RPW Operation Centre

ROC-SGSE Calibration Software Interface Control Document

ROC-TST-GSE-ICD-00023-LES
Iss.02, Rev.02

Prepared by:	**Function:**	**Signature:**	**Date**
Manuel Duarte	RPW Ground Segment Software Engineer		06/05/2016
Verified by:	**Function:**	**Signature:**	**Date**
Xavier Bonnin	RPW Ground Segment Software Manager		Dd/mm/yyyy
Approved by:	**Function:**	**Signature:**	**Date**
Yvonne de Conchy	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
1	0	09/08/2015	Xavier Bonnin	First release
2	0	18/11/2015	Manuel Duarte Xavier Bonnin	Major changes in the document structure. Update of the interface proposal (command line execution becomes the primary and unique option)
2	1	16/12/2015	Xavier Bonnin	Section 4.5: Specify the pipeline behavior in case of missing inputs.
2	2	06/05/2016	Manuel Duarte	Rename to ROC-SGSE Calibration Software Interface Control Document. Update Pipeline interface section

Acronym List

Acronym	Definition
BASH	Bourne-Again SHell
BP	Basic Parameters
CDF	Common Data Format
CLI	Command Line Interface
CSV	Comma Separated Values
ICD	Interface Control Document
ID	Identifier
I/O	Input/Output
JSON	JavaScript Object Notation
LFR	Low Frequency Receiver
OS	Operating System
RCS	RPW Calibration Software

RDPP	RPW Data Processing Pipeline
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
THR	Thermal Noise and High Frequency Receivers
WF	Waveform
XML	eXtended Markup Language

Table of Contents

List of Figures

List of Tables

1 GENERAL

1.1 Scope of the Document

The ROC-SGSE Calibration Software Interface Control Document (ICD) describes all of the interfaces to be implemented in order to run the RPW calibration software (RCS) inside the ROC-SGSE [RD1].

The RCS definition covers all of the programs allowing the ROC-SGSE to produce RPW calibrated science data at both analyser and sensor levels. It is the responsibility of each RPW team to develop, test, validate, maintain and deliver to the ROC fully operational software (S/W) for its own sub-system.

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-GEN-SYS-PLN-00015-LES/1/1	ROC Software Development Plan	X.Bonnin	15/10/2015
AD2				
AD3				
AD4				
AD5				

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-TST-GSE-SPC-00004-LES/0/2	ROC SGSE Description	X.Bonnin	06/11/2015
RD2	ROC-TST-GSE-SPC-00017-LES/1/2	Data format and metadata definitions for the ROC-SGSE data	X.Bonnin	28/10/2015
RD3	cdf360ug.pdf/3/6	CDF User's Guide	NASA/GSFC	20/02/2015
RD4	ECMA-404.pdf/1st edition	The JSON Data interchange format	ECMA	10/2013
RD5	ROC-OPS-PIP-NTT-00008-LES/1/1	ROC Engineering Guidelines	X.Bonnin	18/11/2015
RD6	ROC-OPS-PIP-NTT-00019-LES/1/1	ROC Engineering Guidelines for External Users	X.Bonnin	18/11/2015

2 CONSIDERATIONS & OBJECTIVES

2.1 Software implementation philosophy

During the on-ground calibration tests at system level, the ROC-SGSE shall be able to produce RPW science calibrated data. Such functionality requires RCS to be implemented in the ROC-SGSE pipeline in a standard way in order to: (i) minimize the human intervention; (ii) ensure the traceability of S/W and data; (iii) maximize the re-use of the RCS development for the RPW science data processing during the Solar Orbiter mission.

The implementation processes explained in this document shall be in agreement with the conventions defined in [RD6].

2.2 Deliverables

The RPW sub-system teams will supply RCS to the ROC. 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 and maintenance processes.

A single S/W by team but producing several RPW data products is the optimal solution for the ROC team.

2.3 Data products

The data products generated by the RCS will be typically RPW calibrated science data files in the CDF format.

A given data product should be as much as possible generated by a given class or function in the RCS.

Derived science data products such as summary plots should be as much as possible generated by other programs from the resulting CDF calibrated data files. The teams are strongly encouraged to also provide to the ROC these third-party programs.

The full list of datasets to be produced by the ROC-SGSE, as well as the data processing levels to be applied, are defined in [RD2].

2.4 Software interfaces

In order for the teams to be able to produce and configure their RCS to be run at the ROC, it is necessary to define in detail a number of interfaces. Specifically we must define:

- How the RCS shall be called in a standard way
- How the ROC-SGSE pipeline will identify and call the S/W
- How the ROC-SGSE pipeline will identify and provide to S/W the required input data
- How and where the S/W will store the generated data products
- How the ROC-SGSE pipeline will check that S/W run without error, and the resulting data products were correctly written.
- How, and in what form, each team will deliver the RCS to the ROC, including the S/W documentation and required materials (e.g., calibration tables, CDF master files, etc.)

This will be the purpose of the next sections. Section 3 describes how the RCS shall be called using a standard command line interface (CLI). Section 4 presents how the ROC-SGSE

pipeline will manage the RCS execution using the CLI, and which additional material must be delivered by the teams with their S/W to have a fully compatible CLI.

3 DESCRIPTION OF THE SOFTWARE COMMAND LINE INTERFACE

The purpose of this section is to describe how to call the RCS using the ROC-SGSE CLI.

3.1 General conventions

The CLI shall comply the following conventions:

- The ROC shall be able to run the RCS by calling an executable - binary or script – file.
- There must be only one executable file per S/W.
- The teams shall ensure that the RCS executable files can be launched on the ROC server environment: a Linux Debian Operating System (OS) with Bourne-Again Shell (BASH) as a primary shell (see [AD1] for details).
- The name of the executable shall contain alphanumeric characters only.

The CLI inputs/outputs to be used to call the RCS executable is described in the next section.

3.2 Input parameters

3.2.1 Software function name

Each S/W shall contain one or more functions (or classes) that are used to perform the expected tasks, which are typically the creation of calibrated science data files. Each function has its own set of inputs/outputs that must be known by the ROC-SGSE to call it correctly. This will be the purpose of the S/W descriptor file, which is described in details in the section 4.3.

In order to allow the ROC-SGSE 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 S/W executable calling sequence shall start as followed:

```
$EXECUTABLE func1
```

, where `EXECUTABLE` is here the name of the S/W executable, and `func1` is the name of the S/W function to be run. This rule is nevertheless not verified in the case of the input keywords, as explained in the section 3.2.4.

3.2.2 Common input parameters

In addition to the name of the function as a primary argument, every RCS executable calling sequence shall support the following inputs parameters that take only one argument each:

`--output [/path/to/outputdir]`: the absolute path **`/path/to/outputdir`** to the directory where output files will be saved.

`--log [/path/to/logdir]`: the absolute path **`/path/to/logdir`** to the directory where the log file(s) will be saved.

`--config [/path/to/configfile]`: the absolute path **`/path/to/configfile`** to a possible configuration file.

3.2.3 Specific input parameters

Specific input parameters can be defined for each function of a S/W. These parameters shall provide to a given function, the path to the input data files required to perform the expected task; typically the production of calibrated science data files in CDF format.

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

- The specific input parameter calling sequence shall be formed by a *"--flag value*" pair, where "*flag*" shall be a lowercase string, containing only alphanumeric characters and the underscore '_' as separator, and preceded by the double hyphen prefix '--'. The "*value*" is the parameter 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 some common and specific parameters of a given `func1` function is:

```
$EXECUTABLE func1 --input_1 /path/to/input1 --input_2 /path/to/input2 \
    --output /output/directory/path --log /path/to/log
```

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

The way the ROC-SGSE automatically identifies a function and its inputs/outputs, and how it builds the corresponding calling sequence is explained in the section 4.

Note that additional inputs such as master CDF or calibration table files shall not be passed as input parameters to the CLI, but can be provided in a configuration file to be loaded by the S/W it-self, using the `--config` parameter.

3.2.4 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 4.3 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.

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

```
$EXECUTABLE --version
```

, where `EXECUTABLE` is name of the executable file.

Calling the input keyword does not require to provide the function name `func1` as a first argument.

3.3 Standard output

If the S/W ends with no error, the standard output (*stdout*) shall return a list of the output files produced in the JSON format. For instance:

```
{
    "output_cdf1": "output_filename1.cdf",

    "output_cdf2": "output_filename2.cdf",
}
```

, where `"output_cdf1"` and `"output_cdf2"` are the name of the `"outputs"` JSON objects as defined in the S/W descriptor file, `"output_filename1.cdf"` and `"output_filename2.cdf"` are the name of the output files produced by the current S/W function.

The output filenames listed in the *stdout* will be then caught by the ROC-SGSE pipeline, and use with the value provided by the common input parameter `--output [/path/to/outputdir]` (see Section 3.2.2), in order to reach the data files `/path/to/outputdir/output_filename1.cdf` and `/path/to/outputdir/output_filename2.cdf`, written by the S/W.

Note that the S/W will have the task to name the output CDF files it-self. This can be done using the right global attributes referenced in the corresponding master CDF.

3.4 Exceptions and errors

If a problem happens, the S/W must exit with an error code, whose detail (such as message, explanation, etc.) is written in the standard error (*stderr*). This message will then be caught by the ROC-SGSE pipeline, and stored into the ROC database for investigations.

If the error code is 0, the ROC pipeline will consider that the S/W didn't have a problem and continue. Any other value will stop the job in the ROC pipeline and emit an error.

A full example of a CLI calls in the framework of the ROC-SGSE pipeline interface is given in the appendix 5.1.

4 ROC-SGSE PIPELINE SOFTWARE CALLING INTERFACE

In order to allow ROC-SGSE to automatically identify and call RCS, a dedicated interface is required on the pipeline side. To be fully operational this interface relies on the CLI and additional key files called descriptors, which must be written by the teams and delivered with their S/W following some specific rules.

The basic principle, the procedures as well as the descriptor files related to this pipeline interface are described in details in this section.

4.1 Principle of the ROC-SGSE pipeline interface

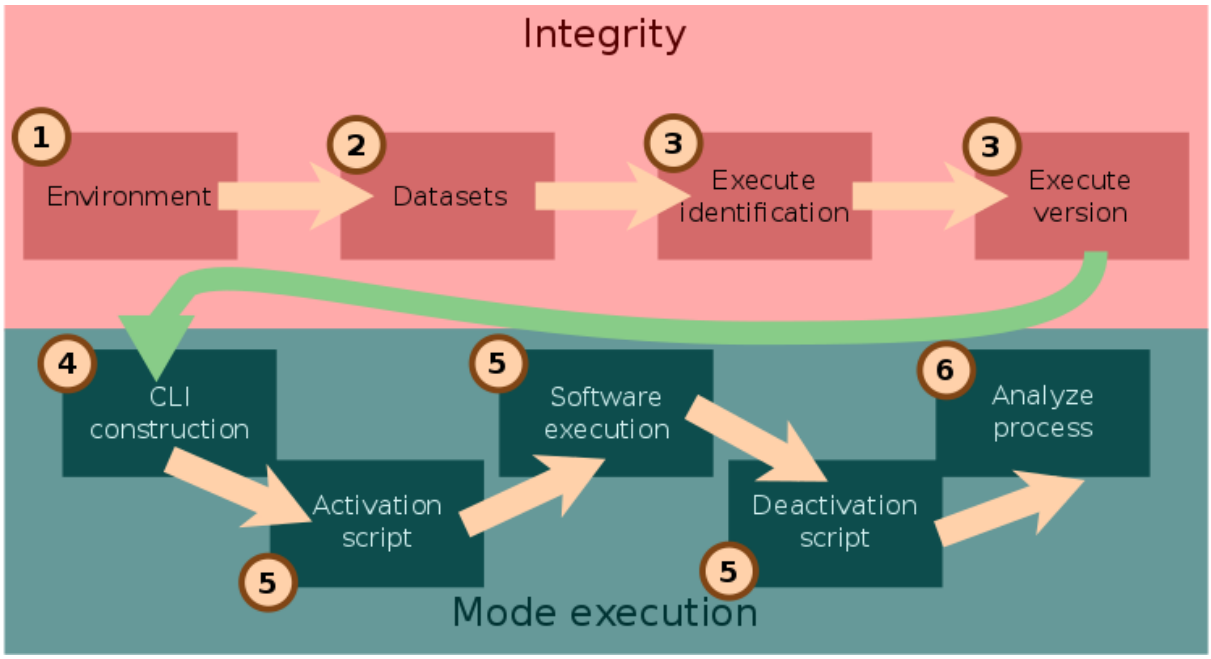

Figure 1. Schematic representation of the integration of S/W inside the ROC-SGSE pipeline.

Figure 1 shows the integration of RCS inside the ROC-SGSE pipeline. There are two phases: registration and execution. The registration phase is described in the next sections.

There are six main steps for the S/W execution:

1. Check of the integrity of the environment defined by the S/W. Validity of the executable, activation/deactivation script, etc., are checked at this moment to be able to use the S/W for following steps.
2. Check that required datasets have correctly been registered inside the ROC database.
3. The identification command of the S/W is performed to check the consistency between the S/W registered version and the one called by the ROC-SGSE pipeline. The same is done for the S/W version.
4. Construction of the CLI for calling the S/W with the appropriate inputs as required by the S/W and registered by the ROC-SGSE pipeline.
5. Activation script *sourcing*, S/W execution with the CLI and deactivation script *sourcing*. Specific environment configuration of the S/W can be done through the activation and deactivation script. The latter allows to let the environment in a cleaned state, or make some cache clean up for example.
6. At the end of the execution, the output of the S/W is analyzed to check that the data has been produced correctly.

If an error is encountered in any of these steps, the ROC-SGSE pipeline raises an exception and the error is stored into the ROC database for future inspections.

Details on each step are given in the following sections.

4.2 Software delivery and installation procedures

The installation and integration of the RCS inside the ROC-SGSE pipeline is detailed.

4.2.1 Programming environment

The scripts required for setting the S/W environment shall be written using BASH. It allows to have uniformity in the language used, and also to provide the most of functionalities.

An executable with a given CLI shall be provided, following strictly the conventions described in the section 3.

4.2.2 Software delivery procedures

S/W provided by the RCS teams shall be downloadable by the ROC team from the dedicated directory in the ROC SVN repository.

In practice each team has its own directory on the ROC repository, where it can store its S/W releases. For the ROC-SGSE, the teams' directories can be found in:

https://version-lesia.obspm.fr/repos/ROC/GroundTests/ROC-SGSE/Software/RCS

The procedures concerning the S/W delivery shall comply the conventions defined in [RD6]. Especially, every new release of a S/W shall be tagged in a dedicated /tags sub-directory.

For instance, if the THR team wants to upload a new release V1.2.0 of its "THR_Calbar" S/W in the /tags folder, the path to the corresponding directory on the repository will be:

https://version-lesia.obspm.fr/repos/ROC/GroundTests/ROC-SGSE/Software/RCS/THR/Calbar/tags/1.2.0

4.2.3 Software installation procedures

Once a new RCS release is tagged on the ROC repository. The ROC team will download a copy of the S/W directory on the ROC servers for testing.

The only task done by the ROC team on the ROC servers shall be to create a symbolic link in a specific location of the ROC pipeline directory, pointing to the latest version of the RCS in the local copy of the SVN repository. Especially, the ROC team will not perform any compilation or installation process, and thus the teams shall ensure that their S/W are ready to be used on the ROC servers before delivery. In other words, it means that the S/W directory shall contain all of the files required to run the program in the ROC servers, including the executable files, when the ROC team will deploy it.

If it is necessary to compile the program, it is the responsibility of the teams to store also on the repository all of the libraries required for compilation as well as binaries that can be run of the ROC servers.

4.3 Software descriptor file specifications

The S/W descriptor file is key element of the interface. It contains information about the RCS that helps the pipeline 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 ROC-SGSE pipeline requires to read the S/W descriptor files in order to identify, to call and to monitor any RCS.

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

4.3.1 General conventions

The RCS teams are in charge of the descriptor file edition for their own S/W. They shall follow the convention below:

- The descriptor file shall be written in the JSON format [RD4]
- It shall be placed in the S/W root directory.
- It shall be named "***roc_sw_descriptor.json***".
- There must be only one descriptor file per S/W and per release.
- All of the paths defined in the S/W descriptor file shall be relative to the S/W root directory.

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

4.3.2 Identification

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

- `"project"`**:** name of the project. It shall be "ROC-SGSE" for S/W used in the ROC-SGSE pipeline, and "ROC" otherwise.
- `"name"`**:** a cool name for the calibration software, to be human readable.
- `"identifier"`**:** a unique name used as reference by the ROC-SGSE pipeline to identify the S/W. It shall contain Latin alphabet uppercase letters only. The hyphen character shall be used a separator if required (e.g., "THR-CALBAR"). In all case, the ROC team shall validate the identifier (ID) to avoid duplicated names.
- `"description"`**:** Short description of the software. This description will be saved in the ROC database.

4.3.3 Release

The `"release"` object shall inform about the current S/W release. It is also used to describe the output data (see Section 4.3.5.2). It shall contain the following attributes:

- `"version"`**:** Current version of the S/W in the format 'MAJOR.MINOR.REVISION', following the ROC conventions [AD2].
- `"date"`**:** Date and hour 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)
- `"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:

- `"file"`: file name to reference a data schema, master CDF, as an indication for the ROC team. Only required in the outputs modes object descriptions.

- `"reference"`: name of a file as reference for the documentation, used as an indication for the ROC team.
- `"url"`: indication for an online resource.

4.3.4 Environment

The attributes defined in the object `"environment"` are used for setting the 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, using the CLI. (It corresponds to the **`EXECUTABLE`** in the calling sequence example given in the section 3.2.3.)

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. If this attribute is provided, the pipeline will pass the value to the **`--config`** common input parameter.
- `"activation"`: Relative path to the directory where the scripts setting up the environment or doing some things needed by the software are done. These scripts shall require any input argument to be run.
- `"deactivation"`: Relative path to the script used to reset the environment to its initial state. These scripts shall require any input argument to be run.

If the command to launch the software is in several parts (for example, IDL is used and it works with `idl program.sav`), the integrity test will fail because the interface doesn't tolerate this. In such situation, the program MUST be wrapped into a BASH script for example, which will be executed by the ROC-SGSE pipeline. The script must be compliant with the interface of the ROC- SGSE pipeline too.

4.3.5 Modes

The `"modes"` object 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 datasets to be read/saved shall be supplied. It allows the pipeline to build the S/W function input calls automatically, and to control 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. This name will be used by the pipeline as first argument in the CLI, as described in the section 3.2.1.
- `"purpose"`: A short description of the purpose of the function
- `"inputs"`: A JSON object containing the list of the specific input parameters to be provided to the CLI when the current S/W mode is called.
- `"outputs"`: A JSON object containing the list of the outputs that are returned by the current S/W function in the *stdout* message in JSON format (see Section 3.3).

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

4.3.5.1 Inputs

The pipeline requires information in order to identify the specific input parameters of a given S/W function, and to build the corresponding CLI calling sequence. This is the aim of the

"inputs" object, which provides the list of the specific input parameters in JSON objects, with the two following mandatory attributes:

- "identifier": The dataset ID associated to the input data file, as referenced in the ROC system.
- "version": The version of the input data file. This allows the ROC pipeline to ensure that S/W will process the right version of the input file.

Note that the ROC pipeline does not require to know the path to the input data files. The two attributes above are sufficient to retrieve the corresponding input data file path in its database, and to pass it to the CLI.

The teams are in charge to define the list of inputs for their S/W in the descriptor file.

This work must 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 input datasets read by the RCS. In practice, this list will be built from the list of outputs produced by the ROC (e.g., L1) and other teams (L2r/L2s).

Since the version of given input file can change, it permits to the teams to take account of this modification in their S/W, and to update the descriptor file in consequence.

Following the example of CLI calling sequence in the section 3.2.3, the "inputs" array shall be structured as followed:

```
"inputs":[
"input_1":{
                    "identifier":"ROC-SGSE_DATASET_ID1”,
                    "version":"01"
                },
"input_2":{
                    "identifier":"ROC-SGSE_DATASET_ID2",
                    "version":"02"
                }
          ],
```

, where "input_1" and "input_2" objects corresponds respectively to the specific input parameter keywords --input_1 and --input_2, and where "ROC-SGSE_DATASET_ID1”/"ROC-SGSE_DATASET_ID2” and "1.0.0”/"2.0.0” are the corresponding dataset IDs and versions that permits to the ROC pipeline to retrieve the input data file paths /path/to/input1 and /path/to/input2.

Note that the CDF data versioning convention is “XX”, where XX is a 2-digits integer (e.g., “01”).

4.3.5.2 Outputs

The pipeline requires information in order to verify that the expected output data files have been correctly produced at the end of a given S/W function execution.

To achieve this goal, the pipeline will rely on the metadata supplied on both the *stdout* JSON message and the "outputs" array. The latter contains a list of JSON objects corresponding to the list of outputs referenced in the *stdout*, especially the name of the objects shall be the same than the name of the attributes in the *stdout*.

Each JSON object listed in `"outputs"` shall described in details the corresponding dataset, using the following attributes/objects:

- `"identifier"`: The dataset ID associated to the output, as referenced in the ROC system. It shall be unique and comply the naming convention listed in [AD2].
- `"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. Allowed values are "LZ", "L0", "L1", "L2", "L2R", "L2S", "L3", "L4", "AUX", "LL0", "LL1", "HK".
- `"release"`: information about the release of the dataset. The structure is the same as defined in the section 4.3.3. In the case of a dataset using the CDF format, the `"file"` attribute shall provide the name of the master CDF file used to generate the output data files.

The teams have the responsibility to define the list of the datasets produced by their S/W. The dataset ID allocation should be done in collaboration with the ROC in order to avoid ID naming duplication.

In any case, the ROC pipeline will perform an automated validation of the S/W descriptor file at each new release, in order to check that the file content is consistent with the ROC database information. In particular, it will verify that the datasets declared the `"modes"` object are all defined in the descriptor and across S/W.

Since the outputs of given S/W can be the inputs of another one, the team in charge shall 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.teams@sympa.obspm.fr* mailing list. This modification shall be done in the framework of a new S/W release. The procedures to be applied in case of S/W updates are given in [RD6].

The inter-dependencies between the RCS I/O are illustrated on the figure below. In this example, the L2r datasets produced by the RCS #1 are the input datasets of the RCS #2. The L2r datasets are fully described in the `"outputs"` object of the RCS #1 descriptor, whereas the RCS #2 descriptor file lists the same L2r datasets in the `"inputs"` object.

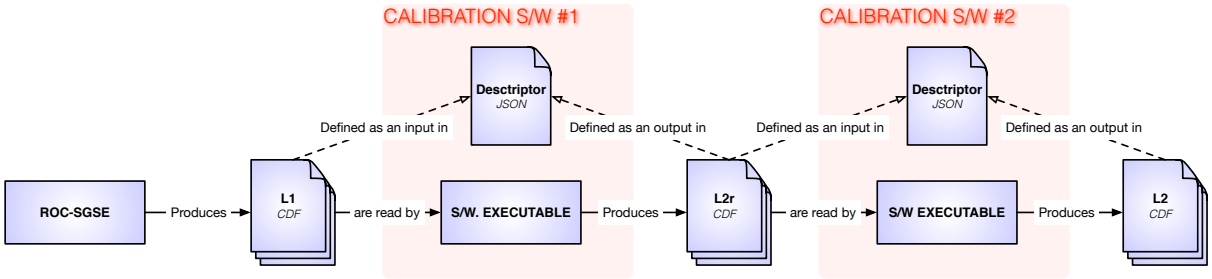

Figure 2. RCS I/O data inter-dependencies.

4.4 Environment setup

The environment setup is only necessary for S/W requiring specific environment variables to work. The ROC-SGSE pipeline will set the environment automatically before starting the related pipeline jobs.

A script, defined in the `"environment.activation"` attribute of the descriptor file, will be launched by the pipeline, without any argument. The script will be "sourced" before starting the S/W, and the stdout of the program redirected to */dev/null*.

For example in BASH:

```
$ source activate.sh 1>/dev/null
```

,where `activate.sh` is here the script path as read in `"environment.activation"`.

If the S/W ends without error, the corresponding deactivation scripts will be launched (if present), sourcing the script defined in the `"environment.deactivation"`field of the descriptor file.

It must be noticed that the ROC pipeline will raise an error if the files defined in the `"environment"` object are not found or not executable.

If scripts edit the value of already existing and/or system environment variables, such as $LD_LIBRARY_PATH or $PATH, the teams in charge shall ensure that they do not overwrite the variable values, since they can be used by the pipeline or other S/W.

To avoid overwriting in BASH enter:

```
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/one/more/path"
```

4.5 Software automated validation

The validation of a RCS by the ROC-SGSE pipeline is done in two steps:

- At the registration: the consistency between the required input datasets and the existence of their definition in the ROC database will be done. If the pipeline doesn't find the definition in the ROC database of an input dataset, or an inconsistency in the inter-dependency between the registered RCS is detected, the execution will be aborted.
- At the execution: the software will be called with the `--identification` keyword before running a mode, in order to check that the information returned by the executable is as expected (i.e, as the information read from the roc_sw_descriptor.json and stored in the ROC-SGSE database). In the case of a mismatch, the execution is stopped, otherwise the S/W is executed as explained in the following section.

4.6 Software automated execution

Figure 3 illustrates the execution schema of two functions "calibration_X" and "calibration_Y" of a given RCS by the pipeline. These two modes are respectively in charge of producing "Output X-a/b" and "Output Y-a/b" data files from the input data files "Input X-a/b" and "Input Y-a/b".

The pipeline's choice to call these functions is made as a first step during the "Calibration mode selection" using the information provided inside the `"modes"` object of the S/W descriptor file. Especially it will allow the pipeline to build the list of the inputs to be passed to the S/W, from the information saved in the ROC database.

The path to the directory where the S/W data products will be saved, the path of the log files or the configuration file to use are provided by the ROC pipeline with the corresponding flags, respectively **`--output`**, **`--log`** and **`--config`**.

At the end of the execution, the S/W will return the *stdout* message in JSON format. This message will be read by the pipeline to get the list of outputs. If the outputs are correctly

validated by the pipeline, then the latter will register information about the outputs, including the path, in the ROC database.

If an exception occurs during the execution, the S/W shall be transmitted all the relevant information through the *stderr*.

Teams shall be aware that the pipeline will not call the S/W at all in the case where, at least one of the input files defined in the `"inputs"` of the descriptor is not found. It should be taken into account in the S/W architecture design, especially in the number of available modes.

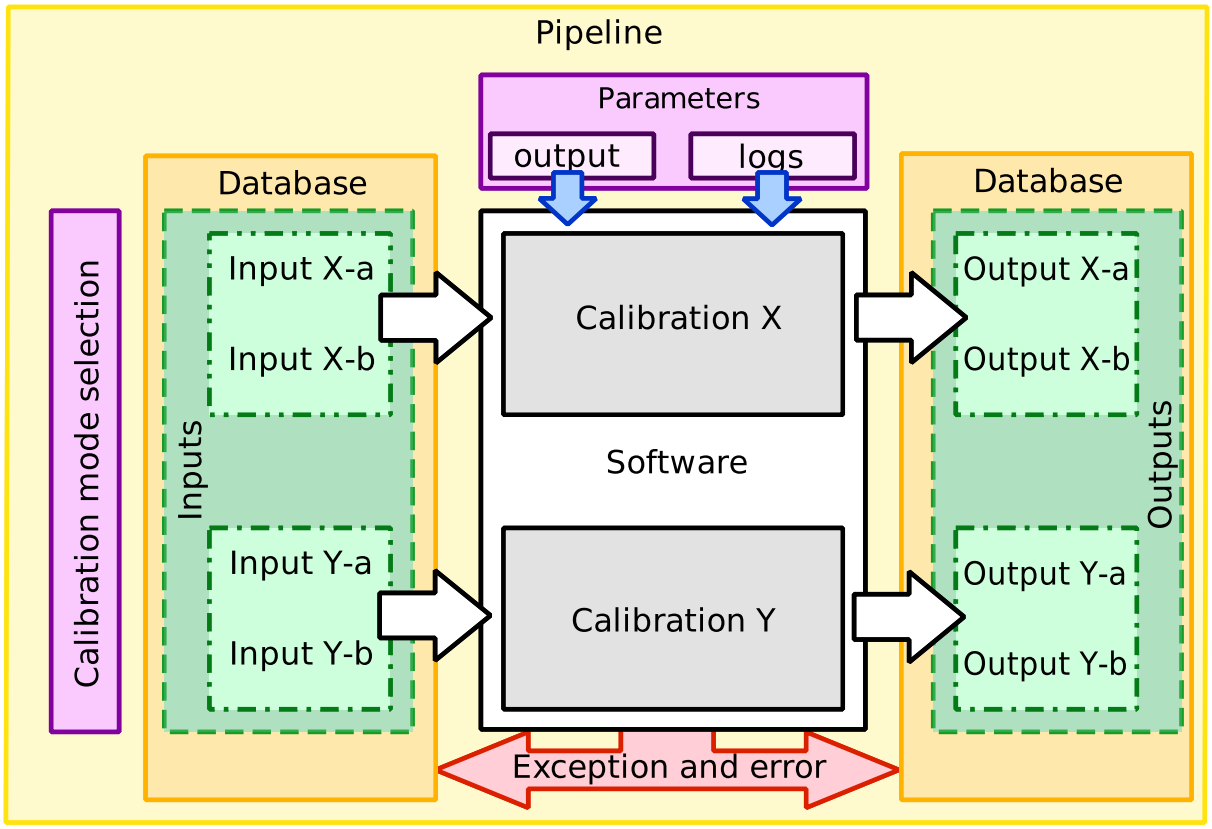

Figure 3. Detail of the execution of a RCS.

4.7 Tests

The ROC team shall be able to easily check the RCS performances just after S/W deployment on the ROC server. To achieve this goal, each S/W shall be delivered with a set of typical input data files as well as their corresponding expected outputs.

To perform the verification, it shall be possible to run S/W in a dedicated test mode in order to compare the output files produced locally with the set of expected outputs.

4.8 Maintenance

The Calibration Wrapper (CaWa) dedicated module of the ROC pipeline must provide commands for quasi-automated launches of a specific S/W, a specific mode of the S/W with specific input files. Used to run the S/W on a specific file with specific conditions to reproduce an isolated problem.

A typical use case for the command is:

```
$pop cawa run CAL-SOFT calibration1 --output /output/directory/path/ \
    --log /path/to/log --inputs --input-1 /path/to/input1 --input_2
/path/to/input2
```

where **pop** is the executable of the ROC-SGSE pipeline.

If the S/W is already registered into the ROC-SGSE pipeline with the name `CAL-SOFT`, the latter will run the calibration mode called `calibration1` of the S/W only if defined in the descriptor file, passing argument for output and log paths as if it was the ROC pipeline while running, and all inputs and other flags following the `--inputs` flag are transmitted directly to the software executable.

If the S/W is not yet registered and you need to test its integration with the ROC pipeline, you can do the following command:

```
$ pop cawa test /path/to/sw/root/directory calibration1 --output
/output/directory/path/ \
    --log /path/to/log --inputs --input-1 /path/to/input1 --input_2
/path/to/input2
```

, which will do exactly the same thing as the previous command but using only the S/W at the root directory provided as first argument of the `test` sub-command.

Inputs arguments that must be given to the S/W executable must follow the `--inputs` flag, which must be placed after all other arguments and flags.

The `--inputs` flag is specific to the CaWa CLI, and must be only used by the pipeline in its CLI. The RCS CLI is not expected to include this flag.

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-SGSE",
        "name": "THR-CALBAR",
        "identifier": "ROC-SGSE-THR-CALBAR",
        "description": "The RPW TNR-HFR CALibration softwARe (THR-CALBAR)
produces RPW TNR-HFR calibrated science data"
    },
    "release": {
        "version": "1.1.0",
        "date": "2015-07-29",
        "author": "Antonio Vecchio",
        "contact": "antonio.vecchio@obspm.fr",
        "institute": "LESIA",
        "modification": "Update tnr_l2r_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_l2r_cal",
            "purpose": "Produce calibrated science TNR data file at analys
er level (L2r)",
            "inputs": {
                "input_l1_tnr": {
                    "identifier": "ROC-SGSE_TNR_L1-SURV",
                    "version": "01"
                }
            },
            "outputs": {
                "output_l2r_tnr": {
                    "identifier": "ROC-SGSE_TNR_L2R-SURV",
                    "name": "ROC-SGSE TNR L2R survey data",
                    "description": "RPW TNR L2R science data in survey for
ROC-SGSE",
                    "level": "L2R",
                    "release": {
                        "author": "Xavier Bonnin",
                        "date": "2015-11-13",
                        "version": "02",
                        "contact": "xavier.bonnin@obspm.fr",
                        "file": "ROC-SGSE_TNR_L2R-SURV_V02.cdf",
                        "institute": "LESIA",
                        "reference": "RPW_THR _Data_products.pdf",
                        "url": "https://where.is.the.doc/info",
                        "modification": "Updated structure"
                    }
                }
            }
        },
        {
            "name": "hfr_l2r_calibration",
            "purpose": "Produce calibrated science HFR data file at analys
er level (L2R)",
            "inputs": {
                "input_l1_hfr": {
                    "identifier": "ROC-SGSE_HFR_L1-SURV",
                    "version": "01"
                }
            },
            "outputs": {
                "output_l2r_hfr": {
                    "identifier": "ROC-SGSE_HFR_L2R-SURV",
                    "name": "ROC-SGSE HFR L2R survey data ",
                    "description": "RPW TNR L2R science data in survey for
ROC-SGSE",
```

```
                    "level": "L2R",
                    "release": {
                        "author": "Xavier Bonnin",
                        "date": "2015-11-13",
                        "version": "01"
                    }
                }
            }
        }
    ]
}
```

5.2 Examples of a CLI calling sequence

With the following configuration, the pipeline will call the S/W for the calibration mode `tnr_l2r_cal` as (assuming that the environment as 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_l2r_cal --input_l1_tnr /path/to/input_cdf \
    --output /path/to/output/directory --log /path/to/log/directory \
    --config $SW_ROOT/config/configuration.file
```

If there are no errors, the S/W should return the following JSON in the stdout:

```
{
    "output_l2r_tnr":"output_file_name.cdf"
}
```

Then the ROC pipeline will search for a file at `/path/to/output/directory/output_file_name.cdf`.

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 ROC pipeline, it will store not necessarily useful informations.

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-SGSE",
    "name":"CALBAR",
    "identifier":"RPW-TEST-CALBAR"
}
```

, 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 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+)$"
                },
                "file": {"type": "string"},
                "institute": {"type": "string"},
                "reference": {"type": "string"},
                "url": {
                    "type": "string",
                    "format": "uri"
                },
                "contact": {
                    "type": "string",
                    "format": "email"
                },
                "modification": {"type": "string"}
            },
            "required": ["author", "date", "version"],
            "additionalProperties": false
        },
        "output_dataset": {
            "type": "object",
            "properties": {
                "identifier": {
                    "type": "string",
                    "pattern": "^[\\w-]+$"
                },
                "name": {"type": "string"},
                "level": {"enum": ["L0", "L1", "L2R", "L2S", "L2", "L3",
"AUX", "LL0", "LL1", "HK"]},
                "description": {"type": "string"},
                "release": {"$ref": "#definitions/release"}
            },
            "required": ["identifier", "name", "level", "description",
"release"],
            "additionalProperties": false
        },
        "input_dataset": {
            "type": "object",
            "properties": {
                "identifier": {
                    "type": "string",
                    "pattern": "^[\\w-]+$"
                },
                "version": {
                    "type": "string",
                    "pattern": "^(\\d+\\.)?(\\d+\\.)?(\\d+)$"
                }
            },
            "required": ["identifier", "version"],
            "additionalProperties": false
        }
    },
    "type": "object",
```

```
    "properties": {
        "identification": {
            "type": "object",
            "properties": {
                "project": {"type": "string"},
                "name": {"type": "string"},
                "identifier": {"type": "string"},
                "description": {"type": "string"}
            },
            "required": ["project", "name", "identifier"],
            "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-]Cols
		Science-Cols

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