RPW Operation Centre

RPW Calibration Software Interface Control Document

ROC-PRO-PIP-ICD-00037-LES
Iss.01, Rev.02

Prepared by:	Function:	Signature:	Date
Manuel Duarte Xavier Bonnin	RPW Ground Segment software engineer RPW Ground Segment Project Manager		DD/MM/YYYY
Verified by:	**Function:**	**Signature:**	**Date**
RPW RCS teams	N/A		Dd/mm/yyyy
Approved by:	**Function:**	**Signature:**	**Date**
Xavier Bonnin	RPW Ground Segment Project Manager		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
0	0	08/07/2015	Manuel Duarte	First draft (inherits from the RCS ICD for ROC-SGSE, ROC-TST-GSE-ICD-00023)
1	0	09/11/2016	Xavier Bonnin	First release
1	1	12/10/2017	Xavier Bonnin	- RCS CDF outputs path is now passed via the CLI specific input parameters, and not returned by the software in the stdout. - “outputs.release” attribute removed from the descriptor file - Add “outputs.template” attribute in the descriptor file - Rename “Exceptions and errors” section to “Exceptions handling” and update the content - Add section 4 on RCS-related data - Remove section on the interface from the pipeline side (this section is copied into the ROC Software System Design Document)
2	1		Xavier Bonnin	- Add section 3.3 “RCS reserved environment variables” - Add software version convention in section 3.2.2.2 “Release” - Add icd_version attribute in section 3.2.2.1 “Identification”

Acronym List

Acronym	Definition
BP	Basic Parameters
CDF	Common Data Format

CLI	Command Line Interface
CSV	Comma Separated Values
ICD	Interface Control Document
ID	Identifier
I/F	Interface
I/O	Input/Output
JSON	JavaScript Object Notation
ROC	RPW Operation Centre
RPW	Radio and Plasma Waves instrument
SCM	Search Coil Magnetometer
SGSE	Software Ground Support Equipment
SVN	SubVersioN
S/W	Software
TDS	Time Domain Sampler
LFR	Low Frequency Receiver
THR	Thermal Noise and High Frequency Receivers
WF	Waveform
XML	eXtended Markup Language

Table of Contents

List of figures

Aucune entrée de table d'illustration n'a été trouvée.
Dans le document, sélectionnez les mots à inclure dans la table des matières, puis, sur l'onglet Accueil, sous Styles, cliquez sur un style d'en-tête. Répétez l'opération pour chaque en-tête à inclure, puis insérez la table des matières dans le document. Pour créer manuellement une table des matières, sur l'onglet Éléments de document, sous Table des matières, pointez sur un style, puis cliquez sur la flèche vers le bas. Cliquez sur un des styles sous Table des matières manuelle, puis tapez les entrées manuellement.

List of tables

1 GENERAL

1.1 Scope of the Document

The RPW Calibration Software Interface Control Document (RCS ICD) describes the interface to be implemented in the RPW Calibration Software (RCS), in order to be autonomously run by the RPW data processing pipelines.

The RCS definition covers all of the programs that produce RPW calibrated science data at both analyser and sensor levels.

It is the responsibility of the teams in charge to deliver to the RPW Operations Centre (ROC) a RCS fully compliant with this ICD. The way the software (S/W) and associated materials shall be delivered to the ROC is described in the "ROC Engineering Guidelines for External Users" (REGU) document [RD4]. The REGU should be hence read in complement to the present document.

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	ROC-PRO-DAT-NTT-00006-LES/1/2 (draft)	ROC Data Products	X.Bonnin	
AD2				
AD3				
AD4				

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-SPC-00036-LES/1/0	ROC Software System Design Document	X.Bonnin, S.Lion	02/12/2016
RD2	ROC-TST-GSE-SPC-00004-LES/1/0	ROC SGSE Software Design Document	X.Bonnin	14/02/2016
RD3	ROC-TST-GSE-ICD-00023-LES/2/3	ROC-SGSE Calibration Software Interface Control Document	Manuel Duarte	10/05/2016
RD4	ROC-GEN-SYS-NTT-00019-LES/1/2	ROC Engineering Guidelines for External Users	X.Bonnin	15/12/2016
RD5	ROC-TST-GSE-SPC-00004-LES/1/1	ROC-SGSE Software Design Document	X.Bonnin	15/04/2016
RD6	ROC-GEN-SYS-NTT-00008-LES/1/2	ROC engineering guidelines	X.Bonnin	15/12/2016
RD7	ECMA-404.pdf/1st edition	The JSON Data interchange format	ECMA	10/2013
RD8	ROC-TST-SFT-SUM-00027-LES/1/1	RCS Interface Validation Tool User Manual	M.Duarte	06/05/2016
RD9				

2 CONSIDERATIONS & OBJECTIVES

2.1 RPW Calibration Software (RCS) implementation philosophy

During the Solar Orbiter mission, the ROC will be in charge to regularly produce and deliver to ESA calibrated science data for the RPW instrument. The task will be performed in an automated way by the ROC Operations and Data Pipeline (RODP) [RD1]; the main RPW data processing pipeline hosted on the ROC server at LESIA (Meudon, France). Nevertheless, it is the role of the RCS to calibrate the instrument science data and to generate the resulting output data files. Such functionality requires RCS to be integrated into the RODP.

Since the RCS programming language can differ from a S/W to another, the integration shall be done in a standard way. It implies to use a dedicated common interface in order to: (i) minimize the human intervention, (ii) ensure the traceability of S/W and its data products, (iii) mitigate the possible points of failure in the RODP.

In addition, the RCS will be also integrated into the ROC-SGSE [RD5]; the ROC pipeline in charge of processing the RPW data stored into the MEB Ground Support Equipment (GSE) database. In both cases, the RODP and ROC-SGSE, the same interface will be used, and the present ICD is thus applicable.

2.2 RCS deliverables

RCS will be supplied to the ROC by teams in charge. It is strongly recommended to reduce as much as possible the number of S/W to be delivered by each team in order to optimize the delivery, installation, execution and maintenance processes.

A single S/W by team, but producing several RPW data products, should be the optimal solution. Additionally, a given data product should be as much as possible generated by a given class or function in the RCS.

2.3 RCS data products

The RCS will produce RPW calibrated science data files at processing levels higher than level 1 (L1) and in the Common Data Format (CDF), as defined in [AD1]. Especially, the ROC does not expect that its pipelines handle other RCS data products than these data sets. RCS should not hence be used to generate derived science data products, such as summary plots. If it is the case, the RCS shall include an option to trigger the production of derived products separately.

The convention about the RCS data products (e.g., file naming, processing levels, data format and metadata description) are given in [AD1].

2.4 Objectives related to this document

In order for the teams to deliver RCS to be run by the ROC pipelines, it is necessary to define in detail the specification of the interface, and more particularly:

- How the RCS shall be called in a standard way
- How a ROC pipeline will identify and call the RCS
- How a ROC pipeline will identify and provide to RCS the required input data
- How and where the RCS will store the generated data products
- How a ROC pipeline will check that RCS run without error, and the resulting data products were correctly written.

This will be the purpose of the next sections. Section 3 describes the interface to be implemented by teams in charge on the RCS side. Section 4 presents conventions related to the RCS data.

3 DESCRIPTION OF THE INTERFACE TO BE IMPLEMENTED IN THE RCS

This section presents the interface that shall be found in the RCS, in order to be autonomously run by a ROC pipeline. This interface is composed of two main elements:

- A command line interface (CLI), that allows a ROC pipeline to call the RCS in a standard way
- A descriptor file (also called "descriptor"), that provides relevant information about the RCS to a ROC pipeline. The descriptor shall help the pipeline to identify the RCS, its inputs/outputs and to build the CLI calling sequence.

Besides, the RCS shall support the loading of reserved environment variables, as explained in the section 3.3.

3.1 Description of the RCS command line interface (CLI)

The RCS CLI is detailed in this section, namely: the general convention and the calling sequence formalism. A full example of such a CLI call is given in the appendix 5.1.

3.1.1 CLI executable convention

The CLI shall comply the following convention:

- The ROC pipelines shall be able to run the RCS by calling an executable - binary or script – file only.
- There shall be a single executable file per S/W.
- A team in charge of a RCS shall ensure that the executable file can be launched on the ROC server environment: a Linux Debian Operating System (OS) with Bourne-Again Shell (BASH), as a primary shell.
- The name of the executable file shall contain alphanumeric characters only. Only the "_" underscore and "-" hyphen characters must be used as delimiters to separate two fields in the name.

According to the convention above, every RCS CLI calling sequence shall start as:

```
$EXECUTABLE
```

Where `EXECUTABLE` is here the name of the executable.

Notes:

- **The interface does not work in the case where the executable is split in several words (for example, IDL batch commands usually work running "`idl -e program -args [...]`" calling sequence). In such a situation, the actual RCS launch command shall be wrapped into a BASH script. This script will be seen as the main RCS executable from the ROC pipelines point of view, and shall be thus compliant with this ICD.**

- **In the case where the executable is a Python script, a dedicated virtual environment shall be pre-installed by the ROC team with the support of the RCS team. The virtual environment shall be then activated prior to the software execution in a way defined by both teams.**

3.1.2 CLI input parameters

The CLI input parameters to be used to call the RCS executable, are described in the next section.

3.1.2.1 RCS function name

Each RCS shall contain one or more functions (or classes), which are used to create output data files. Each function has its own set of inputs/outputs that shall be known by the ROC pipelines to call it correctly. This will be the purpose of the S/W descriptor file, as explained in the section 3.2.

In order to allow the ROC pipelines to launch a given function of the S/W from the CLI, the executable shall provide the name of this function as a first argument.

It means that every calling sequence of a RCS executable shall start as followed:

```
$EXECUTABLE func1
```

Where `EXECUTABLE` is here the name of the executable, and `func1` is the name of the S/W function to run.

Note that this syntax is not mandatory in the specific case where input keywords are called, as explained in the section 3.1.2.4.

3.1.2.2 Common input parameters

In addition to the name of the function as a primary argument, the RCS executable calling sequence shall accept at least the following common input parameters:

`--log [/path/to/logfile]`: the absolute path **`/path/to/logfile`** to the log file

`--config [/path/to/configfile]`: if required, the absolute path **`/path/to/configfile`** to the S/W configuration file.

These input parameters shall take only one argument.

3.1.2.3 Specific input parameters

The specific input parameters shall permit to pass as input arguments, the path of the input/output RPW data files to be read/saved by the RCS.

The specific input parameters shall be called in the CLI following the convention above:

- The specific input parameter calling sequence shall be formed by a *"--flag value*" pair, where "*flag*" must be a lowercase string, containing only alphanumeric characters and the underscore '_' as a separator, and preceded by the double hyphen prefix '--'. The "*value*" is the parameter argument value to be passed to the S/W function.
- The "*flag*" name shall be unique for a given S/W function calling sequence.

A typical RCS call, mixing common and specific parameters of a given `func1` function, looks like:

```
$EXECUTABLE func1 --input_1 /path/to/inputfile1 --input_2 /path/to/inputfi
le2 --output_1 /path/to/outputfile1 --log /path/to/logfile
```

The order of the input parameters is not important, except the function name, which shall be always the first argument to be called.

The way a ROC pipeline automatically identifies a function and its inputs/outputs from information in the descriptor file, and how it builds the corresponding calling sequence is explained in [RD1].

Notes:

- **The path to the master CDF or calibration table files, shall not be passed as input arguments to the CLI, but can shall be loaded by the S/W using the dedicated environment variables (see section 3.3).**
- The path to the configuration file can be provided, using the dedicated `--config` input parameter.

3.1.2.4 Input keywords

The RCS executable calling sequence shall support at least the following input keywords:

`--identification`: return information about the S/W. The format and content of the returned stream shall correspond to the `"identification"` JSON object in the S/W descriptor file. (See the section 3.2 for more details about the descriptor file.)

`--version`: return information about the current S/W release. The content of the returned stream shall comply the `"release.version"` attribute value in the S/W descriptor file.

`--help`: display a help message, containing information about how to call the S/W.

For instance, to print information about the current version of the S/W:

```
$EXECUTABLE --version
```

Where `EXECUTABLE` is name of the executable file.

Note:

- Input keywords shall not take argument.
- Calling the input keyword shall not require to provide the function name `func1` as a first argument.

3.2 RCS descriptor file specification

The RCS descriptor file (or simply descriptor) is the second key element of the interface with the CLI. It contains information about the RCS that helps the ROC pipelines to automatically identify:

- The S/W it-self (e.g., name, identifier, description)
- The release (e.g., version number, release date, author)
- The execution environment (e.g., setup script, executable, configuration, etc.)
- A detailed description of the output files produced by the S/W (name, identifier, description, release, etc.)
- The list of calling sequences for each S/W function and the corresponding inputs and outputs

The information is used, among others, by the ROC pipelines to automatically build the CLI calling sequence for a given RCS function, and to check the output file creation.

The content of the descriptor is checked and inserted into the ROC pipeline database during the registration step (see [RD1] for more details). A ROC pipeline is hence not able to run a RCS delivered without or with a badly formatted descriptor file.

An example of such a descriptor file can be found in the appendix 5.1.

3.2.1 General convention

The descriptor files shall be delivered with the RCS, following the convention below:

- The descriptor file shall be written in the JSON format [RD7]
- The descriptor file shall be placed in the S/W main directory
- There shall be only one descriptor file per S/W
- The descriptor file shall be named "***descriptor.json***"
- All of the paths defined in the descriptor file shall be relative to the S/W main directory.

3.2.2 Descriptor file structure

The structure of the descriptor file is detailed in the following sections.

3.2.2.1 Identification

Each S/W shall be identified in the pipeline by the attributes provided in the `"identification"` JSON object:

- `"project"`**:** name of the project. It shall be "ROC".
- `"name"`**:** full name of the RCS, to be human readable.
- `"identifier"`**:** a name used as a unique reference by the ROC pipeline to identify the S/W. This ID shall contain alphanumerical uppercase characters only. The hyphen or underscore characters shall be used separators only. The list of possible ID is: "THR_CALBAR", "LFR_CALBUT", "TDS_CALBA", "SCMCAL" and "BICAS".
- `"description"`**:** Short description of the S/W.
- `"icd_version"` : Version (i.e., issue.revision) of the RCS ICD

3.2.2.2 Release

The `"release"` object shall inform about the current S/W release. It shall contain the following attributes:

- `"version"`**:** Current version of the S/W. 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 software (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 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.

- `"date"`**:** Date of the release of the S/W in the format 'YYYY-MM-DD', where 'YYYY', 'MM' and 'DD' are respectively the year, month and date of the release.
- `"author"`: Name of the person, team or entity responsible of the release
- `"contact"`: contact of the author (e.g., email or address)
- `"institute"`: Name of the institute that delivers the release
- `"modification"`: a string containing the list of S/W modifications in the current release.

In addition, the `"release"` object can provide the following optional attributes:

- `"reference"`: reference for the RCS user manual documentation
- `"source"`: indication for an online URL of the RCS source files (e.g., Git repository URL) .

3.2.2.3 Environment

The attributes defined in the object `"environment"` are used for setting the RCS environment variables, and reset it to the initial state. It shall contain:

- `"executable"`: Relative path to the S/W executable file to be called by the ROC pipeline via the CLI. It corresponds to the **`EXECUTABLE`** in the calling sequence example given in the section 3.1.2.3.

In addition, the `"environment"` object can contain the following optional attributes:

- `"configuration"`: Relative path to a possible configuration file that it is required by the S/W to run correctly. Note that if the **`--config`** common input parameter is provided in the CLI, it supersedes the value given in this attribute.
- `"activation"`: Relative path to the directory containing script(s) used to set up the environment, or to be run prior to the S/W execution. The script(s) shall work in the BASH shell and shall run without any input argument.
- `"deactivation"`: Relative path to script(s) used to reset the environment to its initial state. The script(s) shall work in the BASH shell and shall run without any input argument.

3.2.2.4 Modes

The `"modes"` element is a JSON objects array, which contains the list of S/W functions that can be run by the pipeline, when it calls the executable file.

For each S/W function, its name, its purpose and the list of input/output RPW datasets to be read/saved shall be supplied. It allows the pipeline to build the S/W function input calls automatically, and to check if the expected output data files are correctly saved.

Each function listed in the `"modes"` object shall contain the following attributes and JSON arrays:

- `"name"`: The name of the function. It shall be unique. It corresponds to the **`func1`** in the calling sequence as presented in the section 3.1.2.1.
- `"purpose"`: A short description of the purpose of the function
- `"inputs"`: A JSON array containing the list of the RPW dataset(s), to be provided as input arguments to the function
- `"outputs"`: A JSON array containing the list of the output RPW dataset(s) to be produced by the function.

The purpose and the content of the `"inputs"` and `"outputs"` objects are detailed in the two next sections.

3.2.2.4.1 Inputs

The "inputs" JSON array contains the list of input RPW dataset(s) to be passed to a given function func1 of the RCS.

The following rules shall be applied for the "inputs" array entries:

- There shall be one entry in the array per specific input parameters found in the RCS CLI. E.g., one entry for "--input_1" and one entry for "--input_2", in the RCS CLI example of the section 3.1.2.3.
- The name of an entry shall be the same than the specific input parameter keyword used in the RCS CLI, but without the "--" prefix. E.g., the name of the "--input_1" and "--input_2" entries are "input_1" and "input_2" respectively, in the RCS CLI example of the section 3.1.2.3.
- Each entry shall be a JSON object containing the mandatory attribute "identifier". This attribute shall provide the RPW dataset ID of the input data file, passed to the corresponding RCS CLI specific input parameter keyword.

According to the example of RCS CLI calling sequence in the section 3.1.2.3, the "inputs" array will be structured as followed:

```
"inputs":[
      "input_1":{
                  "identifier":"RPW_DATASET_ID1”
               },
      "input_2":{
                  "identifier":"RPW_DATASET_ID2"
               }
         ],
```

Where "RPW_DATASET_ID1” and "RPW_DATASET_ID2” are the RPW dataset ID of the input data files, to be passed to a given function func1, respectively as --input_1 /path/to/inputfile1 and --input_2 /path/to/inputfile2 in the RCS CLI calling sequence.

Notes:

- The list of RPW dataset ID are presented in [AD1].
- **The "inputs" array shall not contain any path to an input data file**. The input data file path will be automatically built and passed to the RCS CLI by the ROC pipelines them-selves, from the information provided in the descriptor file.
- The teams are in charge to define the list of inputs for their S/W in the descriptor file. This work shall be done in collaboration with the other teams and the ROC. Especially, the ROC shall supply as soon as possible information required to develop and maintain the RCS and their descriptor file, including an up-to-date list of the RPW datasets ID and versions used by the RCS.
- The pipeline will not run a given function of the S/W in the case where, at least one of the input files of the expected RPW input dataset, as defined in the "inputs" of the

descriptor, is not found. It should be taken into account in the S/W architecture design, especially when defining the number of available modes.

3.2.2.4.2 Outputs

The `"outputs"` JSON array has the same objective than the `"inputs"` JSON array, but for the software outputs. It contains the list of output RPW dataset(s) generated by a given function `func1` of the RCS. Additionally, it provides descriptive information about the output RPW dataset(s) saved by the RCS.

The following rules shall be applied for the `"outputs"` array entries:

- There shall be one entry in the array per specific input parameters found in the RCS CLI. E.g., one entry for `"--output_1"`, in the RCS CLI example of the section 3.1.2.3.
- The name of an entry shall be the same than the specific input parameter keyword used in the RCS CLI, but without the "`--`" prefix. E.g., the name of the `"--output_1"` entry is `"output_1"`, in the RCS CLI example of the section 3.1.2.3.
- Each entry shall be a JSON object containing the attributes listed below.

Each entry listed in `"outputs"` shall contain the following mandatory attributes:

- `"identifier"`: The RPW dataset ID of the output data file(s), passed to the corresponding RCS CLI specific input parameter keyword.
- `"name"`: a more human-readable name for the dataset, not necessarily unique.
- `"description"`: a short description of the dataset.
- `"level"`: the processing level of the dataset. The authorized values are "L1R" or "L2".

In addition, the `"outputs"` object can contain the following optional attributes:

- `"template"`: file name to reference a data schema used for the output generation, as an indication for the ROC team. It should provide the master CDF name.
- `"reference"`: Any reference (paper, note, Web site) giving a description of the output (e.g., RPW Data Product Description Document reference).

According to the example of RCS CLI calling sequence in the section 3.1.2.3, the `"outputs"` array will be structured as followed:

```
"outputs":[
      "output_1":{
                  "identifier":"RPW_DATASET_ID3"
               }
         ],
```

Where `"RPW_DATASET_ID3"` is the RPW dataset ID of the output data file, generated by a given function `func1` and for which the target path `/path/to/outputfile1` is passed as `--output_1 /path/to/outputfile1` in the RCS CLI calling sequence.

Notes:

- The list of RPW dataset ID are presented in [AD1].
- **The `"outputs"` array shall not contain any path to an output data file**. The output data file path will be automatically built and passed to the RCS CLI by the ROC pipelines them-selves, from the information provided in the descriptor file.
- The ROC pipeline will perform an automated validation of the S/W descriptor file at each new release, in order to check that the content of the file is consistent with the information in the pipeline database. In particular, it will verify that the datasets declared in the `"modes"` object are all defined in the descriptor and across S/W.

3.3 ROC reserved environment variables

Table below gives the list of the environment variables defined by the ROC pipelines before launching the RCS. These variables shall not be created or modified by the RCS.

Environment variable	Description	Need	Comment
ROC_PIP_NAME	Name of the ROC pipeline. Must be “RGTS” for the ROC-SGSE and “RODP” for the RODP	M	This variable allows the RCS to identify by which ROC pipeline it is called.
ROC_RCS_CAL_PATH	Absolute path to the directory containing the RCS calibration table files.	M	This variable allows RCS to load the calibration tables.
ROC_RCS_MASTER_PATH	Absolute path to the directory containing the RCS master CDF files	M	This variable allows RCS to load the master CDF files.
ROC_PIP_VERSION	ROC pipeline version	O	
ROC_RCS_ABS_PATH	Absolute path to the directory containing the RCS	O	

Table 1. ROC reserved environment variables.

3.4 RCS exception handling mechanism

The RCS shall include a mechanism to handle exceptions that can occur during the execution. The exceptions to be notified to the ROC pipelines, and the expected actions to be performed by the RCS are explained in this section.

3.4.1 RCS exception level definition

Table below gives the definition of the exception levels to be implemented in the RCS.

Exception level	Definition
Medium	It is a "warning" exception level. It alerts about a potential issue, but does not necessarily require stopping the execution.
High	It is an "error" exception level, which requires aborting the execution.

Table 2. RCS exception levels.

In both case the "medium" and "high" exception levels, the RCS shall return a message detailing the reason why the exception has been triggered.

In addition, a WARNING/ERROR level event shall be also reported by the RCS into its log file for each "medium"/"high" event (see the section 4.2.4 for the definition of log levels).

3.4.2 Identified exceptions

A RCS shall raise an exception if it encounters one of the following events.

Event	Exception level	Comment
Input(s) data file(s) cannot be found or read correctly	High	
Input(s) file(s) inconsistency	High	It concerns the case where the input file(s) is/are not as expected.
Additional file(s) cannot be found or read correctly	High	It can concern configuration. file, master CDF and/or calibration table file(s)
Output(s) file(s) cannot be written correctly	High	e.g., bad write permissions, output directory not found, etc.
Log file cannot be written correctly	High	e.g., bad write permissions, output directory not found, etc.
S/W execution environment not defined	High	e.g., environment variables required by the RCS are not defined correctly
Inconsistencies of metadata for current output dataset	Medium	The metadata of the input files or master CDF are not as expected

Table 3. RCS exception list.

3.4.3 Output error code convention

If a "high" level exception is raised, the RCS shall exit with an error code 1. The error information (i.e., code, type of error and message) shall be written in the standard error (*stderr*). It will be then caught by the ROC pipeline and stored into its database for investigations.

If the error code is 0, the ROC pipeline will consider that the S/W execution has succeeded. Any other code value will stop the job in the ROC pipeline and emit an error.

4 RCS-RELATED DATA CONVENTIONS

This section lists conventions concerning RCS-related data.

4.1 RCS CDF data convention

4.1.1 General convention and metadata definition.

The RCS CDF input/output data must comply the convention described in [AD1].

4.1.2 CDF meta-data setting convention

The table below gives the list of CDF meta-data - global or variable attributes - that must be updated by the RCS, when producing the output CDF files using the master CDF.

Attribute name	Attribute type	Comment
Generation_date	Global	
Logical_file_id	Global	
Parents	Global	
Parent_version	Global	
Software_name	Global	
Software_version	Global	
SPECTRAL_RANGE_MAX	Global	
SPECTRAL_RANGE_MIN	Global	
TIME_MAX	Global	
TIME_MIN	Global	
CAL_ENTITY_NAME	Global	
CAL_ENTITY_AFFILIATION	Global	
CAL_EQUIPMENT	Global	
CALIBRATION_TABLE	Global	Only concerns the L1R/L2 datasets
CALIBRATION_VERSION	Global	Only concerns the L1R/L2 datasets
SCALEMAX	Variable	
SCALEMIN	Variable	

Table 4. CDF metadata setting list.

Note that the "FILE_UUID", "Data_version", "Generated_by" and "Validate" global attributes will be updated by the ROC pipeline it-self, just after the creation of the RCS output file.

4.1.3 CDF zVariables setting convention

TBW (quality_bitmask/quality_flag)

4.2 RCS log file convention

This section describes the convention concerning the log file, to be generated by the RCS during its execution at the ROC site.

4.2.1 General convention

Teams in charge must follow the following log file related general convention:

- There must be one log file per S/W.
- The S/W it-self must create the log file, from the path and name provided by the argument of the **`--log`** input parameter. If a log file with the same name already exists in the log directory, the S/W must not create a new log file, but must append new entries into the existing file.

Note that the ROC might need at some points to move an existing log file from the log directory. In this case, the S/W will then create a new log file.

4.2.2 File naming convention

The RCS log file must follow the file naming convention:

[*rcs_name*].log

Where [*rcs_name*] is the name of the RCS in lower case characters, with only hyphens "-" or underscores "_" as separators (e.g., tds_calba.log).

It must be noticed that, in practice, this naming convention is only applicable to the ROC team, since the log filename is an input argument of the RCS.

4.2.3 Format and content

The log file must be an ASCII format file. It must contain enough information about the execution of the RCS, in order to allow the ROC and the teams in charge to monitor the S/W behaviour and to diagnose unexpected events.

Each line of the file must start with the following prefix:

[*YYYY:MM:DD*] [*hh:mm:ss*] -- [*log_level*] -- [*message*]

Where [*YYYY:MM:DD*] and [*hh:mm:ss*] are the date (*YYYY*=4-digits year, *MM*=2-digits month and *DD*=2-digits day of month) and time (*hh*=2-digits hours, *mm*=2-digits minutes and *ss*=2-digits seconds) of the log event. [*log_level*] is the level of severity of the log event (see table in the next section) and [*message*] is a one line description of the log event.

4.2.4 Log event severity levels

Table below gives the list of event severity levels to be used in the RCS log file.

Severity level	Definition	Actions to be performed by the RCS
DEBUG	Debug event (only used in debug mode)	No specific action
INFO	Normal event (e.g., RCS start/end times, routine tasks information, etc.)	No specific action
WARNING	Event that requires attention, but does not	No specific action

	compromise the software execution or data production	
ERROR	Event that requires special attention, and compromises the data production or the software execution (e.g., unexpected values in the data, not input data file found, env. variable not well defined, etc.)	Stop the software execution and exit with an exception (error code 1)

Table 5. RCS log event severity levels.

5 APPENDICES

5.1 Example of a RPW S/W descriptor file

Here is an example of a RPW descriptor file for the THR Calibration software (THR_CALBAR) RCS.

```
{
    "identification": {
        "project": "ROC",
        "name": "THR CALBAR",
        "identifier": "THR_CALBAR",
        "description": "The RPW TNR-HFR CALibration softwARe (THR-CALBAR)
produces RPW TNR-HFR calibrated science data",
        "icd_version": "1.2"
    },
    "release": {
        "version": "1.1.0",
        "date": "2015-07-29",
        "author": "Antonio Vecchio",
        "contact": "antonio.vecchio@obspm.fr",
        "institute": "LESIA",
        "modification": "Update tnr_l2_calibration function",
    },
    "environment": {
        "activation": "scripts/setup_thr-calbar_env.sh",
        "deactivation": "scripts/unset_thr-calbar_env.sh"
```

```
        "executable": "bin/thr-calbar-api",
        "configuration": "config/thr-calbar_configuration.json"

    }
    "modes": [
        {
            "name": "tnr_l2_cal",
            "purpose": "Produce calibrated science TNR data file at analys
er level (L2)",
            "inputs": {
                "input_l1_tnr": {
                    "identifier": "SOLO_L1_RPW-TNR-SURV"
                }
            },
            "outputs": {
                "output_l2_tnr": {
                    "identifier": "SOLO_L2_RPW-TNR-L2-SURV",
                    "name": "RPW TNR L2 survey data",
                    "description": "RPW TNR L2 science data in survey mode
",
                    "level": "L2",
                }
            }
        },
        {
            "name": "hfr_l2_calibration",
            "purpose": "Produce calibrated science HFR data file at analys
er level (L2)",
            "inputs": {
                "input_l1_hfr": {
                    "identifier": "SOLO_L1_RPW-HFR-SURV"
                }
            },
            "outputs": {
                "output_l2_hfr": {
                    "identifier": "SOLO_L2_RPW-HFR-SURV",
                    "name": "RPW HFR L2 survey data ",
                    "description": "RPW TNR L2 science data in survey mode
",
                    "level": "L2"
                }
            }
        }
    ]
}
```

5.2 Examples of a CLI calling sequence

With the following configuration, the pipeline will call the THR_CALBAR S/W for the calibration mode `tnr_l2_cal` as (assuming that the environment has been setup previously and that the path to the S/W directory is contained in the `$SW_ROOT` variable):

```
$$SW_ROOT/bin/thr-calbar-api tnr_l2_cal --input_l1_tnr
/path/to/input_l1_tnr_cdf_file \
    --output_l2_tnr /path/to/output_l2_tnr_cdf_file --log /path/to/log \
    --config $SW_ROOT/config/configuration.file
```

If the S/W ends without error, then the pipeline will then search for a file at `/path/to/output_m2_tnr_cdf_file`.

If an error occurs, the S/W must stop with an error code greater than 0 and a relevant message in *stderr*. If the error can be caught by the pipeline, it will store not necessarily useful information.

For the identification command, the S/W will return the following JSON message in the *stdout*:

```
$ $SW_ROOT/bin/thr-calbar-api --identification
{
        "project": "ROC",
        "name": "THR CALBAR",
        "identifier": "THR_CALBAR",
        "description": "The RPW TNR-HFR CALibration softwARe (THR-CALBAR)
produces RPW TNR-HFR calibrated science data",
        "icd_version": "1.2"
}
```

And for the version command:

```
$ $SW_ROOT/bin/thr-calbar-api --version
{
    "version":"1.1.0"
}
```

5.3 ROC S/W descriptor file validation

A JSON format file, as for XML, can be validated against a schema. A JSON schema is following the specifications from http://json-schema.org/draft-04/schema. Dedicated documentation can be found at https://spacetelescope.github.io/understanding-json-schema.

The JSON schema for the ROC S/W descriptor file is described below. It will be used by the ROC pipeline to automatically validate a S/W descriptor file after delivery.

```
{
    "$schema":"http://json-schema.org/draft-04/schema#",
    "definitions":{
        "input":{
            "type":"object",
            "patternProperties":{
                "^[A-Za-z][\\w-]+$":{"$ref": "#definitions/input_dataset"}
            },
            "additionalProperties":false
        },
        "output":{
            "type":"object",
```

```
            "patternProperties":{
                "^[A-Za-z][\\w-]+$":{"$ref":
"#definitions/output_dataset"}
            },
            "minProperties":1,
            "additionalProperties":false
        },
        "mode":{
            "type": "object",
            "properties": {
                "name": {
                    "type": "string",
                    "pattern": "^[A-Za-z][\\w-]+$"
                },
                "purpose": {"type": "string"},
                "inputs": {"$ref": "#definitions/input"},
                "outputs": {"$ref": "#definitions/output"}
            },
            "required": ["name", "purpose", "inputs", "outputs"],
            "additionalProperties": false
        },
        "release": {
            "type": "object",
            "properties": {
                "author": {"type": "string"},
                "date": {
                    "type": "string",
                    "format": "date-time"
                },
                "version": {
                    "type": "string",
                    "pattern": "^(\\d+\\.)?(\\d+\\.)?(\\d+)$"
                },
                "file": {"type": "string"},
                "institute": {"type": "string"},
                "reference": {"type": "string"},
                "source": {
                    "type": "string",
                    "format": "uri"
                },
                "contact": {
                    "type": "string",
                    "format": "email"
                },
                "modification": {"type": "string"}
            },
            "required": ["author", "date", "version"],
            "additionalProperties": false
        },
        "output_dataset": {
            "type": "object",
            "properties": {
```

```
                "identifier": {
                    "type": "string",
                    "pattern": "^[\\w-]+$"
                },
                "name": {"type": "string"},
                "level": {"enum": ["LZ", "L0", "L1", "L1R", "L2", "AUX",
"LL01", "LL02", "HK"]},
                "description": {"type": "string"},
            },
            "required": ["identifier", "name", "level", "description"],
            "additionalProperties": false
        },
        "input_dataset": {
            "type": "object",
            "properties": {
                "identifier": {
                    "type": "string",
                    "pattern": "^[\\w-]+$"
                },
            },
            "required": ["identifier"],
            "additionalProperties": false
        }
    },
    "type": "object",
    "properties": {
        "identification": {
            "type": "object",
            "properties": {
                "project": {"type": "string"},
                "name": {"type": "string"},
                "identifier": {"type": "string"},
                "description": {"type": "string"},
                "icd_version": {"type": "string"}
            },
            "required": ["project", "name", "identifier", "description",
"pipeline"],
            "additionalProperties": false
        },
        "release": {"$ref": "#definitions/release"},
        "environment": {
            "type": "object",
            "properties": {
                "activation": {"type": "string"},
                "deactivation": {"type": "string"}
            },
            "additionalProperties": false
        },
        "executable": {"type": "string"},
        "configuration": {"type": "string"},
        "modes": {
            "type": "array",
```

```
            "items": {"$ref": "#definitions/mode"}
        }
    },
    "additionalProperties": false,
    "required": ["identification", "release", "environment", "executable",
"modes"]
}
```

6 LIST OF TBC/TBD/TBWS

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

7 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		M.MAKSIMOVIC
	x	X.BONNIN
	x	S.LION
	x	Q-N.NGUYEN
		D.BERARD
	x	A.VECCHIO
	x	L.MATTEINI
		N.FULLER

LESIA CNRS		

EXTERNAL (To modify if necessary)

CNES		C. FIACHETTI
		E.LORFEVRE
		R.LLORCA-CEJUDO
		E.LOURME
		M-O. MARCHE
		J.M.TRAVERT
		J.PANH
	x	D.RAULIN
IRFU		L. BYLANDER
		C.CULLY
	x	A.ERIKSSON
		SE.JANSSON
	x	A.VAIVADS
LPC2E	x	J-Y.BROCHOT
		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
	x	D.PISA
		O.SANTOLIK
	x	J. SOUCEK
		L.UHLIR
IWF		G.LAKY
		T.OSWALD
		H. OTTACHER
		H. RUCKER
		M.SAMPL
		M. STELLER
LPP	x	T.CHUST
		A. JEANDET
	x	B.KATRA
		M.MORLOT
	x	R.PIBERNE