.Y.Z", where "X" is an integer indicating the release (major changes), "Y" is an integer indicating an issue (minor changes) and "Z" is an integer indicating a revision (e.g., bug correction).

5.1.1 ROC database naming

In the case of the ROC pipelines databases, the name of databases must be the name of the instance of the pipeline. Otherwise, the name must be explicit enough to identify the purpose and the function of the database (e.g., "_dev" for developement database).

Name of the databases, schemas, tables and columns must contain alphanumerical characters in lower-case only. Only the underscore character "_" can be used as a separator if required.

Name of the first column of a given table "tablename" containing the primary key must be the name of the table preceded by the "id_" prefix (e.g. "id_tablename").

Name of a column containing a foreign key must be the name of the column it refers to – for instance "colname", followed by the suffix "_id" (e.g., "colname_id"). If the referenced column is primary key (e.g., "id_colname") then the prefix "id_" must be removed (e.g., "colname_id" and not "colname_id").

6 GUIDELINES FOR THE RPW OPERATION SUPPORT SOFTWARE

TBW

7 GUIDELINES FOR THE ROC PROJECT MANAGEMENT AND DEVELOPMENT TOOLS

7.1 ROC project management tools

The list of the key personnel and Institutes involved in the ROC project are given in [RD1].

Only the following people are authorized to modify the project planning using the dedicated management tool:

- RPW Principal Investigator
- RPW Ground Segment Project Manager
- RPW Ground Segment Deputy Project Manager

7.2 Team collaboration tools

The team collaboration tools are S/W or services that allow teams involved in the ROC software development activities to collaborate.

The collaboration tools available are described in details in the ROC Software Development Plan (SDP) [RD2].

7.2.1 ROC issue tracker tool

The JIRA issue tracker tool permits to manage issues inside the ROC project. This tool must be used when:

- A new major development is started concerning a S/W, a database or a pipeline on the ROC servers.
- A new major S/W or database integration is performed on the ROC servers.
- An issue is found in a data set, S/W, a database or a pipeline (i.e. bug report).

A JIRA issue must only be closed if the corresponding task is fully completed (e.g., a bug is fixed, a S/W integration succeeds, etc.).

If an issue concerns software, its version shall be given as a label.

7.2.2 ROC Git repositories

The ROC team must uses Git [RD9] as a revision control system to store the S/W source files related to ROC Software System (RSS).

The ROC team must apply the Git branching model of V.Driessen [RD10] on its Git repositories storing software. Especially it means that any ROC software Git repository must contain at least the two following branches:

- A “master” branch used to store the software releases only (i.e., no development or testing versions). Especially, the “master” branch must only contain tagged version of the S/W releases as explained below.
- A “develop” branch used to as a main branch for the developments.

In addition, the ROC developers should apply the following rules on their local repositories:

- Create and use a new “feature/” type branch each time a new feature must be developed
- Create and use a new “release/” type branch before each S/W release

The ROC developers can use the “git-flow” tool (AVH edition) to work with Git.

In practice the ROC developer teams should never use the “master” branch. Committing on the “master” branch should be only performed by a single identified person. This person should be the only one in charge of performing the S/W release of a new stable version in the “master” branch for a given repository.

The tag name in the “master” branch must be the software version “X.Y.Z” as defined in the section 2.2.2. In the special case where the repository requires modifications that do not concern the software it-self, the tag shall be named as “X.Y.Z.K”, where “X.Y.Z” is the software version and “K” is an additional incremental integer starting at “1”. For instance, in the following sequence of tags: [“1.2.1” –> “1.2.1.1” --> “1.2.1.2” --> “1.3.0”], the software was only modified in the first and last commits.

Any Git commit must be commented to keep a track of the modification history.

7.2.3 ROC Wiki pages at LESIA

The ROC team should use the dedicated ROC Confluence page to disseminate information inside the ROC project.

7.3 ROC documentation support software guidelines

7.3.1 General

The ROC documents officially released must archived using the ROC Baghera web page.

Every new document written in the framework of the ROC project must apply the specific template provided in the Baghera Web page.

7.3.1 ROC documentation Versioning

The ROC documentation versioning must follow the RPW documentation convention defined in the PMP [RD11].

7.3.2 ROC Software development documentation

See Software Development Plan (SDP).

8 LIST OF TBC/TBD/TBWS

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

9 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-]CoIs
		Science-CoIs

INTERNAL

LESIA CNRS		

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
		A.VAIVADS
LPC2E		P. FERGEAU
		G. JANNET
		T.DUDOK de WIT
		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
		J. SOUCEK
		L.UHLIR
IWF		G.LAKY
		T.OSWALD
		H. OTTACHER
		H. RUCKER
		M.SAMPL
		M. STELLER
LPP		T.CHUST
		A. JEANDET
		P.LEROY
		M.MORLOT