BIOMASS Processing Suite Software User Manual

SOFTWARE USER MANUAL

biomass

Ref: BIO-BPS-SUM-ARE-010479
Issue: 4.2.0
Date: 28/11/2025

Document information

Title	**BPS Software User Manual**
Internal Ref.	
External Ref.	BIO-BPS-SUM-ARE-010479
Date	28/11/2025
Issue	4.2.0
Pages	109

Recipients	
Cristiano Lopes	ESA – ESRIN
Michele Caccia	ESA – ESRIN
Muriel Pinheiro	ESA – ESRIN

Prepared by	
Riccardo Piantanida	
Davide D'Aria	
Paolo Mazzucchelli	

Checked by	
Davide D'Aria	

Approved by	
Davide D'Aria	

Change history

Date	Issue	Change	Author
29/06/2022	1.0	First issue of the document	Riccardo Piantanida
30/09/2022	1.0.1	Post-FAT0 issue of the document: • Added BPS processors simplified execution syntax (Sec. 5.3) • Updated Docker images execution syntax (Sec. 5.3)	Riccardo Piantanida
08/02/2023	2.0.0	FAT1 issue of the document • Added information about L1 Framing Processor (ALL) • BPSFAT0-17, 18, 26, 29: Added details about processors configuration files (Sec. 6) • BPSFAT0-17, 23, 29, 30: Software package description updated and aligned to BPS SDD document (Sec. 3) • BPSFAT0-24: Minimum hardware requirements added (Sec. 4.1) • BPSFAT0-17, 25, 29: External dependencies detailed (Sec. 4.4.1); NOTE: usage of pip together with conda still not amended • BPSFAT0-17, 27, 29: Procedures for installation and usage of Docker images highlighted (Sec. 4, 5); Added appendix with Docker files (Sec. 7) • BPSFAT0-17, 28, 29, 30: Dependency section reorganized and improved, adding details about internal resources installation (Sec. 4.4.2)	Riccardo Piantanida
14/04/2023	2.1.0	Post-FAT1 issue of the document NOTE: This version of the document describes installation and usage procedures applicable starting from BPS V2.1. Due to breaking changes introduced, procedures described are no more applicable to previous BPS versions • BPS-149, BPS-235, BPSFAT1-78: Added details on logging strategy (Sec. 5.3.4) • BPS-226, BPSFAT0-25: Installation procedure updated to use only conda (Sec. 4.3.1) • BPS-228, BPS-339, BPSFAT0-30, BPSFAT1-82: Improve toml files description (Sec. 5.3.1, 6) • BPS-229, BPSFAT1-81: Removed run command in BPS calls (Sec. 4.5.1) • BPS-243: Added details about stop-and-resume functionality (Sec. 5.3.5) • BPS-261: Added details on steps requiring internet connection (ALL) • BPS-263: Added details on management of different processors versions (Sec. 4.3.1, 5.3.1) • BPS-329: Clarified procedure for installation of DEM (Sec. 4.4.2.1) • BPS-341: Added details on installation pre-requirements (Sec. 4.2) • BPSFAT0-25: Added note on Docker images YAML files (Sec. 7)	Riccardo Piantanida

Date	Issue	Change	Author
		• BPSFAT0-27, BPSFAT0-28: Added details on Docker images layout (Sec. 4.3.2) and usage (Sec. 5.3.2) • BPSFAT1-84: Added sample execution details (Sec. 5.3.3)	
31/07/2023	2.1.2	Post-FAT2 issue of the document • BPS-243, BPS-244, BPSFAT2-16: Stop-and-resume functionality description updated (Sec. 5.3.7) • BPS-263: Added details on management of different processors versions (Sec. 4.3.1, 5.3.1) • BPS-329, BPSFAT0-28, BPSFAT0-25: Added details about internal resources and DEM installation (Sec. 3, 4.4.2) • BPS-339: Configuration files (.toml) updated (Sec. 6) • BPS-341: Reviewed list of software dependencies and added versions (Sec. 4.1) • BPS-430: Clarified convention used for software versioning (Sec. 3) • BPSFAT0-17, BPSFAT0-25: Docker files updated (Sec. 7); Build procedures (partially) added (Sec. 8) • BPSFAT0-25: Added details about Docker environment files (Sec. 3, 8.3) • BPSFAT0-27: Added missing paths (Sec. 4.3.2) • BPSFAT0-29, BPSFAT1-78: Added information about exit codes (Sec. 5.3.5); Clarified convention for stdout and stderr usage (Sec. 5.3.4); Installation procedure extended to cover all BPS processors (Sec. 4) • BPSFAT1-80: Added indication of IRI model index files location in BPS bundle package (Sec. 3) • BPSFAT2-2: Added details about installation and usage by different users (Sec. 4.3.1, 5.3.1)	Riccardo Piantanida
06/10/2023	2.1.3	Post-FAT2 issue of the document (associated to BPS V2.1.3) • Clarified usage of /app/wd/tmp by various BPS Processors (ALL) • Removed DEM from STA_P internal resources (Sec. 4.4.2) • Updated sample Docker run command (Sec. 5.3.2) • Updated sample Job Order file (Sec. 5.3.3)	Riccardo Piantanida
19/01/2024	2.2	Delta-FAT2 issue of the document (associated to BPS V2.2.0) • BPS-262, BPS-417, BPSFAT0-17: Build procedures updated and completed (Appendix C) • BPSFAT1-82: Added further clarifications on configuration files and working directory usage (ALL) • BPSFAT1-84: Added further clarifications on processors usage (ALL) • BPSFAT2-13: Added clarifications on resources required to run BPS processors when using Docker images (Sec. 4.3.2, 5.3.2)	Riccardo Piantanida
17/05/2024	2.2.2	Post Delta-FAT2 issue of the document (associated to BPS V2.2.2) • Overall document review to align it to latest software updates (ALL) • BPS-537: Management of intermediate products reviewed and updated (ALL)	Riccardo Piantanida

Date	Issue	Change	Author
		• BPSFAT0-28: Description of Copernicus DEM index file reviewed and updated (Sec. 4.4.2.1) • BPSDFAT2-25: Uninstallation procedure updated (Sec. 4.5)	
06/09/2024	3.0.0	FAT3 issue of the document (associated to BPS V3.0.0)	Riccardo Piantanida
13/12/2024	3.1.0	Post-FAT3 issue of the document (associated to BPS V3.1.0) • Added docker package content description (Sec. 3) • Added notes on environment reproducibility and software update procedure (Sec. 4.3.1, 7.4) • Updated docker image layout (Sec. 4.3.2) • Updated list of external dependencies (Sec. 4.4.1) • IRI model updated to IRI2020; IRI data now a L1 Processor internal resource (Sec. 4.4.2, 4.4.2.3) • Updated DEM installation procedure (Sec. 4.4.2.1) • Section on sample execution results totally re-worked (Sec. 5.3.3) • Updated docker files (Sec. 6) • Updated overall build procedure (Sec. 7.1) • Added details to build external libraries (Sec. 7.3.4)	Riccardo Piantanida
07/03/2025	3.2.0	Post-FAT3 issue of the document (associated to BPS V3.2.0) • Copernicus DEM website link updated (Sec. 1.6) • Added clarification on BPS processors help message displaying (Sec. 4.3.1) • Dependencies versions updated (Sec. 4.4.1, 7.1) • Full review and improvement of Processors Configuration section (Sec. 5.3.3) • Troubleshooting section filled (Sec. 5.4)	Riccardo Piantanida
30/04/2025	3.3.0	Post-FAT3 issue of the document (associated to BPS V3.3.0) • Updated dependencies versions (Sec. 4.4.1) • Added details on usage via docker images (Sec. 5.3.2) • Added details on input auxiliary files content and usage (Sec. 5.3.3, 5.3.6) • Added clarification on RAMDISK usage for L1 Processor (Sec. 5.3.3.1) • Improved troubleshooting notes about stop and resume functionality (Sec. 5.4) • Conda packages extension updated (Sec. 7.1) • Added appendix with detailed description of processors inputs selection criteria (Sec. 9)	Riccardo Piantanida Matteo Aletti
08/06/2025	3.3.1	Post-FAT3 issue of the document (associated to BPS L3_P delivery) • Added details on L3_P processor	Nathan Paillou
23/07/2025	4.0.0	Post-launch issue of the document (associated to BPS V4.0.0) • Updated dependencies versions (mainly python, numpy and opencv) (Sec. 4.4.1)	Riccardo Piantanida Matteo

Date	Issue	Change	Author
		• Updated description of STA_P working directory	Aletti Federico Boniardi
28/11/2025	4.2.0	Post-launch issue of the document (associated to BPS V4.2.0) • Updated command to initialize local conda channels (Sec. 4.3.1) • Updated list of external dependencies (Sec. 4.4.1)	Riccardo Piantanida Matteo Aletti

Table of contents

1. Introduction

1.1. Context

BIOMASS is ESA's 7th Earth Explorer mission. The purpose of the BIOMASS is to reduce the uncertainty in the worldwide spatial distribution and dynamics of forest biomass in order to improve current assessments and future projections of the global carbon cycle. This objective will be achieved by the implementation of a P-band SAR mission, providing global maps of forest biomass stocks, forest disturbance and growth.
The BIOMASS mission will also provide 3D views of forests through the "tomographic" phase and will also support additional science needs arising from the opportunity to explore the Earth for the first time with a P-band SAR system from space offering the possibility to "see" below the vegetated areas and in beneath bare soil or icy regions.

To achieve its challenging objectives, the BIOMASS mission requires a specific data processing strategy that combines different technologies at different processing level in order to generate the final Above Ground Biomass global map.

1.2. Scope of the document

The purpose of this document is to provide the "Software User Manual" for the BIOMASS Processing Suite (BPS) project.
This document is organized as follows:

- Section 2: BPS system overview;
- Section 3: BPS software package description;
- Section 4: Installation manual;
- Section 5: User manual;
- Section 6: Appendix A: Docker files;
- Section 7: Appendix B: Build procedure.

1.3. Acronyms

Acronym	Description
AD	Applicable Document
AGB	Above Ground Biomass
ARESYS	Advanced Remote Sensing and Systems
ATP	Acceptance Test Plan
BCT	Biomass Calibration Transponder

Acronym	Description
BEES	BIOMASS End-to-End Simulator
BEEPS	BIOMASS End-to-End Performance Simulator
BEEPS-FE	BEEPS Front End
BPS	BIOMASS Processing Suite
CBEEPS	Complete BEEPS
DEM	Digital Elevation Model
DGM	Detected Ground-range Multi-looked
ESA	European Space Agency
FA	Forest Assembly
FAT	Factory Acceptance Test
FD	Forest Disturbance
FH	Forest Height
INT	Interferometric Phase
IRF	Impulse Response Function
L0	Level-0 product
L0M	Level-0 Monitoring product
L0S	Level-0 Standard product
L1a	Level-1a product
L1b	Level-1b product
L1c	Level-1c product
L1M	Level-1 Monitoring product
L1S	Level-1 Standard product
L2a	Level-2a product
L2b	Level-2b product
L3	Level-3 product
PARC	Polarimetric Active Radar Calibrator
PF	Processing Facility
RD	Reference Document
RXO	RX-Only product
SAR	Synthetic Aperture Radar
STA	Coregistered Stack
TBD	To Be Defined
TBC	To Be Confirmed

Acronym	Description
TDS	Test Data-Set
TOM	Tomographic Phase
VTP	Validation Test Plan

1.4. Symbols and Variables

Acronym	Description
/	/

1.5. Applicable documents

[AD1] [BPS_ICD] BPS Interface Control Document, I/R 3/2
[AD2] ESA-EOPG-EEGS-ID-0092, BIOMASS CPF-Processor ICD, I/R 2/3
[AD3] ESA-EOPG-EEGS-ID-0083, Generic Processor ICD, I/R 1/4

1.6. Reference documents

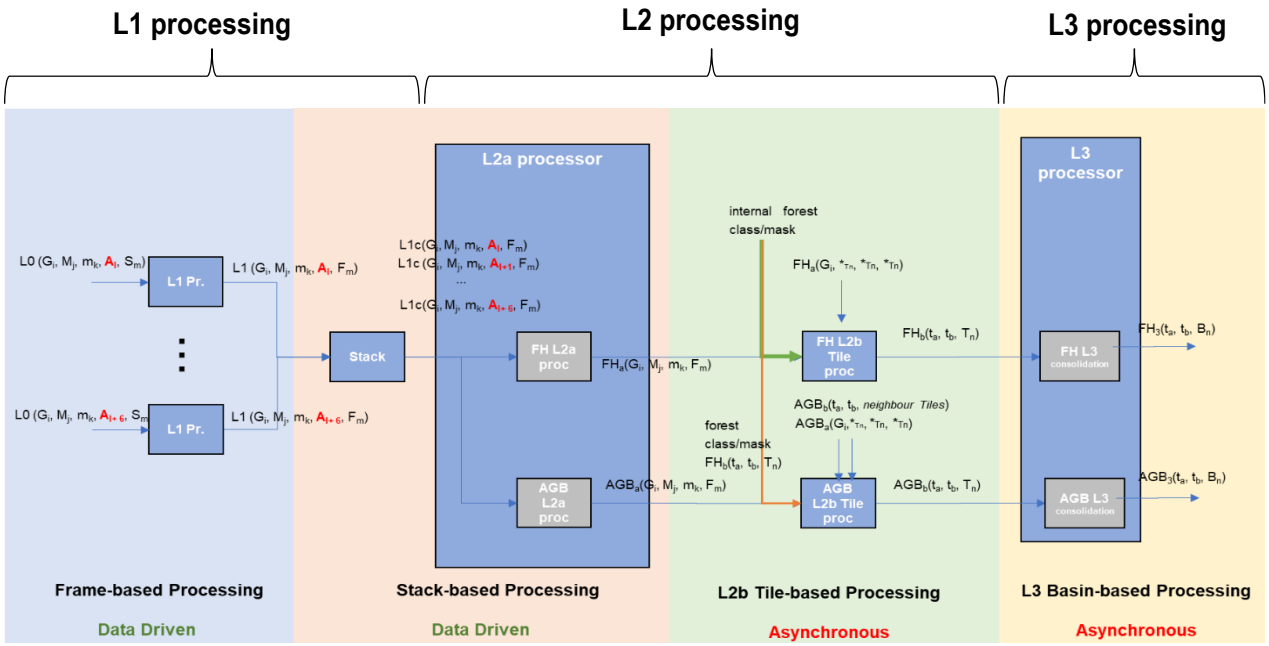
[RD1] [BPS_SDD] BPS Software Design Document, I/R 3/4
[RD2] [BPS_TN_BENCH] BPS Technical Note on Benchmarking, I/R 1/6/2
[RD3] [BPS_AUX_FMT] BPS Auxiliary Products Format, I/R 3/4
[RD4] [BPS_IODD] BPS I/O Data Definition Document, I/R 3/1
[RD5] Copernicus DEM website, https://dataspace.copernicus.eu/explore-data/data-collections/copernicus-contributing-missions/collections-description/COP-DEM
[RD6] EGM2008 geoid website, https://www.agisoft.com/downloads/geoids
[RD7] SRTM DEM website, https://www.earthdata.nasa.gov/sensors/srtm
[RD8] GeographicLib – Magnetic models https://geographiclib.sourceforge.io/C++/doc/magnetic.html
[RD9] Geomagnetic poles, https://www.ncei.noaa.gov/products/wandering-geomagnetic-poles

2. BPS Overview

The BIOMASS Processing Suite (BPS) is in charge of processing BIOMASS Level-0 data up to Level-3, generating a wide set of products. Product generation scheme is depicted in the following two figures respectively for TOM phase and for INT phase. The processing foresees the following steps:

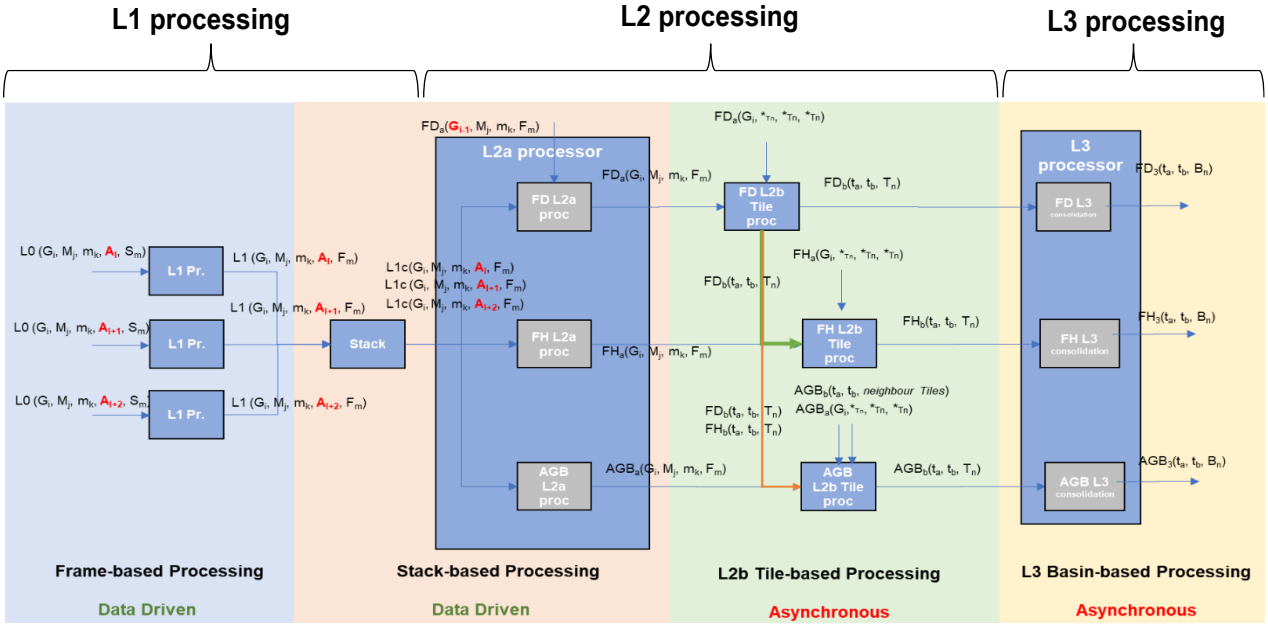
- **L1 Processing**: the BIOMASS image formation step, taking in input 1 L0 slice and generating in output N>=1 L1 frames;
- **Stack Processing**: the coregistration and stacking processing step that will define the required 7-images Tomographic stacks or the 3-images Interferometric stacks. It includes stack phase calibration;
- **L2a Processing**: processing step for each stack that provides the inputs to the further products Tile Generation performed by the L2b processors. For one global cycle the number of stacks varies mainly with latitude, approximately from a minimum of two at the Equator (ascending/descending) to about six at higher latitudes (three ascending/descending pairs);
- **L2b Processing**: processing that aggregates all the stack-based L2a products insisting on the same location on ground (Tile) in the same time interval and generates a single BIOMASS L2b product. A different number of stacks may be available for each DGG tile, based on tile dimensions and swath coverage (which is not related to DGG tiling pattern);
- **L3 Processing**: consolidation processing removing discontinuities in L2b products.

An additional processing tool is provided as part of BPS, i.e., the **L1 Framing** one, which is in charge of computing the L1 frames start and stop times needed to trigger the L1 processing step.



Biomass high-level processing scheme from L0 data to L3 products, TOM phase.

Fig.1



Biomass high-level processing scheme from L0 data to L3 products, INT phase.

Fig.2

NOTE: Regarding stack images cardinality, their number can vary (not always 3 or 7) due to contingency reasons and it can go up to 8 in TOM phase.
NOTE: Regarding AGB, the nomenclature AGB L2a (AGBa) indicates the processing unit and flow, whereas the product for AGB flow at L2a is the ground cancelled product (GN).

3. Software Package Description

The BPS software is delivered in two different solutions, i.e.:

- as a **compressed archive (bundle)**, including all the packages and tools required for BPS installation and execution
  Naming convention: ***`bps-bundle-vM.m.p.tar.gz`*** *(where M, m and p indicate respectively the major, minor and patch version)*

- as a set of **Docker images** (one for each BPS processor), with the software already pre-installed and ready for usage
  Naming convention: ***`bps-xxx-processor-dockerimage-vM.m.p.tar.gz`*** *(where xxx is the acronym of the processor, e.g., l1, and M, m and p indicate respectively the major, minor and patch version)*

It has to be noted that the just presented convention used to indicate BPS software versions (i.e., `vM.m.p`) can be mapped on the one used for software tags (used, e.g., in Task Tables, Job Orders, …) in the following way:

- `vM.m.p` → `0M.0m` until version V2.1.1
- `vM.m.p` → `0M.mp` starting from version V2.1.2

Depending on the corresponding BPS version and on the processors included in it, the content of the delivery bundle may be different. The following tables summarise respectively the expected content of the BPS delivery bundle and the list of packages and tools for each BPS processor.

BPS Delivery Bundle (one folder for each BPS processor)	
∟ pkgs	Folder including Python conda packages, one for each BPS package. Packages are organized in noarch or linux-64 subfolders.
∟ bin	Folder including C++ binaries (executables) required by BPS processors in order to run
∟ (lib)	Folder including C++ binaries (libraries) required by BPS processors in order to run (Only for L1 Framing, L1 and Stack Processors)
∟ README.md	ReadMe file reporting the procedure to be followed to install BPS bundle (the same one described in this document, in brief)

Tab.1 BPS delivery bundle content (one folder for each BPS processor)

BPS Processor	Packages (Python conda packages)	Tools (C++ binaries)
L1 Processor	bps_l1_processor bps_l1_pre_processor bps_l1_core_processor	BiomassL0ImportPreProc BPSL1CoreProcessor libareapplib_common.so libareapplib_ionosphericCalibration.so libareapplib_rfimitigator.so libare_core.so libare_focus.so libare_parsing.so libare_sarbase.so IonosphericHeightIRI20Estimator
Stack Processor	bps_stack_processor bps_stack_pre_processor bps_stack_coreg_processor bps_stack_cal_processor	BPSStackProcessor libareapplib_common.so libare_core.so libare_parsing.so libare_sarbase.so
L2a Processor	bps_l2a_processor	/
L2b FD Processor	bps_l2b_fd_processor	/
L2b FH Processor	bps_l2b_fh_processor	/
L2b AGB Processor	bps_l2b_agb_processor	/
L3 Processor	bps_l3_processor	/
L1 Framing Processor	bps_l1_framing_processor	libCfiDataHandling.so libCfiEECommon.so libCfiFileHandling.so libCfiLib.so libCfiOrbit.so libCfiPointing.so libgeotiff.so.2 libproj.so libproj.so.14 libproj.so.14.0.2 libtiff.so libtiff.so.5 libxml2.so

		libxml2.so.2
Common (shared by all the processors)	bps_common bps_transcoder arepytools arepyextras_runner	/
Auxiliary	arepyextras-copernicus_dem_extractor	/

NOTE: BTK is not part of this list as provided as a separate bundle. Instructions on how to install and use it are in the scope of a dedicated document.

Tab.2 BPS list of packages and tools

For what concern the Docker images, they are provided together with:

- the associated Docker file (see Sec. 6)
- the associated environment file (.yml) (see Sec. 7.4 and Sec. 4.3.1)

BPS docker package	
∟ prereq	bps-prereq-dockerfile-vX.Y.Z bps-prereq-vX.Y.Z.yml
∟ l1_framing_processor	bps-l1-framing-processor-dockerfile-vX.Y.Z bps-l1-framing-processor-dockerimage-vX.Y.Z.tar.xz bps-l1-framing-processor-environment-vX.Y.Z.yml
∟ l1_processor	bps-l1-processor-dockerfile-vX.Y.Z bps-l1-processor-dockerimage-vX.Y.Z.tar.xz bps-l1-processor-environment-vX.Y.Z.yml
∟ l2a_processor	bps-l2a-processor-dockerfile-vX.Y.Z bps-l2a-processor-dockerimage-vX.Y.Z.tar.xz bps-l2a-processor-environment-vX.Y.Z.yml
∟ l2b_agb_processor	bps-l2a-agb-processor-dockerfile-vX.Y.Z bps-l2a-agb-processor-dockerimage-vX.Y.Z.tar.xz bps-l2a-agb-processor-environment-vX.Y.Z.yml
∟ l2b_fd_processor	bps-l2b-fd-processor-dockerfile-vX.Y.Z bps-l2b-fd-processor-dockerimage-vX.Y.Z.tar.xz bps-l2b-fd-processor-environment-vX.Y.Z.yml
∟ l2b_fh_processor	bps-l2b-fh-processor-dockerfile-vX.Y.Z bps-l2b-fh-processor-dockerimage-vX.Y.Z.tar.xz bps-l2b-fh-processor-environment-vX.Y.Z.yml

∟ l3_processor	bps-l3-processor-dockerfile-vX.Y.Z bps-l3-processor-dockerimage-vX.Y.Z.tar.xz bps-l3-processor-environment-vX.Y.Z.yml
∟ stack_processor	bps-stack-processor-dockerfile-vX.Y.Z bps-stack-processor-dockerimage-vX.Y.Z.tar.xz bps- stack-processor-environment-vX.Y.Z.yml

Tab.3 BPS docker package content

In addition to this, BPS software is provided together with its internal resources (GMF, IRI, FNF, LCM and CAL_AB), provided each one via a dedicated compressed archive, except for DEM one (see Sec. 4.4.2).

NOTE: LCM and CAL_AB will be provided starting from the first BPS version including L2b AGB Processor.

This document describes in Sec. 4 the procedure to be followed to install BPS and in Sec. 0 how the BPS processors can be used, providing where needed specific notes for both the delivery solutions.

4. Installation Manual

4.1. Minimum Hardware and Software Requirements

The hardware and software requirements for BPS installation and usage are the following:

Hardware requirements

- CPU: Intel Xeon-class multi-core CPU, 16 physical cores (minimum: 1 core)
- Clock frequency >= 2.4 GHz
- Memory: 64 GB (memory/physical core ratio = 4 GB/core) (minimum: 20 GB)
- Storage*:
  - SSD or NvMe
  - Local storage (for installation): 1 GB
  - Local storage (for output data): 25 GB (peak usage: Stack Processor, TOM phase)
  - Shared memory (for intermediate data): 10 GB (peak usage: L1 Processor, delete-on-consume activated)**

** NOTE: for more details on resources usage by the different BPS processors refer to [RD1] and [RD2].*
*** NOTE: shared memory value is considered on top of the specified RAM.*

Software requirements

- OS (host): Photon (natively supporting Docker technology)
- OS (guest): Linux RedHat 8.0 (or compatible)
  - (on top of OS installation) libgomp library (required for software parallelization through openMP) (version: libgomp.so.1)
  - (on top of OS installation) libgfortran library (required by L1 Processor to execute IonosphericHeightIRI20Estimator tool) (version: libgfortran.so.5)
- Python (see next sections for details about Python environment creation and version used)
- Conda (Miniconda distribution)

4.2. Installation Pre-requisites

The only pre-requirement for the BPS installation is the satisfaction of the hardware and software requirements described in the previous section.

For the installation of Python + Conda (Miniconda), instructions can be found in the official documentation, i.e., https://docs.conda.io/en/latest/miniconda.html.

It has to be noted that, in case of usage of Docker images, this operation is not needed, being the software already pre-installed on the provided image.

4.3. Installation Procedure

4.3.1. Installation using delivery bundle

NOTE: In order to install BPS using delivery bundle, internet connection is required

In order to properly install BPS starting from the delivery bundle, the following steps shall be performed:

1. Extract the compressed archive in a temporary folder:

```
$ tar -xzvf bps-bundle-vM.m.p.tar.gz
```

2. Create and activate a dedicated Python environment (e.g., bps_env):

```
$ conda create --name bps_env python=3.12

$ conda activate bps_env
```

3. Initialize local conda channels and configure them:

```
$ conda install conda-build

$ mkdir -p bps_conda_channel && find bundle -type d -wholename
"*/pkgs/*" -exec cp -r {} bps_conda_channel \; && conda index --
verbose ${PWD}/bps_conda_channel

$ conda config --prepend channels conda-forge && conda config --
prepend channels ${PWD}/bps_conda_channel
```

The three instructions above are respectively aimed at:

- install conda-build, which is needed in order to manage local conda channels;
- initialize local conda channels (one for all the processors, as in the command above, or one per processor, as preferred);
- configure conda channels priorities, in order to ensure that the proper channels are used in the correct order;

4. Install BPS processors starting from Python conda packages included in the bundle:

```
$ conda install bps-l1_processor

$ conda install bps-stack_processor

$ conda install bps-l2a_processor

$ conda install bps-l2b_fd_processor

$ conda install bps-l2b_fh_processor

$ conda install bps-l2b_agb_processor

$ conda install bps-l3_processor

$ conda install bps-l1_framing_processor
```

Environment reproducibility

In order to reproduce exactly the same environment of the Docker images provided, in the BPS docker package dedicated environment files (.yml) are available as described in Tab.3.

The environment files describe the environments that are installed in the delivered docker image and contain two parameters that may need to be updated to the local setup:

- the environment name;
- the local conda channel position.

```
name: env

channels:

       - file:///tmp/python-packages
```

Here the user may change the environment name “env” and needs to change the local conda channel position from `- file:///tmp/python-packages` to `- file:///local/path/to/local_conda_channel`.

Once each environment.yml has been adapted to the local setup, the following command shall be used:

```
$ conda env create -f environment.yml
```

5. Copy C++ binaries (executables and libraries) included in the bundle to desired folders and add them to the path:

```
$ export PATH=${PATH}:/path_to_bin_folder

$ export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/path_to_lib_folder
```

6. Remove the temporary folder where the bundle has been extracted.

An example of complete installation result could be:

- /opt/bps/vXYZ/bin/
  - BiomassL0ImportPreProc
  - BPSL1CoreProcessor
  - IonosphericHeightIRI20Estimator
  - BPSStackProcessor
- /opt/bps/vXYZ/lib/
  - libareapplib_common.so
  - libareapplib_ionosphericCalibration.so
  - libareapplib_rfimitigator.so
  - libare_core.so
  - libare_focus.so
  - libare_parsing.so
  - libare_sarbase.so
  - libCfiDataHandling.so
  - libCfiEECommon.so
  - libCfiFileHandling.so
  - libCfiLib.so
  - libCfiOrbit.so
  - libCfiPointing.so
  - libgeotiff.so.2
  - libproj.so
  - libproj.so.14
  - libproj.so.14.0.2
  - libtiff.so
  - libtiff.so.5
  - libxml2.so
  - libxml2.so.2

In order to verify the successful completion of the installation procedure, run BPS processors without inputs to display the help message. Note that it is possible to display the help of all processors without setting the environment variables.

```
$ bps_l1_processor --help
```

```
$ bps_stack_processor --help

$ bps_l2a_processor --help

$ bps_l2b_fd_processor --help

$ bps_l2b_fh_processor --help

$ bps_l2b_agb_processor --help

$ bps_l3_processor --help

$ bps_l1_framing_processor --help
```

A sample output, obtained with L1 Processor, is here below reported:

```
Usage: bps_l1_processor [OPTIONS] JOB_ORDER_FILE

  BIOMASS Processing Suite - L1 Processor

  Starts processing from Job Order file

Options:
  --working-dir PATH    Working directory (optional)
  --version             Show processor version and exit
  --help                Show this message and exit
```

Note that in case of update of BPS software two options can be followed:

- uninstall the software and then reinstall it from scratch;
- update conda packages using the command `conda update` and then replace the C++ binaries (executables and libraries) with the updated ones.

It has to be noted also that different versions of the same processor can be installed on the same system given that:

- different Python environments are used;
- different configuration files are used;
- different paths are set to point to different C++ binaries (see also Sec. 5.3.1).

On the other hand, in case different users want to use the same installation, the same Python environment can be used, given that it is accessible and activable by all of them, and the same C++ binaries.

4.3.2. Installation using Docker images

In order to properly install BPS starting from the Docker images, only the following command is required:

```
$ docker load -i bps-xxx-processor-dockerimage-vM.m.p.tar.gz
```

The following layout is created inside the Docker images:

- `/opt/conda_folder/envs/env`: location of conda environment
- `/opt/conda_folder/envs/env/bin`: location of BPS processor entry point, e.g:
  - `/opt/conda_folder/envs/env/bin/bps_l1_processor`
- `/usr/local/lib`: location of the EOCFI libraries and of the BPS processors C++ libraries, e.g.:
  - `/usr/local/lib/lib_arecore.so`
- `/usr/local/bin`: location of the BPS processors executables and C++ binaries, e.g.:
  - `/usr/local/bin/bps_l1_processor -> /opt/conda_folder/envs/env/bin/bps_l1_processor`
  - `/usr/local/bin/L1CoreProcessor`
- `/storage`: location of BPS internal resources (see Sec. 5.3.2), e.g.:
  - `/storage/DEM` for DEM
  - `/storage/GMF` for GMF
  - `/storage/IRI` for IRI
  - `/storage/FNF` for FNF
  - `/storage/LCM` for LCM
  - `/storage/CAL_AB` for CAL_AB

4.4. Dependencies

4.4.1. External dependencies

During the BPS installation process, the external dependencies are automatically resolved by the package and environment management system (i.e., Conda), using the setup specifications (setup.cfg file) contained in each package.
In particular, each BPS component has its own dependencies, as summarized here below (external dependencies in **bold**, with version if needed). The bps platform dependent packages that are not available on non GNU/Linux platforms are highlighted in red.

```
Environment setup (see Sec. 4.3)
    python>=3.12
bps-common
   arepyextras-runner
   arepytools
```

xsdata
xmlschema
gdal>=3.10,<3.11
numpy
bps-transcoder
arepytools>=1.7.6
gdal>=3.10,<3.11
netcdf4
numpy
opencv-python-headless
pystac
scipy>=1.10
bps-common
bps-l1_framing_processor
arepytools
click
xsdata
scipy
numpy
bps-common
bps-transcoder
bps-l1_pre_processor
arepytools
scipy>=1.10
numpy
netCDF4
bps-common
bps-transcoder
bps-l1_core_processor
arepytools
bps-common
bps-l1_processor
arepytools>=1.7.6
click
arepyextras-runner
numpy
unlzw3
scipy
bps-common
bps-l1_pre_processor
bps-l1_core_processor
bps-transcoder
shapely
pyproj
netcdf4
bps_stack_pre_processor
arepytools>=1.7.6
bps-common

```
   numpy
   scipy
bps_stack_coreg_processor
   arepytools>=1.7.6
   bps-common
   scipy
   numpy
   xmltodict
bps_stack_cal_processor
   arepytools>=1.7.6
   bps-common
   numba>=0.57.0
   numpy
   opencv-python-headless
   scipy
   scikit-image
bps_stack_processor
   arepytools>=1.7.6
   arepyextras-runner
   bps-common
   bps-stack_pre_processor
   bps-stack_coreg_processor
   bps-stack_cal_processor
   bps-transcoder
   click
   numpy
   xsdata
bps_l2a_processor
   arepytools
   click
   numba>=0.57
   numpy
   numexpr
   netCDF4
   opencv-python-headless
   gdal>=3.10,<3.11
   scipy
   bps-common
   bps-transcoder
bps_l2b_fd_processor
   arepytools
   click
   numba>=0.57
   numpy
   scipy
   bps-common
   bps-transcoder
bps_l2b_fh_processor
```

```
   arepytools
   click
   numba>=0.57
   numpy
   scipy
   bps-common
   bps-transcoder
bps_l2b_agb_processor
   arepytools
   click
   numba>= 0.57
   numpy
   gdal>=3.10,<3.11
   scipy
   bps-common
   bps-transcoder
bps_l3_processor
   arepytools
   click
   libsqlite>=3.47.0
   numba>=0.57
   numpy<2.0.0
   petsc4py>=3.21.5
   pillow>=11.0.0
   sqlite>=3.47.2
   xsdata
   bps-common
   bps-transcoder
```

4.4.2. Internal resources

In order to complete BPS installation and allow its usage, the corresponding internal resources shall be made available. In particular, each BPS processor has its own resources, as summarized in the following table.

BPS Processor	Internal resources
L1 Processor	• Digital Elevation Model (DEM) database • Geomagnetic Field (GMF) database • IRI Model (IRI) database
Stack Processor	• Forest/Non-Forest mask (FNF) database
L2a Processor	• Forest/Non-Forest mask (FNF) database
L2b FD Processor	/
L2b FH Processor	/

L2b AGB Processor	• Land Cover Map (LCM) database • Reference AGB (CAL_AB) database
L3 Processor	• Land Cover Map (LCM) database
L1 Framing Processor	/

Tab.4 BPS processors internal resources

Detailed content of each resource is described in [RD4].

The following delivery and installation strategies are adopted:

- for GMF, IRI, FNF, LCM and CAL_AB:
  - thanks to their reduced dimensions, are provided together with the BPS delivery bundle via dedicated compressed archives, i.e.:
    - BPS_GMF__VYYYYMMDD.tar.gz;
    - BPS_IRI__VYYYYMMDD.tar.gz;
    - BPS_FNF__VYYYYMMDD.tar.gz;
    - BPS_LCM__VYYYYMMDD.tar.gz;
    - BPS_CAL_AB__VYYYYMMDD.tar.gz;

    where the version is represented by the start validity date;
  - their installation can be performed simply extracting these archives in the desired system path;
- DEM:
  - due to its huge dimensions, is not provided directly but it shall be retrieved and installed following a dedicated procedure described in the next sub-section.

In case of installation using delivery bundle, it is enough to specify the installation path of these resources in the Job Order file (see [AD1]).
In case of installation using Docker images, in addition to this the installation path has to be mounted when running the processors execution (see example in Sec. 5.3.2).

Once installed, there resources are shared by the various BPS processors, which use the same installation path / mounting point.

4.4.2.1. DEM database

NOTE: In order to install DEM database, internet connection is required

To install the DEM database the following instructions shall be implemented:

1. Choose an install path (e.g. `/storage/DEM`)
2. Create two directories inside named "`copernicus`" and "`srtm`"

3. Due to its huge dimensions, the Copernicus DEM cannot be provided directly. Instructions to retrieve it are the following:
   - Download DEM and geoid from official websites [RD5][RD6] into `/storage/DEM/copernicus/COP-DEM_GLO-90-DGED-2021_1`. The versions used by BPS are:
     - DEM: COP-DEM_GLO-90-DGED-2021_1
     - Geoid: EGM2008-2.5 (gridded version)

     NOTE: In case of installation of different versions of the same DEM type, it will be enough to repeat the procedure changing the name of the folder above. The DEM to be used during processing will be then specified via auxiliary files (see [RD3]). In particular, to use these versions of DEM and geoid the heightModel field in AUX_PP1 file shall be set as:

     ```
     <heightModel version="COP-DEM_GLO-90-DGED-2021_1 EGM2008-
     2.5">Copernicus DEM</heightModel>
     ```

     Note also that information derived from DEM (latitude, longitude, height, incidence and look angles, …) are stored in L1 products in form of 2D raster (LUT). These are decimated w.r.t. output L1 products sampling grid according to parameters specified via auxiliary files (see [RD3]). The default decimation factors for a 90m-resolution DEM are 2(range) x 12(azimuth), while for a 30m one they are 1 x 4 (these allows to get LUT sampling steps close to the one of the input DEM);
   - Move to this path;
   - Rename geoid file into `egm2008-2.5.tif`;
   - Generate the DEM index file (`demIndex.xml`) using the dedicated tool included in the BPS delivery bundle (`copernicus_dem_extractor`):

     ```
     $ copernicus_dem_extractor generate-index --
     path=/path_to_copernicus_dem_folder --index=demIndex.xml
     ```

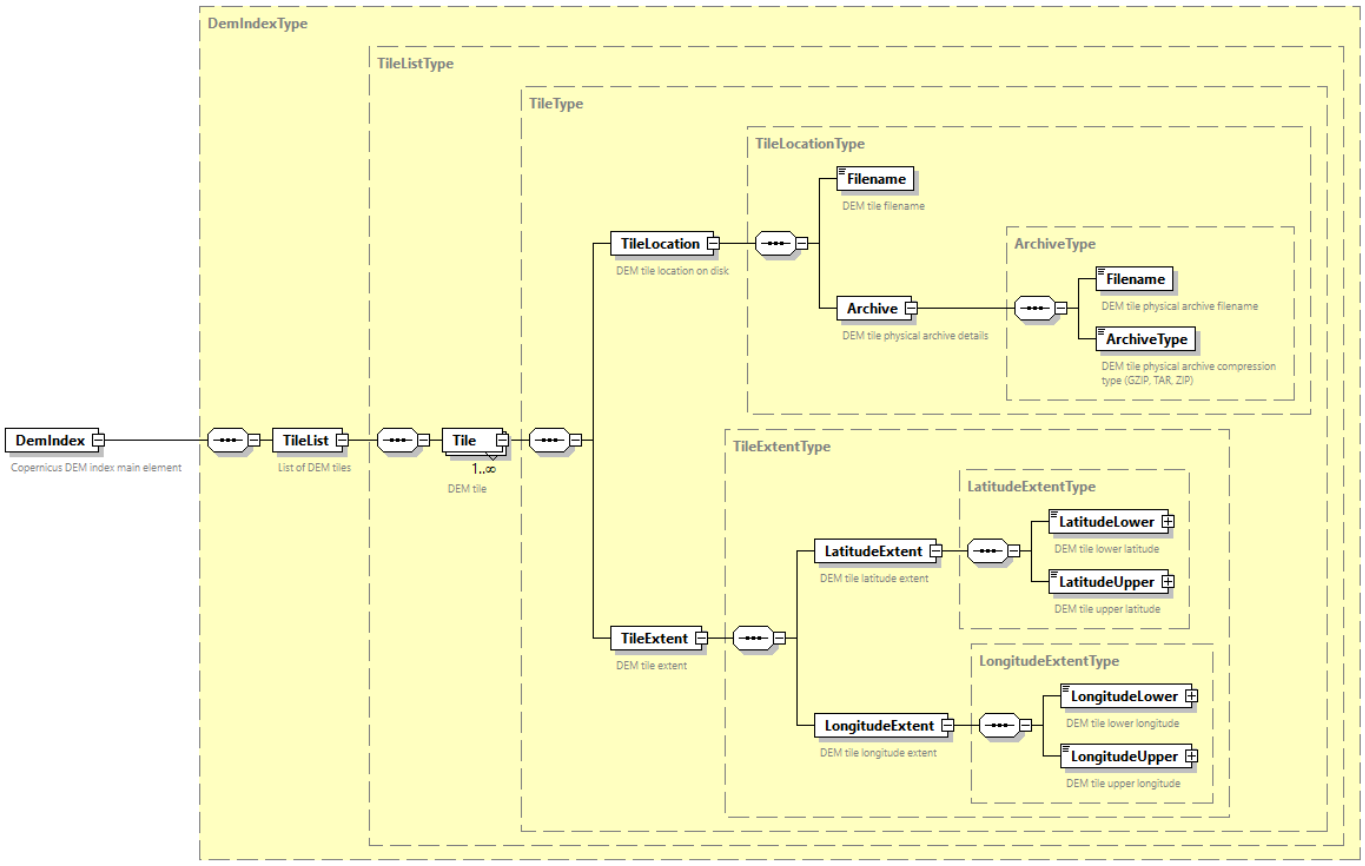
     Note that, for the execution of the BPS L1 processor, the DEM index file should be always called "`demIndex.xml`".

     The file is generated in the same folder of DEM files. Note that this tool is agnostic w.r.t. the physical structure of this folder and allows the BPS processors to access the DEM tiles independently from it. In any case it is strongly recommended, due the huge number of files, to keep a non-flat folder structure for them, where tiles are organized, e.g., per continent. Here below an extract of `demIndex.xml` file content:

     ```
     <?xml version="1.0" encoding="UTF-8"?>
     <DemIndex>
     ```

```
  <TileList>
    ... <!-- list of tiles-->
    <Tile>
      <TileLocation>
        <Filename>Copernicus_DSM_30_S15_00_W178_00/\
DEM/Copernicus_DSM_30_S15_00_W178_00_DEM.tif</Filename>
        <Archive>
          <Filename>Oceania/Wallis_and_Futuna/\
DEM1_SAR_DGE_90_20110609T173009_20121005T173028_ADS_000000_0920.DEM
.tar</Filename>
          <ArchiveType>TAR</ArchiveType>
        </Archive>
      </TileLocation>
      <TileExtent>
        <LatitudeExtent>
          <LatitudeLower unit="deg">-15.0</LatitudeLower>
          <LatitudeUpper unit="deg">-14.0</LatitudeUpper>
        </LatitudeExtent>
        <LongitudeExtent>
          <LongitudeLower unit="deg">-178.0</LongitudeLower>
          <LongitudeUpper unit="deg">-177.0</LongitudeUpper>
        </LongitudeExtent>
      </TileExtent>
    </Tile>
    ...
  </TileList>
</DemIndex>
```

The complete file format is described in the following figure.



Copernicus DEM index file complete structure and description.

In order to use the above-mentioned tool for the generation of the `demIndex.xml` file, the following installation steps are required in advance:

Fig

```
$ conda create -n dem_index_env python=3.9

$ conda activate dem_index_env

$ conda install -c conda-forge GDAL=3.8

$ conda install conda-build

$ mkdir -p dem_ch/noarch

$ cp -r ./bundle/copernicus-dem-index-generator/* dem_ch/noarch

$ conda index --verbose ${PWD}/dem_ch

$ conda config --prepend channels ${PWD}/dem_ch

$ conda install arepyextras-copernicus_dem_extractor
```

4. Due its dimensions, SRTM DEM ([RD7]) is provided directly, together with its geoid file, as a single compressed archive. In order to install it, it is enough to extract the archive in the `/storage/DEM/srtm/SRTM_v1` folder. The versions used by BPS are:
   - DEM: SRTM_v1
   - Geoid: EGM96

   NOTE: In case of installation of different versions of the same DEM type, it will be enough to repeat the procedure changing the name of the folder above. The DEM to be used during processing will be then specified via auxiliary files (see [RD3]). In particular, to use these versions of DEM and geoid the heightModel field in AUX_PP1 file shall be set as:

```
<heightModel version="SRTM_v1 EGM96">SRTM</heightModel>
```

It shall be noted that both Copernicus and SRTM DEM are considered as static resources, i.e. there is no pre-defined plan for their update. Only for Copernicus DEM and only if deemed necessary, on demand update will be performed, considering that yearly updates will be potentially available. In this case it will be enough to repeat the installation procedure to replace previous product with new one.
The SRTM DEM version used includes data at 3 arc-second (90m) resolution (except for United States where 1arc-second (30m) resolution data are available). BPS software

natively supports both the resolution levels (1 and 3 arc-second), even if it has been fully tested only with 3 arc-second one.
The SRTM DEM does not need any DEM index file.

NOTE: Additional instructions will be added in future versions of the document to install also BIOMASS DTM.

4.4.2.2. GMF database

As mentioned above, thanks to their reduced dimensions, these data are provided together with the BPS delivery bundle via dedicated compressed archives, i.e., BPS_GMF__VYYYYMMDD.tar.gz.
The archive contains the following files:

- igrf13.wmm
- igrf13.wmm.cof
- geomagnetic_poles.xml
- biomassGeomagneticPoles.xsd

The two igrf* files can be downloaded from [RD8].
The third file can be populated taking the information from [RD9]. The complete file format is described in the following figure (this corresponds to the content of biomassGeomagneticPoles.xsd file, included in the archive too).

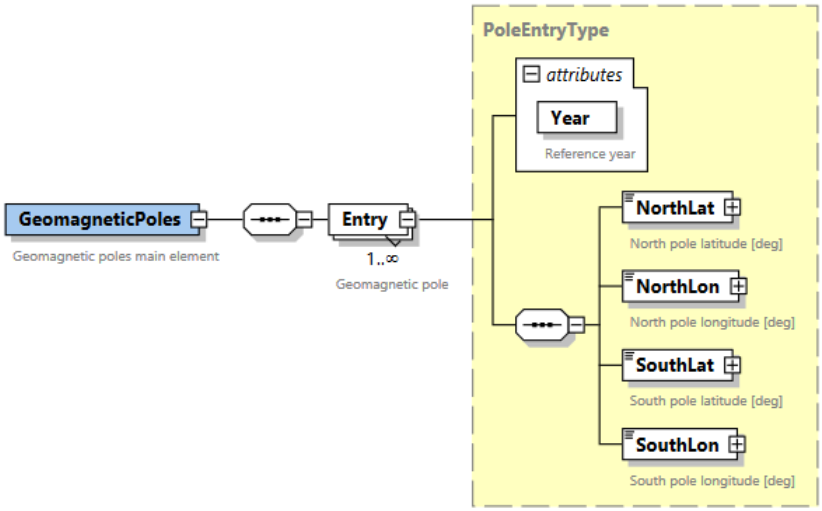


Fig.4

GMF geomagnetic poles file complete structure and description.

4.4.2.3. IRI database

The IRI database is a (flat) collection of 55 files that can be retrieved from the official https://irimodel.org website. The naming is reported in section 4.4.2. Below, the list of the 55 files and the corresponding download folder link:

- apf107.dat -> https://irimodel.org/indices/
- ccir11.asc -> https://irimodel.org/COMMON_FILES/
- ccir12.asc -> https://irimodel.org/COMMON_FILES/
- ccir13.asc -> https://irimodel.org/COMMON_FILES/
- ccir14.asc -> https://irimodel.org/COMMON_FILES/
- ccir15.asc -> https://irimodel.org/COMMON_FILES/
- ccir16.asc -> https://irimodel.org/COMMON_FILES/
- ccir17.asc -> https://irimodel.org/COMMON_FILES/
- ccir18.asc -> https://irimodel.org/COMMON_FILES/
- ccir19.asc -> https://irimodel.org/COMMON_FILES/
- ccir20.asc -> https://irimodel.org/COMMON_FILES/
- ccir21.asc -> https://irimodel.org/COMMON_FILES/
- ccir22.asc -> https://irimodel.org/COMMON_FILES/
- dgrf1945.dat -> https://irimodel.org/IRI-2016/
- dgrf1950.dat -> https://irimodel.org/IRI-2016/
- dgrf1955.dat -> https://irimodel.org/IRI-2016/
- dgrf1960.dat -> https://irimodel.org/IRI-2016/
- dgrf1965.dat -> https://irimodel.org/IRI-2016/
- dgrf1970.dat -> https://irimodel.org/IRI-2016/
- dgrf1975.dat -> https://irimodel.org/IRI-2016/
- dgrf1980.dat -> https://irimodel.org/IRI-2016/
- dgrf1985.dat -> https://irimodel.org/IRI-2016/
- dgrf1990.dat -> https://irimodel.org/IRI-2016/
- dgrf1995.dat -> https://irimodel.org/IRI-2016/
- dgrf2000.dat -> https://irimodel.org/IRI-2016/
- dgrf2005.dat -> https://irimodel.org/IRI-2016/
- dgrf2010.dat -> https://irimodel.org/IRI-2016/
- dgrf2015.dat -> https://irimodel.org/IRI-2016/
- ig_rz.dat -> https://irimodel.org/indices/
- igrf2020.dat -> https://irimodel.org/IRI-2016/
- igrf2020s.dat -> https://irimodel.org/IRI-2016/
- mcsat11.dat -> https://irimodel.org/IRI-2016/
- mcsat12.dat -> https://irimodel.org/IRI-2016/
- mcsat13.dat -> https://irimodel.org/IRI-2016/
- mcsat14.dat -> https://irimodel.org/IRI-2016/
- mcsat15.dat -> https://irimodel.org/IRI-2016/
- mcsat16.dat -> https://irimodel.org/IRI-2016/
- mcsat17.dat -> https://irimodel.org/IRI-2016/
- mcsat18.dat -> https://irimodel.org/IRI-2016/
- mcsat19.dat -> https://irimodel.org/IRI-2016/
- mcsat20.dat -> https://irimodel.org/IRI-2016/

- mcsat21.dat -> https://irimodel.org/IRI-2016/
- mcsat22.dat -> https://irimodel.org/IRI-2016/
- ursi11.asc -> https://irimodel.org/COMMON_FILES/
- ursi12.asc -> https://irimodel.org/COMMON_FILES/
- ursi13.asc -> https://irimodel.org/COMMON_FILES/
- ursi14.asc -> https://irimodel.org/COMMON_FILES/
- ursi15.asc -> https://irimodel.org/COMMON_FILES/
- ursi16.asc -> https://irimodel.org/COMMON_FILES/
- ursi17.asc -> https://irimodel.org/COMMON_FILES/
- ursi18.asc -> https://irimodel.org/COMMON_FILES/
- ursi19.asc -> https://irimodel.org/COMMON_FILES/
- ursi20.asc -> https://irimodel.org/COMMON_FILES/
- ursi21.asc -> https://irimodel.org/COMMON_FILES/
- ursi22.asc -> https://irimodel.org/COMMON_FILES/

4.5. Uninstallation Procedure

4.5.1. Uninstallation using delivery bundle

In order to properly uninstall BPS in case it has been installed from delivery bundle, the following steps shall be performed:

1. Remove the dedicated Python environment (e.g., bps_env):

```
$ conda remove --name bps_env -all
```

2. Remove the folder where conda channels have been created;

Remove the folders where C++ binaries (executables and libraries) have been installed; In the example from Sec 4.3.1, the user can remove /opt/bps/vXYZ.

4.5.2. Uninstallation using Docker images

In case of usage of Docker images only the following command is required:

```
$ docker image rm bps-xxx-processor:M.m.p
```

where bps-xxx-processor:M.m.p is the image name.

5. User Manual

5.1. Usage Pre-requisites

In order to properly use BPS starting from delivery bundle installation, the following pre-requisites shall be satisfied:

1. The Python environment created during the installation process shall be activated:

```
$ conda activate bps_env
```

2. C++ binaries folders, including executables and libraries, shall be added to the path as done during installation from bundle:

```
$ export PATH=${PATH}:/path_to_bin_folder
$ export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/path_to_lib_folder
```

In case of usage of Docker images, no additional operation is needed, being the software already pre-installed on the provided image.

5.2. Commands and Options

The main commands and options made available for each BPS processor can be displayed in its help message:

```
$ bps_l1_processor --help
$ bps_stack_processor --help
$ bps_l2a_processor --help
$ bps_l2b_fd_processor --help
$ bps_l2b_fh_processor --help
$ bps_l2b_agb_processor --help
$ bps_l3_processor --help
```

```
$ bps_l1_framing_processor --help
```

For example, for the L1 Processor this command gives in output:

```
Usage: bps_l1_processor [OPTIONS] JOB_ORDER_FILE

  BIOMASS Processing Suite - L1 Processor

  Starts processing from Job Order file

Options:
  --working-dir PATH    Working directory (optional)
  --version             Show processor version and exit
  --help                Show this message and exit
```

5.3. Usage Procedure

5.3.1. Usage in case of installation from delivery bundle

In order to properly use BPS in case it has been installed from delivery bundle, the following "minimal" command shall be executed:

```
$ bps_l1_processor path_to_JobOrder.xml

$ bps_stack_processor path_to_JobOrder.xml

$ bps_l2a_processor path_to_JobOrder.xml

$ bps_l2b_fd_processor path_to_JobOrder.xml

$ bps_l2b_fh_processor path_to_JobOrder.xml

$ bps_l2b_agb_processor path_to_JobOrder.xml

$ bps_l3_processor path_to_JobOrder.xml

$ bps_l1_framing_processor path_to_JobOrder.xml
```

All the input and output data and main processing parameters are specified in the Job Order, as described in [AD1]. Job Order is the main input for each BPS processor and the only mandatory one.
In alternative, in order to use all the available options, the command is:

```
$ bps_l1_processor \
--working-dir path_to_working_directory \
path_to_JobOrder.xml
```

where are specified:

- the name of the processor (in the example `bps_l1_processor` for L1 processor);
- the path to the working directory where intermediate data shall be stored (note: the path to the directory where output products shall be stored is indicated in the Job Order). This is not a required interface: all the BPS processors can run without specifying a working directory. In such a case, the processors will use the same directory where the Job Order is located, which is the standard behaviour of the processors. The only exception is the L1 processor: when the command line option is not specified, it will use the RAMDISK mount point from the Job Order, if provided, otherwise it will use the directory of the Job Order.

It has to be noted that different versions of the same processor can be used on the same system given that:

- requirements on installation of different processor versions are satisfied (see Sec. 4.3.1);
- at runtime, the Python environment corresponding to the desired version is activated and the environment path is updated adding on top the one of the corresponding C++ binaries (executables and libraries);
- different working directories are used.

In case of different users using the same installation, different working directories shall be used, placing Job Orders in different folders or specifying working directories through the `--working-dir` command line option.

5.3.2. Usage in case of installation from Docker images

In case of usage of Docker images, the following command shall be executed:

```
$ INPUT_DIR=<path to host folder containing input data>
$ MNT_INPUT_DIR=<path to guest folder containing input data>
$ DEM_DIR=<path to host folder containing DEM>
$ MNT_DEM_DIR=<path to guest folder containing DEM>
$ GMF_DIR=<path to host folder containing GMF>
$ MNT_GMF_DIR=<path to guest folder containing GMF>
$ IRI_DIR=<path to host folder containing IRI>
$ MNT_IRI_DIR=<path to guest folder containing IRI>
$ OUTPUT_DIR=<path to host folder to contain output data>
$ MNT_OUTPUT_DIR=<path to guest folder to contain output data>
$ MNT_RAM_DISK_DIR=<path to guest folder where the RAMDISK will be
mount>
```

```
$ docker run --rm -t \
-w ${MNT_OUTPUT_DIR} \
-v ${OUTPUT_DIR}:${MNT_OUTPUT_DIR} \
-v ${INPUT_DIR}:${MNT_INPUT_DIR}:ro \
-v ${DEM_DIR}:${MNT_DEM_DIR}:ro \
-v ${GMF_DIR}:${MNT_GMF_DIR}:ro \
-v ${IRI_DIR}:${MNT_IRI_DIR}:ro \
--mount type=tmpfs,destination=${MNT_RAM_DISK_DIR},tmpfs-size=24g \
-u $(id -u):$(id -g) \
bps-l1-processor:02.22 bps_l1_processor JobOrder.xml
```

where the various fields can be customized by the orchestrator. As visible, it is not needed to specify additional options. In addition, mounting points for internal resources have been specified.

For the various BPS processors, the command above shall be updated in the following way:

- IMAGENAME: specified at Docker build time, it is the same one specified in Task Table and Job Orders. Can be: `bps-l1-processor`, `bps-stack-processor`, `bps-l2a-processor`, `bps-l2b-fd-processor`, `bps-l2b-fh-processor`, `bps-l2b-agb-processor`, `bps-l1-framing-processor`;
- IMAGETAG: specified at Docker build time, it is the same one specified in Task Table and Job Orders. Follows the convention described in Sec. 3;
- Processor name: Can be: `bps_l1_processor`, `bps_stack_processor`, `bps_l2a_processor`, `bps_l2b_fd_processor`, `bps_l2b_fh_processor`, `bps_l2b_agb_processor`, `bps_l1_framing_processor`;
- Internal resources (DEM, …): different for each processor, see Sec. 4.4.2;
- MNT_RAM_DISK_DIR: this is used only for L1_P Processor, which is the only processor using shared memory to store intermediate data and needs to mount the corresponding resource (see also Sec. 4.3.2 and 5.3.7).

For example, the command for STA_P Processor will be:

```
$ INPUT_DIR=<path to host folder containing input data>
$ MNT_INPUT_DIR=<path to guest folder containing input data>
$ FNF_DIR=<path to host folder containing FNF>
$ MNT_FNF_DIR=<path to guest folder containing FNF>
$ OUTPUT_DIR=<path to host folder to contain output data>
$ MNT_OUTPUT_DIR=<path to guest folder containing Job Order and to
contain output data>

$ docker run --rm -t \
-w ${MNT_OUTPUT_DIR} \
-v ${OUTPUT_DIR}:${MNT_OUTPUT_DIR} \
-v ${INPUT_DIR}:${MNT_INPUT_DIR}:ro \
-v ${FNF_DIR}:${MNT_FNF_DIR}:ro \
-u $(id -u):$(id -g) \
bps-stack-processor:02.22 bps_stack_processor JobOrder.xml
```

This way of triggering the processing is in common to all the BPS processors. Then, for each one a specific processing chain is instantiated (e.g., for L1 Processor, three steps called Pre-, Core- and Post-Processing are executed in sequence). For all the details refer to [RD1].

It is important to recall that:

- all the paths reported in the JobOrder refers to the filesystem of the docker image. For this reason, all input data needs to be mounted from the host device into a mount position. For an example see the scripts above where image paths are stored in variables starting with MNT_ (which stands for mount);
- when using the RAMDISK (available only for L1 processor), the size of the tempfs should be specified. This size should be smaller than the available host RAM and should be large enough to store all intermediates files. The MNT_RAM_DISK_DIR path should be also reported in the job order (see bullet above). The value for the reference configuration is specified in the table below;
- when running docker it is possible (although not required) to specify host limits:

```
--cpu-period=1000000 --cpu-quota=${CPU_LIMIT}

--memory=${MEMORY_LIMIT} --memory-swap=${MEMORY_LIMIT}
```

See the table below for the value in the reference configurations.

Processor	Memory Limit	Of which tmpfs size	CPU Limit
L1F	2048m		1000000
L1	25600m	15360m	6000000
STA	65536m		7000000
L2A	30720m		8000000
L2B_FH	9216m		6000000
L2B_FD	9216m		6000000
L2B_AGB	65536m		6000000

- in L1P, when using RAMDISK stop-and-resume functionality is enabled only when the intermediates are enabled (see dedicated section below).

5.3.3. Processors Configuration

The BIOMASS satellite primarily operates in the Stripmap acquisition mode, where it illuminates a swath of approximately 50 km using a single antenna beam. This mode allows for comprehensive global coverage by interleaving operations among three complementary

swaths (identified as S1, S2 and S3). In addition to this, a specific acquisition mode, called RX-Only, has been defined and will be used for specific characterization activities e.g. on noise or Radio-Frequency Interferences (RFI). In addition, depending on the specific phase of the mission (Tomographic or Interferometric), on the target acquired (normal scene or area above PARC (or BCT)), on the specific processing needs, … it is possible to process the acquired data in specific configurations.
In the following sub-sections it is described how to configure the various BPS processors to trigger them and which are the main processing options available.

5.3.3.1. L1 Processor

The L1 Processor (L1_P) implements two processing chains (or tasks):

- L1 Standard Processing (L1_P), aimed at generating a L1a products (one frame, both standard and monitoring) and (optionally, default on) a L1b products (standard) starting from a L0 product (one slice, both standard and monitoring);
- L1 RX-Only Processing (L1_RXO_P), aimed at generating a L1a products (one frame, only standard) starting from a L0 RX-Only product (one slice, only standard).

The terms “slice” and “frame” are used respectively to indicate L0 and L1 products and are referred to the products cutting strategy implemented for BIOMASS:

- the L0 Processor is in charge of implementing the L0 slicing, i.e. to derive a set of L0 slices (by default, ~100s-long products) starting from each acquired datatake (that can be also of several minutes);
- the L1 Framing Processor is in charge of implementing the L1 framing, i.e. to compute the start and stop times of each frame (by default, ~21s-long products) contained in a given L0 slice and to pass them to the L1 Processor (via Job Order, see below) to operate the real cutting operation.

The terms “standard” and “monitoring” are used to indicate the type of product: the standard one is complete and contains both measurement and annotation data, while the monitoring one contains only annotations.
In addition to this, L1 Processor requires in input for its execution also a set of auxiliary files and internal resources.
The complete list of inputs, and of corresponding outputs, is reported in the following tables, divided per L1 Processor tasks, indicating also the criteria to be used for selecting each input and the destinations of each output inside the overall BPS processing chain.
Please refer to [AD1] and [AD2] for the complete description of each field of these tables (a summary is reported also in Sec. 9) and to [RD4] for the complete description of each auxiliary file and internal resource.

L1 Standard Processing (L1_P) task: inputs and outputs

Product Type	Description	Card.	Origin	Instances	Start / Stop Offset	Overlap / Min Overlap	Metadata Names	Max. # of Files	Ordering

Sx_RAW__0S	Stripmap Level-0 – Standard product	1	External	Single	0 / 0	Yes / 100	/	1	Latest
Sx_RAW__0M	Stripmap Level-0 – Monitoring product	1	External	Single	0 / 0	Yes / 100	/	1	MaxTemporalOverlap, Latest
AUX_ORB___	Auxiliary Orbit	1	External	Single	0 / 0	Yes / 100	/	1	MaxTemporalOverlap, ClosestStart
AUX_ATT___	Auxiliary Attitude	1	External	Single	0 / 0	Yes / 100	/	1	MaxTemporalOverlap, ClosestStart
AUX_TEC___	Auxiliary TEC Map	1, 2	External	Single	0 / 0	Yes / 0	/	2	MaxTemporalOverlap, ClosestStart
AUX_INS___	Auxiliary Instrument Parameters	1	External	Single	0 / 0	Yes / 100	/	1	MaxTemporalOverlap, ClosestStart
AUX_PP1___	Auxiliary L1 Processing Parameters	1	External	Single	0 / 0	Yes / 100	/	1	MaxTemporalOverlap, ClosestStart
PARC_INFO_	Auxiliary Calibration Site Information	0, 1	External	Single	0 / 0	Yes / 100	/	1	MaxTemporalOverlap, ClosestStart

Product Type	Description	Card.	Dest.	Notes
Sx_SCS__1S	Stripmap L1 SCS – Standard product	1	External (STA_P)	Collected by PF and used as input by Stack processor STA_P task
Sx_SCS__1M	Stripmap L1 SCS – Monitoring product	1	External	Collected by PF, but not used by other BPS tasks
Sx_DGM__1S	Stripmap L1 DGM – Standard product	0..1	External	Collected by PF, but not used by other BPS tasks Generated by default
Sx_SCSc_1S	Stripmap L1 SCS Calibration – Standard product	0, 4	External	Collected by PF, but not used by other BPS tasks Generated only if PARC_INFO_ product is provided in input and only in case of processing of data acquired over PARC

L1 RX-Only Processing (L1_RXO_P) task: inputs and outputs

Product Type	Description	Card.	Origin	Instances	Start / Stop Offset	Overlap / Min Overlap	Metadata Names	Max. # of Files	Ordering
RO_RAW__0S	RX-Only Mode Level-0 – Standard product	1	External	Single	0 / 0	Yes / 100	/	1	Latest
AUX_ORB___	Auxiliary Orbit	1	External	Single	0 / 0	Yes / 100	/	1	MaxTemporalOverlap, ClosestStart
AUX_ATT___	Auxiliary Attitude	1	External	Single	0 / 0	Yes / 100	/	1	MaxTemporalOverlap,

									ClosestStart
AUX_INS___	Auxiliary Instrument Parameters	1	External	Single	0 / 0	Yes / 100	/	1	MaxTemporalOverlap, ClosestStart
AUX_PP1___	Auxiliary L1 Processing Parameters	1	External	Single	0 / 0	Yes / 100	/	1	MaxTemporalOverlap, ClosestStart

Product Type	Description	Card.	Dest.	Notes
RO_SCS__1S	RX-Only Mode L1 SCS – Standard product	1	External	Collected by PF, but not used by other BPS tasks

Job Order preparation

To trigger a L1 Processor execution a dedicated Job Order has to be filled. Here below are reported the list of sections and fields, with the instructions on how to fill them:

- `Processor_Name`, that shall be L1_P;
- `Task_Name`, that can be L1_P or L1_RXO_P (only one task for each Job Order);
- `Processor_Version` / `Task_Version`, that shall contain the processor version (in the form MM.mp, with M for major, m for minor and p for patch);
- a section (`List_of_Inputs`) where to specify all the input products and auxiliary files paths, selected according to the rules listed in the tables above, i.e.:
  - Sx_RAW__0S or RO_RAW__0S: Stripmap mode or RX-Only mode Level-0 Standard (L0S) product (depending on the task);
  - Sx_RAW__0M: Stripmap mode Level-0 Monitoring (L0M) product (not present in case of RX-Only mode);
  - AUX_ORB___: Auxiliary Orbit product, including input position and velocity state vectors (one per second);
  - AUX_ATT___: Auxiliary Attitude product, including input quaternions (one per second);
  - AUX_TEC___: Auxiliary TEC Map product, including input IONEX maps providing the vertical Total Electron Content (can be 2 products if L0S product is crossing midnight; not present in case of RX-Only mode);
  - AUX_INS___: Auxiliary Instrument Parameters product, containing the instrument parameters, i.e., mainly the internal calibration default parameters and the acquisition timeline for each acquisition mode, the doublets antenna patterns, the raw data decoding parameters, the instrument characterization data and the attenuators settings;
  - AUX_PP1___: Auxiliary L1 Processing Parameters product, containing the processing parameters for each L1_P processing chain step (one section of the XML file for each step, each parameter containing dedicated attributes in case it is possible to set it in a swath- or polarization-dependent way). This is the input the user may want to update in case he needs to modify something

related to processing configuration. Full detailed description of this file is reported in [RD3];
        - Note that the L1 Processor uses the parameters stored in AUX_INS___ and AUX_PP1___ to populate internal configuration files that contain, in addition to these, several other lower-level processing parameters. These files are used as an internal interface towards the L1 Processor binary dependencies (L1PreProcessor and L1CoreProcessor). The list of these files is reported in Sec. 5.3.6, while their complete specifications can be found in [RD1] annex;
    - PARC_INFO_: Auxiliary Calibration Site Information product, containing specific information about PARC (status, position, delays and RCS). Note that this product is optional, to be provided only in case data is acquired over PARC to activate dedicated PARC mode processing. In this mode the processing chain is automatically triggered several times, i.e. one time in which the entire image is processed nominally and 4 times (one per transponder scattering mechanism) where just the portion of data around PARC is processed properly compensating the delay introduced by the transponder;
- a section (`List_of_Cfg_Files`) where to specify all the internal resources path, i.e.:
    - Digital Elevation Model (DEM) database;
    - Geomagnetic Field (GMF) database;
    - IRI data (IRI) database;

  Note that in this field just the path to the generic folder containing the internal resource shall be specified. Then it is up to the processor to select the right one to use during processing, basing on information contained in auxiliary files (see [RD3] and Sec. 4.4.2.1);
- a section (`List_of_Outputs`) where to specify all the output products paths, i.e.:
    - Sx_SCS__1S or RO_SCS__1S: Stripmap mode or RX-Only mode L1 SCS Standard (L1aS) product (depending on the task);
    - Sx_SCS__1M: Stripmap mode L1 SCS Monitoring (L1aM) product (not generated in case of RX-Only mode);
    - Sx_DGM__1S: Stripmap L1 DGM Standard (L1bS) product (note: this product is optional, if not specified it is not generated; not generated in case of RX-Only mode);
    - Sx_SCSc_1S: Stripmap L1 SCS Calibration – Standard product (note: these products (4 in total) are optional, generated only in case PARC mode has been activated; not generated in case of RX-Only mode);
- a flag (`Intermediate_Output_Enable`) and a section (`List_of_Intermediate_Outputs`) where to specify if intermediate data shall be kept after processing completion and at which path (see Sec. 5.3.6 and 5.3.7);
- sections (`Request` and `List_of_Proc_Parameters`) to specify all the optional processing parameters, i.e.:

- TOI: time coordinates defining the azimuth extent of input L0 data being processed (in [UTC]). If not specified, no framing is performed and the entire input L0S product (slice) is processed;
- frame_id: absolute frame index of the frame to be generated (expressed as a string of 3 digits zero padded, starting from 1, or as '___' in case no framing is performed;
- frame_status: frame status (NOMINAL, MERGED, PARTIAL, INCOMPLETE, NOT_FRAMED);
  - Note: these three parameters (TOI, frame_id, frame_status) are computed by the L1 Framing Processor, who is in charge of implementing the L1 framing strategy (see Sec. 5.3.3.6);
- range_start_time / range_stop_time: time coordinates defining the slant-range extent of input L0 data being processed (in [s]). If not specified, the whole range extent of the input data is processed;
- rfi_mitigation_flag: flag to activate RFI detection and mitigation step (True if it has to be performed, False otherwise). If specified, this parameter overrules the homonymous one in AUX_PP1___ input Auxiliary Product;

- fields (`Stdout_Log_Level` and `Stderr_Log_Level`) to set the log level;
- fields (`Number_of_CPU_Cores`, `Amount_of_RAM`, `RAMDISK` and `Disk_Space`) to set the hardware resources allocated for the processing:
  - for L1 Processor, `RAMDISK` is used to store intermediate data while `Disk_Space` to store input and output products. The size of tmpfs value when calling docker image (see Sec. 5.3.2) shall be in principle equal to the sum of `Amount_of_RAM` and `RAMDISK`. Note that L1 Processor is the only processor using shared memory to store intermediate data, as described in Sec. 5.3.2 and 5.3.6. In addition, this section is needed only in case of usage via docker images, while it is not necessary in case of usage via delivery bundle (see note at the end of this section).

For the complete description of Job Order structure and fields refer to [AD2] and for the details on how to fill them to [AD1].

Note that, in case of execution from Docker image, instructions reported in Sec. 5.3.2 shall be followed. For what concern the volumes mounting, all the paths in the Job Order file shall refer to the Docker container file system. For what concern the tempfs setting, see note in the last bullet above).

In the box below a sample L1 Processor Job Order is reported.

```
<?xml version="1.0" encoding="UTF-8"?>
<Job_Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
schemaName="BIOMASS CPF-Processor ICD" schemaVersion="String">
  <Processor_Configuration>
    <File_Class>File_Class</File_Class>
    <Processor_Name>L1_P</Processor_Name>
    <Processor_Version>03.30</Processor_Version>
    <Processing_Node>Processing_Node</Processing_Node>
    <List_of_Stdout_Log_Levels>
```

```
      <Stdout_Log_Level>INFO</Stdout_Log_Level>
    </List_of_Stdout_Log_Levels>
    <List_of_Stderr_Log_Levels>
      <Stderr_Log_Level>INFO</Stderr_Log_Level>
    </List_of_Stderr_Log_Levels>
    <Intermediate_Output_Enable>true</Intermediate_Output_Enable>
    <Processing_Station>Processing_Station</Processing_Station>
    <Request>
      <TOI>
        <Start>2017-02-25T09:45:35.620307</Start>
        <Stop>2017-02-25T09:46:00.620307</Stop>
      </TOI>
    </Request>
  </Processor_Configuration>
  <List_of_Tasks>
    <Task>
      <Task_Name>L1_P</Task_Name>
      <Task_Version>03.30</Task_Version>
      <Number_of_CPU_Cores>6</Number_of_CPU_Cores>
      <Amount_of_RAM>10240</Amount_of_RAM>
      <List_of_RAMDISKs>
        <RAMDISK>
          <Amount>15360</Amount>
          <Mount_Path>/ramdisk</Mount_Path>
        </RAMDISK>
      </List_of_RAMDISKs>
      <Disk_Space>6144</Disk_Space>
      <List_of_Proc_Parameters>
        <Proc_Parameter>
          <Name>frame_id</Name>
          <Value>001</Value>
        </Proc_Parameter>
        <Proc_Parameter>
          <Name>frame_status</Name>
          <Value>NOMINAL</Value>
        </Proc_Parameter>
      </List_of_Proc_Parameters>
      <List_of_Cfg_Files>
        <Cfg_File>
          <Cfg_ID>DEM</Cfg_ID>
          <Cfg_File_Name>/storage/DEM</Cfg_File_Name>
        </Cfg_File>
        <Cfg_File>
          <Cfg_ID>GMF</Cfg_ID>
          <Cfg_File_Name> /storage/GMF</Cfg_File_Name>
        </Cfg_File>
        <Cfg_File>
          <Cfg_ID>IRI</Cfg_ID>
          <Cfg_File_Name> /storage/IRI</Cfg_File_Name>
        </Cfg_File>
      </List_of_Cfg_Files>
      <List_of_Inputs>
        <Input>
          <Input_ID>Sx_RAW__0x</Input_ID>
          <Alternative_ID>0001</Alternative_ID>
          <List_of_Selected_Inputs>
            <Selected_Input>
              <File_Type>S1_RAW__0S</File_Type>
              <List_of_File_Names>
                <File_Name>path to L0S product</File_Name>
```

```
            </List_of_File_Names>
          </Selected_Input>
        </List_of_Selected_Inputs>
      </Input>
      <Input>
        <Input_ID>Sx_RAW__0x</Input_ID>
        <Alternative_ID>0001</Alternative_ID>
        <List_of_Selected_Inputs>
          <Selected_Input>
            <File_Type>S1_RAW__0M</File_Type>
            <List_of_File_Names>
              <File_Name>path to L0M product</File_Name>
            </List_of_File_Names>
          </Selected_Input>
        </List_of_Selected_Inputs>
      </Input>
      <Input>
        <Input_ID>AUX_ORB___</Input_ID>
        <Alternative_ID>0001</Alternative_ID>
        <List_of_Selected_Inputs>
          <Selected_Input>
            <File_Type>AUX_ORB___</File_Type>
            <List_of_File_Names>
              <File_Name>path to AUX_ORB product</File_Name>
            </List_of_File_Names>
          </Selected_Input>
        </List_of_Selected_Inputs>
      </Input>
      <Input>
        <Input_ID>AUX_ATT___</Input_ID>
        <Alternative_ID>0001</Alternative_ID>
        <List_of_Selected_Inputs>
          <Selected_Input>
            <File_Type>AUX_ATT___</File_Type>
            <List_of_File_Names>
              <File_Name>path to AUX_ATT product</File_Name>
            </List_of_File_Names>
          </Selected_Input>
        </List_of_Selected_Inputs>
      </Input>
      <Input>
        <Input_ID>AUX_TEC___</Input_ID>
        <Alternative_ID>0001</Alternative_ID>
        <List_of_Selected_Inputs>
          <Selected_Input>
            <File_Type>AUX_TEC___</File_Type>
            <List_of_File_Names>
              <File_Name>path to AUX_TEC product</File_Name>
            </List_of_File_Names>
          </Selected_Input>
        </List_of_Selected_Inputs>
      </Input>
      <Input>
        <Input_ID>AUX_INS___</Input_ID>
        <Alternative_ID>0001</Alternative_ID>
        <List_of_Selected_Inputs>
          <Selected_Input>
            <File_Type>AUX_INS___</File_Type>
            <List_of_File_Names>
              <File_Name>path to AUX_INS product</File_Name>
```

```
              </List_of_File_Names>
            </Selected_Input>
          </List_of_Selected_Inputs>
        </Input>
        <Input>
          <Input_ID>AUX_PP1___</Input_ID>
          <Alternative_ID>0001</Alternative_ID>
          <List_of_Selected_Inputs>
            <Selected_Input>
              <File_Type>AUX_PP1___</File_Type>
              <List_of_File_Names>
                <File_Name>path to AUX_PP1 product</File_Name>
              </List_of_File_Names>
            </Selected_Input>
          </List_of_Selected_Inputs>
        </Input>
      </List_of_Inputs>
      <List_of_Outputs>
        <Output>
          <File_Type>S1_SCS__1S</File_Type>
          <File_Name_Pattern>BIO_S1_SCS__1S_*</File_Name_Pattern>
          <File_Dir>path to output folder</File_Dir>
          <Baseline>01</Baseline>
        </Output>
        <Output>
          <File_Type>S1_SCS__1M</File_Type>
          <File_Name_Pattern>BIO_S1_SCS__1M_*</File_Name_Pattern>
          <File_Dir>path to output folder</File_Dir>
          <Baseline>01</Baseline>
        </Output>
        <Output>
          <File_Type>S1_DGM__1S</File_Type>
          <File_Name_Pattern>BIO_S1_DGM__1S_*</File_Name_Pattern>
          <File_Dir>path to output folder</File_Dir>
          <Baseline>01</Baseline>
        </Output>
      </List_of_Outputs>
      <List_of_Intermediate_Outputs>
        <Intermediate_Output>
          <Intermediate_Output_ID>IntermediateDataDir</Intermediate_Output_ID>
          <Intermediate_Output_File>path to intermediate data
folder</Intermediate_Output_File>
        </Intermediate_Output>
      </List_of_Intermediate_Outputs>
    </Task>
  </List_of_Tasks>
</Job_Order>
```

Sample execution results

The following screenshot describes the results of a successful execution of the L1 Processor when executed with this Job Order in input.

As visible, there are:

- an output directory, where the official output products are stored. This is the folder specified in the Job Order used to trigger the processing;

- a working directory, where the intermediate data are stored. This folder is deleted at the end of the processing if specified in the Job Order (via `Intermediate_Output_Enable` flag).

```
├── output_dir
│   ├──
BIO_S1_SCS__1S_20170101T060307_20170101T060332_I_G03_M03_C03_T010_F
001_01_C1S4MK
│   ├──
BIO_S1_SCS__1M_20170101T060307_20170101T060332_I_G03_M03_C03_T010_F
001_01_C1S4OQ
│   └──
BIO_S1_DGM__1S_20170101T060307_20170101T060332_I_G03_M03_C03_T010_F
001_01_C1S4PO
└── intermediates
    ├── BPSConf.xml
    ├── L1PreProcessorInputFile.xml
    ├── L1PreProcessorConf.xml
    ├── L1CoreProcessorInputFile.xml
    ├── L1CoreProcessorProcessingOptions.xml
    ├── L1CoreProcessorProcessingParameters.xml
    ├── BIO_L1_P_01.02_20230201T165555.log
    ├── BPSL1ProcessorStatusFile.json
    ├── l1_pre_processor_output
    │   ├── iRAW
    │   ├── iRAWDynCAL
    │   └── ...
    └── l1_core_processor_output
        ├── iRGC
        ├── iSLC
        └── ...
```

In case of delivery bundle installation, it is possible to omit the `List_of_RAMDISKs` tag or to leave it empty. In this case intermediate data are stored on disk (and `Disk_Space` shall be increased accordingly). Below a snippet of the portion of the L1 Processor JobOrder.

```
<Task_Name>L1_P</Task_Name>
<Task_Version>03.30</Task_Version>
<Number_of_CPU_Cores>6</Number_of_CPU_Cores>
<Amount_of_RAM>10240</Amount_of_RAM>
<List_of_RAMDISKs/>
<Disk_Space>21504</Disk_Space>
```

5.3.3.2. Stack Processor

The Stack Processor (STA_P) implements one processing chain (or task):

- Stack Processing (STA_P), aimed at generates a set (3 for INT phase, 7 for TOM phase, by default) of L1c products (both standard and monitoring) starting from a set of L1a products (standard).

In addition to this, Stack Processor requires in input for its execution also a set of auxiliary files and internal resources.
The complete list of inputs, and of corresponding outputs, is reported in the following tables, indicating also the criteria to be used for selecting each input and the destinations of each output inside the overall BPS processing chain.
Please refer to [AD1] and [AD2] for the complete description of each field of these tables (a summary is reported also in Sec. 9) and to [RD4] for the complete description of each auxiliary file and internal resource.

Stack Processing (STA_P) task: inputs and outputs

Product Type	Description	Card.	Origin	Instances	Start / Stop Offset	Overlap / Min Overlap	Metadata Names	Max. # of Files	Ordering
Sx_SCS__1S	Stripmap Level-1 – SCS product	2..A	External (L1_P)	Single	-	-	Mission Phase Global Cycle Identifier Major Cycle Identifier Swath Identifier Track Number Frame Number *	8	Latest
AUX_PPS___	Auxiliary Stack Processing Parameters	1	External	Single	0 / 0	Yes / 100	/	1	MaxTemporalOverlap, ClosestStart

Product Type	Description	Card.	Dest.	Notes
Sx_STA__1S	Stripmap L1 Coregistered stack – Standard product	2..A	External (L2A_P)	Collected by PF and used as input by L2a processor L2A_P task
Sx_STA__1M	Stripmap L1 Coregistered stack – Monitoring product	2..A	External	Collected by PF, but not used by other BPS tasks

Job Order preparation

To trigger a Stack Processor execution a dedicated Job Order has to be filled. Here below are reported the list of sections and fields, with the instructions on how to fill them:

- `Processor_Name`, that shall be STA_P;
- `Task_Name`, that shall be STA_P (only one task for each Job Order);

- `Processor_Version` / `Task_Version`, that shall contain the processor version (in the form MM.mp, with M for major, m for minor and p for patch);
- a section (`List_of_Inputs`) where to specify all the input products and auxiliary files paths, selected according to the rules listed in the tables above, i.e.:
  - Sx_SCS__1S: Stripmap mode Level-1a Standard (L1aS) products (up to 8);
  - AUX_PPS___: Auxiliary Stack Processing Parameters product, containing the processing parameters for each STA_P processing chain step (one section of the XML file for each step, each parameter containing dedicated attributes in case it is possible to set it in a swath- or polarization-dependent way). This is the input the user may want to update in case he needs to modify something related to processing configuration. Full detailed description of this file is reported in [RD3];
    - Note that the Stack Processor uses the parameters stored in AUX_PPS___ to populate internal configuration files that contain, in addition to these, several other lower-level processing parameters. These files are used as an internal interface towards the Stack Processor binary dependency (BPSStackProcessor). The list of these files is reported in Sec. 5.3.6, while their complete specifications can be found in [RD1] annex;
- a section (`List_of_Cfg_Files`) where to specify all the internal resources path, i.e.:
  - Forest/Non-Forest mask (FNF) database;

  Note that in this field just the path to the generic folder containing the internal resource shall be specified. Then it is up to the processor to select the right one to use during processing, basing on information contained in auxiliary files (see [RD3] and Sec. 4.4.2.1);
- a section (`List_of_Outputs`) where to specify all the output products paths, i.e.:
  - Sx_STA__1S: Stripmap mode L1 Coregistered Stack Standard (L1cS) products (up to 8);
  - Sx_STA__1M: Stripmap mode L1 Coregistered Stack Monitoring (L1cM) products (up to 8);
- a flag (`Intermediate_Output_Enable`) and a section (`List_of_Intermediate_Outputs`) where to specify if intermediate data shall be kept after processing completion and at which path (see Sec. 5.3.6 and 5.3.7);
- a section (`List_of_Metadata_Parameters`) where to specify the criteria used to select the input L1 products to be used for stack processing:
  - missionPhase: mission phase (INTERFEROMETRIC or TOMOGRAPHIC);
  - globalCoverageID: global coverage identifier;
  - majorCycleID: major cycle identifier;
  - swathIdentifier: swath identifier (S1, S2 or S3);
  - wrsLongitudeGrid: track number;
  - wrsLatitudeGrid: frame number;

- sections (`Request` and `List_of_Proc_Parameters`) to specify all the optional processing parameters, i.e.:
  - TOI: time coordinates defining the azimuth extent of input L1 data being processed (in [UTC], referred to primary image). If not specified, the entire input L1S products (frame) are processed;
  - primary_image: name of input L1 product to be selected as primary image in the coregistration. If not specified, the primary image is automatically selected by the processor;
  - calibration_primary_image: name of input L1 product to be selected as primary image in the multi-baseline calibration steps. If not specified, the calibration primary image is automatically selected by the processor;
  - range_start_time / range_stop_time: time coordinates defining the slant-range extent of input L1 data being processed (in [s]). If not specified, the whole range extent of the input data is processed;
- fields (`Stdout_Log_Level` and `Stderr_Log_Level`) to set the log level;
- fields (`Number_of_CPU_Cores`, `Amount_of_RAM` and `Disk_Space`) to set the hardware resources allocated for the processing.

For the complete description of Job Order structure and fields refer to [AD2] and for the details on how to fill them to [AD1].

Note that, in case of execution from Docker image, instructions reported in Sec. 5.3.2 shall be followed. For what concern the volumes mounting, all the paths in the Job Order file shall refer to the Docker container file system.

In the box below a sample Stack Processor Job Order is reported.

```
<?xml version="1.0" encoding="UTF-8"?>
<Job_Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
schemaName="BIOMASS CPF-Processor ICD" schemaVersion="String">
  <Processor_Configuration>
    <File_Class>File_Class</File_Class>
    <Processor_Name>STA_P</Processor_Name>
    <Processor_Version>03.30</Processor_Version>
    <Processing_Node>Processing_Node</Processing_Node>
    <List_of_Stdout_Log_Levels>
      <Stdout_Log_Level>INFO</Stdout_Log_Level>
    </List_of_Stdout_Log_Levels>
    <List_of_Stderr_Log_Levels>
      <Stderr_Log_Level>INFO</Stderr_Log_Level>
    </List_of_Stderr_Log_Levels>
    <Intermediate_Output_Enable>true</Intermediate_Output_Enable>
    <Processing_Station>Processing_Station</Processing_Station>
    <Request>
      <TOI>
        <Start>2017-02-25T09:45:35.620307000</Start>
        <Stop>2017-03-03T09:46:00.620307000</Stop>
      </TOI>
      <List_of_Metadata_Parameters>
        <Metadata_Parameter>
          <Name>missionPhase</Name>
          <Value>INTERFEROMETRIC</Value>
        </Metadata_Parameter>
```

```
        <Metadata_Parameter>
          <Name>globalCoverageID</Name>
          <Value>1</Value>
        </Metadata_Parameter>
        <Metadata_Parameter>
          <Name>majorCycleID</Name>
          <Value>1</Value>
        </Metadata_Parameter>
        <Metadata_Parameter>
          <Name>swathIdentifier</Name>
          <Value>S3</Value>
        </Metadata_Parameter>
        <Metadata_Parameter>
          <Name>wrsLongitudeGrid</Name>
          <Value>1</Value>
        </Metadata_Parameter>
        <Metadata_Parameter>
          <Name>wrsLatitudeGrid</Name>
          <Value>1</Value>
        </Metadata_Parameter>
      </List_of_Metadata_Parameters>
    </Request>
  </Processor_Configuration>
  <List_of_Tasks>
    <Task>
      <Task_Name>STA_P</Task_Name>
      <Task_Version>03.30</Task_Version>
      <Number_of_CPU_Cores>7</Number_of_CPU_Cores>
      <Amount_of_RAM>64000</Amount_of_RAM>
      <Disk_Space>55000</Disk_Space>
      <List_of_Proc_Parameters>
        <Proc_Parameter>
          <Name>primary_image</Name>
          <Value></Value>
        </Proc_Parameter>
      </List_of_Proc_Parameters>
      <List_of_Cfg_Files>
        <Cfg_File>
          <Cfg_ID>FNF</Cfg_ID>
          <Cfg_File_Name>/storage/FNF</Cfg_File_Name>
        </Cfg_File>
      </List_of_Cfg_Files>
      <List_of_Inputs>
        <Input>
          <Input_ID>Sx_SCS__1S</Input_ID>
          <Alternative_ID>0001</Alternative_ID>
          <List_of_Selected_Inputs>
            <Selected_Input>
              <File_Type>S1_SCS__1S</File_Type>
              <List_of_File_Names>
                <File_Name>path to L1S product #1</File_Name>
                <File_Name>path to L1S product #2</File_Name>
                <File_Name>path to L1S product #3</File_Name>
              </List_of_File_Names>
            </Selected_Input>
          </List_of_Selected_Inputs>
        </Input>
        <Input>
          <Input_ID>AUX_PPS___</Input_ID>
          <Alternative_ID>0001</Alternative_ID>
```

```
          <List_of_Selected_Inputs>
            <Selected_Input>
              <File_Type>AUX_PPS___</File_Type>
              <List_of_File_Names>
                <File_Name>path to AUX_PPS product</File_Name>
              </List_of_File_Names>
            </Selected_Input>
          </List_of_Selected_Inputs>
        </Input>
      </List_of_Inputs>
      <List_of_Outputs>
        <Output>
          <File_Type>S1_STA__1S</File_Type>
          <File_Name_Pattern>BIO_S1_STA__1S_*</File_Name_Pattern>
          <File_Dir>path to output folder</File_Dir>
          <Baseline>01</Baseline>
        </Output>
        <Output>
          <File_Type>S1_STA__1M</File_Type>
          <File_Name_Pattern>BIO_S1_STA__1M_*</File_Name_Pattern>
          <File_Dir>path to output folder</File_Dir>
          <Baseline>01</Baseline>
        </Output>
      </List_of_Outputs>
      <List_of_Intermediate_Outputs>
        <Intermediate_Output>
          <Intermediate_Output_ID>IntermediateDataDir</Intermediate_Output_ID>
          <Intermediate_Output_File>path to intermediate data
folder</Intermediate_Output_File>
        </Intermediate_Output>
      </List_of_Intermediate_Outputs>
    </Task>
  </List_of_Tasks>
</Job_Order>
```

Sample execution results

The following screenshot describes the results of a successful execution of the Stack Processor when executed with this Job Order in input.

As visible, there are:

- an output directory, where the official output products are stored. This is the folder specified in the Job Order used to trigger the processing;
- a working directory, where the intermediate data are stored. This folder is deleted at the end of the processing if specified in the Job Order (via `Intermediate_Output_Enable` flag).

```
├── output_dir/
│   ├──
BIO_S1_STA__1M_20170101T060307_20170101T060332_I_G03_M03_C03_T000_F
001_05_D4XNN2
│   ├──
BIO_S1_STA__1M_20170104T060310_20170104T060335_I_G03_M03_C03_T000_F
002_05_D4XNNF
│   ├──
BIO_S1_STA__1M_20170107T060314_20170107T060338_I_G03_M03_C03_T000_F
```

```
003_05_D4XNNT
│   ├──
BIO_S1_STA__1S_20170101T060307_20170101T060332_I_G03_M03_C03_T000_F
001_05_D4XNMU
│   ├──
BIO_S1_STA__1S_20170104T060310_20170104T060335_I_G03_M03_C03_T000_F
002_05_D4XNN7
│   └──
BIO_S1_STA__1S_20170107T060314_20170107T060338_I_G03_M03_C03_T000_F
003_05_D4XNNJ
└── working_dir
    ├── aux_pps_previous.xml
    ├── BIO_STA_P_03.32_20250611T131113.log
    ├── job_order_previous.xml
    ├── stack_cal_processor_brk
    │   ├── BPSConf.xml
    │   ├── COREGISTRATION_COMPLETE.json
    │   ├── stackCoregConfig.xml
    │   ├── stackCoregPrimaryConfig.xml
    │   ├── stackInputFile_STA_P0.xml
    │   ├── stackInputFile_STA_P1.xml
    │   ├── stackInputFile_STA_P2.xml
    │   ├── STA_P_01
    │   │   └──
BIO_S1_SCS__1S_20170101T060307_20170101T060332_I_G03_M03_C03_T000_F
001_01_D6445H_PF_CorAz
    │   ├── STA_P_02
    │   │   └──
BIO_S1_SCS__1S_20170104T060310_20170104T060335_I_G03_M03_C03_T000_F
002_01_D6449V_PF_CorAz
    │   └── STA_P_03
    │       └──
BIO_S1_SCS__1S_20170107T060314_20170107T060338_I_G03_M03_C03_T000_F
003_01_D644EF_PF_CorAz
    ├── stack_coreg_processor_brk
    │   ├── BPSConf.xml
    │   ├── COREGISTRATION_COMPLETE.json
    │   ├── stackCoregConfig.xml
    │   ├── stackCoregPrimaryConfig.xml
    │   ├── stackInputFile_STA_P0.xml
    │   ├── stackInputFile_STA_P1.xml
    │   ├── stackInputFile_STA_P2.xml
    │   ├── STA_P_01
    │   │   ├── actualizedCoregistrationParameters.xml
    │   │   ├──
BIO_S1_SCS__1S_20170101T060307_20170101T060332_I_G03_M03_C03_T000_F
001_01_D6445H_PF_Cor
    │   │   ├──
BIO_S1_SCS__1S_20170101T060307_20170101T060332_I_G03_M03_C03_T000_F
001_01_D6445H_PF_CorAz
    │   │   ├──
BIO_S1_SCS__1S_20170101T060307_20170101T060332_I_G03_M03_C03_T000_F
001_01_D6445H_PF_CorRg
    │   │   ├──
BIO_S1_SCS__1S_20170101T060307_20170101T060332_I_G03_M03_C03_T000_F
001_01_D6445H_PF_CSA
    │   │   ├──
BIO_S1_SCS__1S_20170101T060307_20170101T060332_I_G03_M03_C03_T000_F
001_01_D6445H_PF_DSI
    │   │   └──
```

```
BIO_S1_SCS__1S_20170101T060307_20170101T060332_I_G03_M03_C03_T000_F
001_01_D6445H_PF_Kz
    │   ├── STA_P_02
    │   │   ├──
BIO_S1_SCS__1S_20170104T060310_20170104T060335_I_G03_M03_C03_T000_F
002_01_D6449V_PF_Cor
    │   │   ├──
BIO_S1_SCS__1S_20170104T060310_20170104T060335_I_G03_M03_C03_T000_F
002_01_D6449V_PF_CorAz
    │   │   ├──
BIO_S1_SCS__1S_20170104T060310_20170104T060335_I_G03_M03_C03_T000_F
002_01_D6449V_PF_CorRg
    │   │   ├──
BIO_S1_SCS__1S_20170104T060310_20170104T060335_I_G03_M03_C03_T000_F
002_01_D6449V_PF_DAPD
    │   │   ├──
BIO_S1_SCS__1S_20170104T060310_20170104T060335_I_G03_M03_C03_T000_F
002_01_D6449V_PF_DSI
    │   │   └──
BIO_S1_SCS__1S_20170104T060310_20170104T060335_I_G03_M03_C03_T000_F
002_01_D6449V_PF_Kz
    │   └── STA_P_03
    │       ├── actualizedCoregistrationParameters.xml
    │       ├──
BIO_S1_SCS__1S_20170107T060314_20170107T060338_I_G03_M03_C03_T000_F
003_01_D644EF_PF_Cor
    │       ├──
BIO_S1_SCS__1S_20170107T060314_20170107T060338_I_G03_M03_C03_T000_F
003_01_D644EF_PF_CorAz
    │       ├──
BIO_S1_SCS__1S_20170107T060314_20170107T060338_I_G03_M03_C03_T000_F
003_01_D644EF_PF_CorRg
    │       ├──
BIO_S1_SCS__1S_20170107T060314_20170107T060338_I_G03_M03_C03_T000_F
003_01_D644EF_PF_CSA
    │       ├──
BIO_S1_SCS__1S_20170107T060314_20170107T060338_I_G03_M03_C03_T000_F
003_01_D644EF_PF_DSI
    │       └──
BIO_S1_SCS__1S_20170107T060314_20170107T060338_I_G03_M03_C03_T000_F
003_01_D644EF_PF_Kz
    └── stack_pre_processor_brk
        ├── PRE_PROCESSING_COMPLETE.txt
        ├── STA_P_01
        │   └──
BIO_S1_SCS__1S_20170101T060307_20170101T060332_I_G03_M03_C03_T000_F
001_01_D6445H_PF
        ├── STA_P_02
        │   ├──
BIO_S1_SCS__1S_20170104T060310_20170104T060335_I_G03_M03_C03_T000_F
002_01_D6449V_PF
        │   └──
BIO_S1_SCS__1S_20170104T060310_20170104T060335_I_G03_M03_C03_T000_F
002_01_D6449V_PF_XYZ
        └── STA_P_03
            └──
BIO_S1_SCS__1S_20170107T060314_20170107T060338_I_G03_M03_C03_T000_F
003_01_D644EF_PF
```

5.3.3.3. L2a Processor

The L2a Processor (L2A_P) implements one processing chain (or task):

- L2a Processing (L2A_P), aimed at generating a L2a product per type (FD, FH and GN) starting from a set of L1c products (3 for INT phase, 7 for TOM phase, by default) and (optionally) a L2a FD product of previous global cycle.

In addition to this, L2a Processor requires in input for its execution also a set of auxiliary files and internal resources.

The complete list of inputs, and of corresponding outputs, is reported in the following tables, indicating also the criteria to be used for selecting each input and the destinations of each output inside the overall BPS processing chain.

Please refer to [AD1] and [AD2] for the complete description of each field of these tables (a summary is reported also in Sec. 9) and to [RD4] for the complete description of each auxiliary file and internal resource.

L2a Processing (L2A_P) task: inputs and outputs

Product Type	Description	Card.	Origin	Instances	Start / Stop Offset	Overlap / Min Overlap	Metadata Names	Max. # of Files	Ordering
Sx_STA__1S	Stripmap Level-1 – Coregistered stack product	2..A	External (STA_P)	Single	-	-	Mission Phase Global Cycle Identifier Major Cycle Identifier Swath Identifier Track Number Frame Number *	8	Latest
FP_FD__L2A	L2a Forest Disturbance product	0, 1	External (L2A_P)	Single	-	-	Mission Phase Global Cycle Identifier Major Cycle Identifier Swath Identifier Track Number Frame Number *	1	Latest
AUX_PP2_2A	Auxiliary L2a Processing	1	External	Single	0 / 0	Yes / 100	/	1	MaxTemporalOverlap, ClosestStart

	Parameters								

Product Type	Description	Card.	Dest.	Notes
FP_FD__L2A	L2a Forest Disturbance product	0, 1	External (L2A_P) External (L2B_FD_P)	Collected by PF and used as input by L2a processor L2A_P task and by L2b processor L2B_FD_P task Generated by default (in INT phase, while optional in TOM phase)
FP_FH__L2A	L2a Forest Height product	0, 1	External (L2B_FH_P)	Collected by PF and used as input by L2b processor L2B_FH_P task Generated by default
FP_GN__L2A	L2a Ground Cancelled product	0, 1	External (L2B_AGB_P)	Collected by PF and used as input by L2b processor L2B_AGB_P task Generated by default

Job Order preparation

To trigger a L2a Processor execution a dedicated Job Order has to be filled. Here below are reported the list of sections and fields, with the instructions on how to fill them:

- `Processor_Name`, that shall be L2A_P;
- `Task_Name`, that shall be L2A_P (only one task for each Job Order);
- `Processor_Version` / `Task_Version`, that shall contain the processor version (in the form MM.mp, with M for major, m for minor and p for patch);
- a section (`List_of_Inputs`) where to specify all the input products and auxiliary files paths, selected according to the rules listed in the tables above, i.e.:
  - Sx_STA__1S: Stripmap mode L1 Coregistered Stack Standard (L1cS) products (up to 8);
  - FP_FD__L2A: L2a Forest Disturbance product (optional);
  - AUX_PP2_2A: Auxiliary L2a Processing Parameters product, containing the processing parameters for each L2A_P processing chain step (one section of the XML file for each step, each parameter containing dedicated attributes in case it is possible to set it in a swath- or polarization-dependent way). This is the input the user may want to update in case he needs to modify something related to processing configuration. Full detailed description of this file is reported in [RD3];
- a section (`List_of_Cfg_Files`) where to specify all the internal resources path, i.e.:
  - Forest/Non-Forest mask (FNF) database;

  Note that in this field just the path to the generic folder containing the internal resource shall be specified. Then it is up to the processor to select the right one to use during processing, basing on information contained in auxiliary files (see [RD3] and Sec. 4.4.2.1);
- a section (`List_of_Outputs`) where to specify all the output products paths, i.e.:

    - FP_FD__L2A L2a: Forest Disturbance product (optional, generated by default in INT phase);
    - FP_FH__L2A L2a: Forest Height product (optional, generated by default);
    - FP_GN__L2A L2a: Ground Cancelled product (optional, generated by default);
- a section (`List_of_Metadata_Parameters`) where to specify the criteria used to select the input L1 products to be used for L2 processing:
    - missionPhase: mission phase (INTERFEROMETRIC or TOMOGRAPHIC);
    - globalCoverageID: global coverage identifier;
    - majorCycleID: major cycle identifier;
    - swathIdentifier: swath identifier (S1, S2 or S3);
    - wrsLongitudeGrid: track number;
    - wrsLatitudeGrid: frame number;
- fields (`Stdout_Log_Level` and `Stderr_Log_Level`) to set the log level;
- fields (`Number_of_CPU_Cores`, `Amount_of_RAM` and `Disk_Space`) to set the hardware resources allocated for the processing.

For the complete description of Job Order structure and fields refer to [AD2] and for the details on how to fill them to [AD1].

Note that, in case of execution from Docker image, instructions reported in Sec. 5.3.2 shall be followed. For what concern the volumes mounting, all the paths in the Job Order file shall refer to the Docker container file system.

In the box below a sample L2a Processor Job Order is reported.

```
<?xml version="1.0" encoding="UTF-8"?>
<Job_Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
schemaName="BIOMASS CPF-Processor ICD" schemaVersion="String">
  <Processor_Configuration>
    <File_Class>File_Class</File_Class>
    <Processor_Name>L2A_P</Processor_Name>
    <Processor_Version>03.30</Processor_Version>
    <Processing_Node>Processing_Node</Processing_Node>
    <List_of_Stdout_Log_Levels>
      <Stdout_Log_Level>INFO</Stdout_Log_Level>
    </List_of_Stdout_Log_Levels>
    <List_of_Stderr_Log_Levels>
      <Stderr_Log_Level>INFO</Stderr_Log_Level>
    </List_of_Stderr_Log_Levels>
    <Intermediate_Output_Enable>false</Intermediate_Output_Enable>
    <Processing_Station>Processing_Station</Processing_Station>
    <Request>
      <List_of_Metadata_Parameters />
    </Request>
  </Processor_Configuration>
  <List_of_Tasks>
    <Task>
      <Task_Name>L2A_P</Task_Name>
      <Task_Version>03.30</Task_Version>
      <Number_of_CPU_Cores>8</Number_of_CPU_Cores>
      <Amount_of_RAM>30720</Amount_of_RAM>
      <Disk_Space>12288</Disk_Space>
```

```
      <List_of_Proc_Parameters />
      <List_of_Cfg_Files>
        <Cfg_File>
          <Cfg_ID>FNF</Cfg_ID>
          <Cfg_File_Name>/storage/FNF</Cfg_File_Name>
        </Cfg_File>
      </List_of_Cfg_Files>
      <List_of_Inputs>
        <Input>
          <Input_ID>Sx_STA__1S</Input_ID>
          <Alternative_ID>0001</Alternative_ID>
          <List_of_Selected_Inputs>
            <Selected_Input>
              <File_Type>S1_STA__1S</File_Type>
              <List_of_File_Names>
                <File_Name>path to L1cS product #1</File_Name>
                <File_Name>path to L1cS product #2</File_Name>
                <File_Name>path to L1cS product #3</File_Name>
              </List_of_File_Names>
            </Selected_Input>
          </List_of_Selected_Inputs>
        </Input>
        <Input>
          <Input_ID>AUX_PP2_2A</Input_ID>
          <Alternative_ID>0001</Alternative_ID>
          <List_of_Selected_Inputs>
            <Selected_Input>
              <File_Type>AUX_PP2_2A</File_Type>
              <List_of_File_Names>
                <File_Name>path to AUX_PP2_2A product</File_Name>
              </List_of_File_Names>
            </Selected_Input>
          </List_of_Selected_Inputs>
        </Input>
      </List_of_Inputs>
      <List_of_Outputs>
        <Output>
          <File_Type>FP_FD__L2A</File_Type>
          <File_Name_Pattern>BIO_FP_FD__L2A_*</File_Name_Pattern>
          <File_Dir>path to output folder</File_Dir>
          <Baseline>02</Baseline>
        </Output>
        <Output>
          <File_Type>FP_FH__L2A</File_Type>
          <File_Name_Pattern>BIO_FP_FH__L2A_*</File_Name_Pattern>
          <File_Dir>path to output folder</File_Dir>
          <Baseline>02</Baseline>
        </Output>
        <Output>
          <File_Type>FP_GN__L2A</File_Type>
          <File_Name_Pattern>BIO_FP_GN__L2A_*</File_Name_Pattern>
          <File_Dir>path to output folder</File_Dir>
          <Baseline>02</Baseline>
        </Output>
      </List_of_Outputs>
      <List_of_Intermediate_Outputs />
    </Task>
  </List_of_Tasks>
</Job_Order>
```

Sample execution results

L2a Processor does not generate intermediate data. Output products are stored in the path specified in the Job Order and log file is stored in the same path of the Job Order file.

5.3.3.4. L2b Processor

The L2b Processor (L2B_P) implements three processing chains (or task):

- L2b FD Processing (L2B_FD_P), aimed at generating a L2b FD product starting from a set of L2a FD products acquired over the same tile (only for INT phase);
- L2b FH Processing (L2B_FH_P), aimed at generating a L2b FH product starting from a set of L2a FH products acquired over the same tile and (only for INT phase) the corresponding L2b FD product (for TOM phase, FD optional input is replaced by an internal map);
- L2b AGB Processing (L2B_AGB_P), aimed at generating a set of L2b AGB products from a set of L2a AGB products acquired over the same tile and over surrounding ones, (at the second iteration only) a set of L2b AGB products acquired over surrounding tiles and (only for INT phase) the corresponding set of L2b FD products (for TOM phase, FD optional input is replaced by an internal map).

In addition to this, L2b Processor requires in input for its execution also a set of auxiliary files and internal resources.

The complete list of inputs, and of corresponding outputs, is reported in the tables in the following sub-sections, indicating also the criteria to be used for selecting each input and the destinations of each output inside the overall BPS processing chain.

Please refer to [AD1] and [AD2] for the complete description of each field of these tables (a summary is reported also in Sec. 9) and to [RD4] for the complete description of each auxiliary file and internal resource.

5.3.3.4.1. L2b FD Processor

L2b FD Processing (L2B_FD_P) task: inputs and outputs

Product Type	Description	Card.	Origin	Instances	Start / Stop Offset	Overlap / Min Overlap	Metadata Names	Max. # of Files	Ordering
FP_FD__L2A	L2a Forest Disturbance product	TD	External (L2A_P)	Single	0 / 0	Yes / 100	Mission Phase Global Cycle Identifier Tile Identifier *	N.D.	Latest
AUX_PP2_FD	Auxiliary L2b FD Processing Parameters	1	External	Single	0 / 0	Yes / 100	/	1	MaxTemporalOverlap, ClosestStart

Product Type	Description	Card.	Dest.	Notes
FP_FD__L2B	L2b Forest Disturbance product	1	External (L2B_FH_P, L2B_AGB_P) External (L3_P)	Collected by PF and used as input by L2b processor L2B_FH_P and L2B_AGB_P tasks and by L3 processor L3_P task

Job Order preparation

To trigger a L2b Processor execution a dedicated Job Order has to be filled. Here below are reported the list of sections and fields, with the instructions on how to fill them:

- `Processor_Name`, that shall be L2B_P;
- `Task_Name`, that shall be L2B_FD_P (only one task for each Job Order);
- `Processor_Version` / `Task_Version`, that shall contain the processor version (in the form MM.mp, with M for major, m for minor and p for patch);
- a section (`List_of_Inputs`) where to specify all the input products and auxiliary files paths, selected according to the rules listed in the tables above, i.e.:
  - FP_FD__L2A: L2a Forest Disturbance products;
  - AUX_PP2_FD: Auxiliary L2b FD Processing Parameters product, containing the processing parameters for each L2B_FD_P processing chain step (one section of the XML file for each step, each parameter containing dedicated attributes in case it is possible to set it in a swath- or polarization-dependent way). This is the input the user may want to update in case he needs to modify something related to processing configuration. Full detailed description of this file is reported in [RD3];
- a section (`List_of_Outputs`) where to specify all the output products paths, i.e.:
  - FP_FD__L2B: L2b Forest Disturbance product;
- a section (`List_of_Metadata_Parameters`) where to specify the criteria used to select the input L1 products to be used for L2 processing:
  - missionPhase: mission phase (INTERFEROMETRIC or TOMOGRAPHIC);
  - globalCoverageID: global coverage identifier;
  - tileID: tile identifier;
- sections (`Request` and `List_of_Proc_Parameters`) to specify all the optional processing parameters, i.e.:
  - tile_id: this parameter is used to pass to the L2b Processor the absolute tile index of the tile to be generated;
- fields (`Stdout_Log_Level` and `Stderr_Log_Level`) to set the log level;
- fields (`Number_of_CPU_Cores`, `Amount_of_RAM` and `Disk_Space`) to set the hardware resources allocated for the processing.

For the complete description of Job Order structure and fields refer to [AD2] and for the details on how to fill them to [AD1].

Note that, in case of execution from Docker image, instructions reported in Sec. 5.3.2 shall be followed. For what concern the volumes mounting, all the paths in the Job Order file shall refer to the Docker container file system.
In the box below a sample L2b FD Processor Job Order is reported.

```
<?xml version="1.0" encoding="UTF-8"?>
<Job_Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
schemaName="BIOMASS CPF-Processor ICD" schemaVersion="String">
  <Processor_Configuration>
    <File_Class>File_Class</File_Class>
    <Processor_Name>L2B_FD_P</Processor_Name>
    <Processor_Version>03.30</Processor_Version>
    <Processing_Node>Processing_Node</Processing_Node>
    <List_of_Stdout_Log_Levels>
      <Stdout_Log_Level>INFO</Stdout_Log_Level>
    </List_of_Stdout_Log_Levels>
    <List_of_Stderr_Log_Levels>
      <Stderr_Log_Level>INFO</Stderr_Log_Level>
    </List_of_Stderr_Log_Levels>
    <Intermediate_Output_Enable>false</Intermediate_Output_Enable>
    <Processing_Station>Processing_Station</Processing_Station>
    <Request>
      <List_of_Metadata_Parameters />
    </Request>
  </Processor_Configuration>
  <List_of_Tasks>
    <Task>
      <Task_Name>L2B_FD_P</Task_Name>
      <Task_Version>03.30</Task_Version>
      <Number_of_CPU_Cores>4</Number_of_CPU_Cores>
      <Amount_of_RAM>9216</Amount_of_RAM>
      <Disk_Space>2048</Disk_Space>
      <List_of_Proc_Parameters>
        <Proc_Parameter>
          <Name>tile_id</Name>
          <Value>S35W056</Value>
        </Proc_Parameter>
      </List_of_Proc_Parameters>
      <List_of_Cfg_Files />
      <List_of_Inputs>
        <Input>
          <Input_ID>FP_FD__L2A</Input_ID>
          <Alternative_ID>0001</Alternative_ID>
          <List_of_Selected_Inputs>
            <Selected_Input>
              <File_Type>FP_FD__L2A</File_Type>
              <List_of_File_Names>
                <File_Name>path to L2A FD product</File_Name>
              </List_of_File_Names>
            </Selected_Input>
          </List_of_Selected_Inputs>
        </Input>
        <Input>
          <Input_ID>AUX_PP2_FD</Input_ID>
          <Alternative_ID>0001</Alternative_ID>
          <List_of_Selected_Inputs>
            <Selected_Input>
              <File_Type>AUX_PP2_FD</File_Type>
```

```
            <List_of_File_Names>
              <File_Name>path to AUX_PP2_FD product</File_Name>
            </List_of_File_Names>
          </Selected_Input>
        </List_of_Selected_Inputs>
      </Input>
    </List_of_Inputs>
    <List_of_Outputs>
      <Output>
        <File_Type>FP_FD__L2B</File_Type>
        <File_Name_Pattern>BIO_FP_FD__L2B_*</File_Name_Pattern>
        <File_Dir>path to output folder</File_Dir>
        <Baseline>02</Baseline>
      </Output>
    </List_of_Outputs>
    <List_of_Intermediate_Outputs />
  </Task>
 </List_of_Tasks>
</Job_Order>
```

Sample execution results

L2b FD Processor does not generate intermediate data. Output products are stored in the path specified in the Job Order and log file is stored in the same path of the Job Order file.

5.3.3.4.2. L2b FH Processor

L2b FH Processing (L2B_FH_P) task: inputs and outputs

Product Type	Description	Card.	Origin	Instances	Start / Stop Offset	Overlap / Min Overlap	Metadata Names	Max. # of Files	Ordering
FP_FH__L2A	L2a Forest Height product	TD	External (L2A_P)	Single	0 / 0	Yes / 100	Mission Phase Global Cycle Identifier Tile Identifier *	N.D.	Latest
FP_FD__L2B	L2b Forest Disturbance product	0, 1	External (L2B_FD_P)	Single	0 / 0	Yes / 100	Mission Phase Global Cycle Identifier Tile Identifier *	1	Latest
AUX_PP2_FH	Auxiliary L2b FH Processing Parameters	1	External	Single	0 / 0	Yes / 100	/	1	MaxTemporalOverlap, ClosestStart

Product Type	Description	Card.	Dest.	Notes

FP_FH__L2B	L2b Forest Height product	1	External (L2B_AGB_P) External (L3_P)	Collected by PF and used as input by L2b processor L2B_AGB_P task and by L3 processor L3_P task

Job Order preparation

To trigger a L2b Processor execution a dedicated Job Order has to be filled. Here below are reported the list of sections and fields, with the instructions on how to fill them:

- `Processor_Name`, that shall be L2B_P;
- `Task_Name`, that shall be L2B_FH_P (only one task for each Job Order);
- `Processor_Version` / `Task_Version`, that shall contain the processor version (in the form MM.mp, with M for major, m for minor and p for patch);
- a section (`List_of_Inputs`) where to specify all the input products and auxiliary files paths, selected according to the rules listed in the tables above, i.e.:
  - FP_FH__L2A: L2a Forest Height products;
  - FP_FD__L2B: L2b Forest Disturbance product (optional);
  - AUX_PP2_FH: Auxiliary L2b FH Processing Parameters product, containing the processing parameters for each L2B_FH_P processing chain step (one section of the XML file for each step, each parameter containing dedicated attributes in case it is possible to set it in a swath- or polarization-dependent way). This is the input the user may want to update in case he needs to modify something related to processing configuration. Full detailed description of this file is reported in [RD3];
- a section (`List_of_Outputs`) where to specify all the output products paths, i.e.:
  - FP_FH__L2B: L2b Forest Height product;
- a section (`List_of_Metadata_Parameters`) where to specify the criteria used to select the input L1 products to be used for L2 processing:
  - missionPhase: mission phase (INTERFEROMETRIC or TOMOGRAPHIC);
  - globalCoverageID: global coverage identifier;
  - tileID: tile identifier;
- sections (`Request` and `List_of_Proc_Parameters`) to specify all the optional processing parameters, i.e.:
  - tile_id: this parameter is used to pass to the L2b Processor the absolute tile index of the tile to be generated;
- fields (`Stdout_Log_Level` and `Stderr_Log_Level`) to set the log level;
- fields (`Number_of_CPU_Cores`, `Amount_of_RAM` and `Disk_Space`) to set the hardware resources allocated for the processing.

For the complete description of Job Order structure and fields refer to [AD2] and for the details on how to fill them to [AD1].

Note that, in case of execution from Docker image, instructions reported in Sec. 5.3.2 shall be followed. For what concern the volumes mounting, all the paths in the Job Order file shall refer to the Docker container file system.

In the box below a sample L2b FH Processor Job Order is reported.

```
<?xml version="1.0" encoding="UTF-8"?>
<Job_Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
schemaName="BIOMASS CPF-Processor ICD" schemaVersion="String">
  <Processor_Configuration>
    <File_Class>File_Class</File_Class>
    <Processor_Name>L2B_FH_P</Processor_Name>
    <Processor_Version>03.30</Processor_Version>
    <Processing_Node>Processing_Node</Processing_Node>
    <List_of_Stdout_Log_Levels>
      <Stdout_Log_Level>INFO</Stdout_Log_Level>
    </List_of_Stdout_Log_Levels>
    <List_of_Stderr_Log_Levels>
      <Stderr_Log_Level>INFO</Stderr_Log_Level>
    </List_of_Stderr_Log_Levels>
    <Intermediate_Output_Enable>false</Intermediate_Output_Enable>
    <Processing_Station>Processing_Station</Processing_Station>
    <Request>
      <List_of_Metadata_Parameters />
    </Request>
  </Processor_Configuration>
  <List_of_Tasks>
    <Task>
      <Task_Name>L2B_FH_P</Task_Name>
      <Task_Version>03.30</Task_Version>
      <Number_of_CPU_Cores>4</Number_of_CPU_Cores>
      <Amount_of_RAM>9216</Amount_of_RAM>
      <Disk_Space>2048</Disk_Space>
      <List_of_Proc_Parameters>
        <Proc_Parameter>
          <Name>tile_id</Name>
          <Value>S35W056</Value>
        </Proc_Parameter>
      </List_of_Proc_Parameters>
      <List_of_Cfg_Files />
      <List_of_Inputs>
        <Input>
          <Input_ID>FP_FH__L2A</Input_ID>
          <Alternative_ID>0001</Alternative_ID>
          <List_of_Selected_Inputs>
            <Selected_Input>
              <File_Type>FP_FH__L2A</File_Type>
              <List_of_File_Names>
                <File_Name>path to L2A FH product</File_Name>
              </List_of_File_Names>
            </Selected_Input>
          </List_of_Selected_Inputs>
        </Input>
        <Input>
          <Input_ID>AUX_PP2_FH</Input_ID>
          <Alternative_ID>0001</Alternative_ID>
          <List_of_Selected_Inputs>
            <Selected_Input>
              <File_Type>AUX_PP2_FH</File_Type>
              <List_of_File_Names>
                <File_Name>path to AUX_PP2_FH product</File_Name>
              </List_of_File_Names>
            </Selected_Input>
```

```
          </List_of_Selected_Inputs>
        </Input>
      </List_of_Inputs>
      <List_of_Outputs>
        <Output>
          <File_Type>FP_FH__L2B</File_Type>
          <File_Name_Pattern>BIO_FP_FH__L2B_*</File_Name_Pattern>
          <File_Dir>path to output folder</File_Dir>
          <Baseline>02</Baseline>
        </Output>
      </List_of_Outputs>
      <List_of_Intermediate_Outputs />
    </Task>
  </List_of_Tasks>
</Job_Order>
```

Sample execution results

L2b FH Processor does not generate intermediate data. Output products are stored in the path specified in the Job Order and log file is stored in the same path of the Job Order file.

5.3.3.4.3. L2b AGB Processor

L2b AGB Processing (L2B_AGB_P) task: inputs and outputs

Product Type	Description	Card.	Origin	Instances	Start / Stop Offset	Overlap / Min Overlap	Metadata Names	Max. # of Files	Ordering
FP_GN__L2A	L2a Ground Cancelled product	TD	External (L2A_P)	Single	0 / 0	Yes / 100	Mission Phase Global Cycle Identifier Tile Identifier *	N.D.	Latest
FP_FD__L2B	L2b Forest Disturbance product	0, TT	External (L2B_FD_P)	Single	0 / 0	Yes / 100	Mission Phase Global Cycle Identifier Tile Identifier *	N.D.	Latest
FP_AGB_L2B	L2b Above Ground Biomass product	0, TT	External (L2B_AGB_P)	Single	0 / 0	Yes / 100	Mission Phase Global Cycle Identifier Tile Identifier *	N.D.	Latest
AUX_PP2_AB	Auxiliary L2b AB Processing Parameters	1	External	Single	0 / 0	Yes / 100	/	1	MaxTemporalOverlap, ClosestStart

Product Type	Description	Card.	Dest.	Notes
FP_AGB_L2B	L2b Above Ground Biomass product	TI	External (L2B_AGB_P) External (L3_P)	Collected by PF and used as input by L2b processor L2B_AGB_P task and by L3 processor L3_P task

Job Order preparation

To trigger a L2b Processor execution a dedicated Job Order has to be filled. Here below are reported the list of sections and fields, with the instructions on how to fill them:

- `Processor_Name`, that shall be L2B_P;
- `Task_Name`, that shall be L2B_FH_P (only one task for each Job Order);
- `Processor_Version` / `Task_Version`, that shall contain the processor version (in the form MM.mp, with M for major, m for minor and p for patch);
- a section (`List_of_Inputs`) where to specify all the input products and auxiliary files paths, selected according to the rules listed in the tables above, i.e.:
  - FP_GN__L2A: L2a Ground Cancelled products;
  - FP_FD__L2B: L2b Forest Disturbance products (optional);
  - FP_AGB_L2B: L2b Above Ground Biomass products (optional);
  - AUX_PP2_AB: Auxiliary L2b AB Processing Parameters product, containing the processing parameters for each L2B_AGB_P processing chain step (one section of the XML file for each step, each parameter containing dedicated attributes in case it is possible to set it in a swath- or polarization-dependent way). This is the input the user may want to update in case he needs to modify something related to processing configuration. Full detailed description of this file is reported in [RD3];
- a section (`List_of_Cfg_Files`) where to specify all the internal resources path, i.e.:
  - Land Cover Map (LCM) database;
  - Reference AGB (CAL_AB) database;

  Note that in this field just the path to the generic folder containing the internal resource shall be specified. Then it is up to the processor to select the right one to use during processing, basing on information contained in auxiliary files (see [RD3] and Sec. 4.4.2.1);
- a section (`List_of_Outputs`) where to specify all the output products paths, i.e.:
  - FP_AGB_L2B: L2b Above Ground Biomass product;
- a section (`List_of_Metadata_Parameters`) where to specify the criteria used to select the input L1 products to be used for L2 processing:
  - missionPhase: mission phase (INTERFEROMETRIC or TOMOGRAPHIC);
  - globalCoverageID: global coverage identifier;
  - tileID: tile identifier;

- sections (`Request` and `List_of_Proc_Parameters`) to specify all the optional processing parameters, i.e.:
  - central_tile_id: this parameter is used to pass to the L2b Processor the absolute tile index of the tile to be generated (central tile for both GB estimation area and AGB neighbourhood);
- fields (`Stdout_Log_Level` and `Stderr_Log_Level`) to set the log level;
- fields (`Number_of_CPU_Cores`, `Amount_of_RAM` and `Disk_Space`) to set the hardware resources allocated for the processing.

For the complete description of Job Order structure and fields refer to [AD2] and for the details on how to fill them to [AD1].

Note that, in case of execution from Docker image, instructions reported in Sec. 5.3.2 shall be followed. For what concern the volumes mounting, all the paths in the Job Order file shall refer to the Docker container file system.

In the box below a sample L2b AGB Processor Job Order is reported.

```
<?xml version="1.0" encoding="UTF-8"?>
<Job_Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
schemaName="BIOMASS CPF-Processor ICD" schemaVersion="String">
  <Processor_Configuration>
    <File_Class>File_Class</File_Class>
    <Processor_Name>L2B_AGB_P_withoutFD_withoutAGB</Processor_Name>
    <Processor_Version>03.30</Processor_Version>
    <Processing_Node>Processing_Node</Processing_Node>
    <List_of_Stdout_Log_Levels>
      <Stdout_Log_Level>INFO</Stdout_Log_Level>
    </List_of_Stdout_Log_Levels>
    <List_of_Stderr_Log_Levels>
      <Stderr_Log_Level>INFO</Stderr_Log_Level>
    </List_of_Stderr_Log_Levels>
    <Intermediate_Output_Enable>false</Intermediate_Output_Enable>
    <Processing_Station>Processing_Station</Processing_Station>
    <Request>
      <List_of_Metadata_Parameters />
    </Request>
  </Processor_Configuration>
  <List_of_Tasks>
    <Task>
      <Task_Name>L2B_AGB_P</Task_Name>
      <Task_Version>03.30</Task_Version>
      <Number_of_CPU_Cores>4</Number_of_CPU_Cores>
      <Amount_of_RAM>65536</Amount_of_RAM>
      <Disk_Space>20480</Disk_Space>
      <List_of_Proc_Parameters>
        <Proc_Parameter>
          <Name>central_tile_id</Name>
          <Value>S35W056</Value>
        </Proc_Parameter>
      </List_of_Proc_Parameters>
      <List_of_Cfg_Files>
        <Cfg_File>
          <Cfg_ID>LCM</Cfg_ID>
          <Cfg_File_Name>/storage/projects/biomass_bps/test_plan/tds/TDS-BPS-
L2B-
```

```
009/aux_files/LCM/BIO_AUX_L2_LCM_20200101T000000_20201231T000000_AFRS00</Cfg_F
ile_Name>
        </Cfg_File>
        <Cfg_File>
          <Cfg_ID>CAL_AB</Cfg_ID>
          <Cfg_File_Name>/storage/projects/biomass_bps/test_plan/tds/TDS-BPS-
L2B-
009/aux_files/CAL_AB/BIO_AUX_CAL_AB_20190417T000000_20230316T000000_AFRS01</Cf
g_File_Name>
        </Cfg_File>
      </List_of_Cfg_Files>
      <List_of_Inputs>
        <Input>
          <Input_ID>FP_GN__L2A</Input_ID>
          <Alternative_ID>0001</Alternative_ID>
          <List_of_Selected_Inputs>
            <Selected_Input>
              <File_Type>FP_GN__L2A</File_Type>
              <List_of_File_Names>
                <File_Name>path to L2A GN product</File_Name>
              </List_of_File_Names>
            </Selected_Input>
          </List_of_Selected_Inputs>
        </Input>
        <Input>
          <Input_ID>AUX_PP2_AB</Input_ID>
          <Alternative_ID>0001</Alternative_ID>
          <List_of_Selected_Inputs>
            <Selected_Input>
              <File_Type>AUX_PP2_AB</File_Type>
              <List_of_File_Names>
                <File_Name>path to AUX_PP2_AB product</File_Name>
              </List_of_File_Names>
            </Selected_Input>
          </List_of_Selected_Inputs>
        </Input>
      </List_of_Inputs>
      <List_of_Outputs>
        <Output>
          <File_Type>FP_AGB_L2B</File_Type>
          <File_Name_Pattern>BIO_FP_AGB_L2B_*</File_Name_Pattern>
          <File_Dir>path to output folder</File_Dir>
          <Baseline>02</Baseline>
        </Output>
      </List_of_Outputs>
      <List_of_Intermediate_Outputs />
    </Task>
  </List_of_Tasks>
</Job_Order>
```

Sample execution results

L2b AGB Processor does not generate intermediate data. Output products are stored in the path specified in the Job Order and log file is stored in the same path of the Job Order file.

5.3.3.5. L3 Processor

L3 Processing (L3_P) task: inputs and outputs

Product Type	Description	Card.	Origin	Instances	Start / Stop Offset	Overlap / Min Overlap	Metadata Names	Max. # of Files	Ordering
FP_FH__L2B	L2b Forest Height product	TE	External (L2B_FH_P)	Single	0 / 0	Yes / 100	Mission Phase Global Cycle Identifier Tile index *	N.D.	Latest
FP_AGB_L2B	L2b Above Ground Biomass product	TE	External (L2B_AGB_P)	Single	0 / 0	Yes / 100	Mission Phase Global Cycle Identifier Tile index *	N.D.	Latest
FP_FD_L2B	L2b Forest Disturbance product	TE	External (L2B_AGB_P)	Single	0 / 0	Yes / 100	Mission Phase Global Cycle Identifier Tile index *	N.D.	Latest
AUX_PP3___	Auxiliary L3 Processing Parameters	1	External	Single	0 / 0	Yes / 100	/	1	MaxTemporalOverlap, ClosestStart

Product Type	Description	Card.	Dest.	Notes
FP_FH__L3T	L3 Forest Height product	1	External	Collected by PF, but not used by other BPS tasks
FP_AGB_L3T	L3 Above Ground Biomass product	1	External	Collected by PF, but not used by other BPS tasks

Job Order preparation

To trigger a L3 Processor execution a dedicated Job Order has to be filled. Here below are reported the list of sections and fields, with the instructions on how to fill them:

- `Processor_Name`, that shall be L3_P;
- `Task_Name`, that shall be L3_P (only one task for each Job Order);
- `Processor_Version` / `Task_Version`, that shall contain the processor version (in the form MM.mp, with M for major, m for minor and p for patch);
- a section (`List_of_Inputs`) where to specify all the input products and auxiliary files paths, selected according to the rules listed in the tables above, i.e.:
  - FP_FH__L2B: L2b Forest Height products;
  - FP_FD__L2B: L2b Forest Disturbance products (optional for first Global Coverage)

  - FP_AGB_L2B: L2b Above Ground Biomass products ;
  - AUX_PP3___: Auxiliary L3 Processing Parameters product, containing the processing parameters for L3 processing. This is the input the user may want to update in case he needs to modify something related to processing configuration. Full detailed description of this file is reported in [RD3];
- a section (`List_of_Cfg_Files`) where to specify all the internal resources path, i.e.:
  - Land Cover Map (LCM) database;
- a section (`List_of_Outputs`) where to specify all the output products paths, i.e.:
  - FP_AGB_L3: L3 Above Ground Biomass product;
  - FP_FH__L3: L3 Forest Height Biomass product;
- fields (`Stdout_Log_Level` and `Stderr_Log_Level`) to set the log level;
- fields (`Number_of_CPU_Cores`, `Amount_of_RAM` and `Disk_Space`) to set the hardware resources allocated for the processing.

For the complete description of Job Order structure and fields refer to [AD2] and for the details on how to fill them to [AD1].

Note that, in case of execution from Docker image, instructions reported in Sec. 5.3.2 shall be followed. For what concern the volumes mounting, all the paths in the Job Order file shall refer to the Docker container file system.

In the box below a sample L2b AGB Processor Job Order is reported.

```
<?xml version="1.0" encoding="UTF-8"?>
<Job_Order                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
schemaName="BIOMASS       CPF-Processor       ICD"       schemaVersion="String"
schemaRef="http://www.altova.com/">
    <Processor_Configuration>
        <File_Class>XXX</File_Class>
        <Processor_Name>L3_P</Processor_Name>
        <Processor_Version>03.01</Processor_Version>
        <Processing_Node>XXX</Processing_Node>
        <List_of_Stdout_Log_Levels>
            <Stdout_Log_Level>INFO</Stdout_Log_Level>
        </List_of_Stdout_Log_Levels>
        <List_of_Stderr_Log_Levels>
            <Stderr_Log_Level>INFO</Stderr_Log_Level>
        </List_of_Stderr_Log_Levels>
        <Intermediate_Output_Enable>false</Intermediate_Output_Enable>
        <Processing_Station>XXX</Processing_Station>
        <Request>
            <List_of_Metadata_Parameters/>
        </Request>
    </Processor_Configuration>
    <List_of_Tasks>
        <Task>
            <Task_Name>L3_P</Task_Name>
            <Task_Version>03.01</Task_Version>
            <Number_of_CPU_Cores>4</Number_of_CPU_Cores>
            <Amount_of_RAM>6144</Amount_of_RAM>
            <Disk_Space>2048</Disk_Space>
            <List_of_Cfg_Files>
                <Cfg_File>
```

```
                    <Cfg_ID>LCM</Cfg_ID>
                     <Cfg_File_Name>/data/BTK/L3_Delivery/CONF/BIO_AUX_L2_LCM_2
0200101T000000_20201231T000000_AFRS00</Cfg_File_Name>
                </Cfg_File>
            </List_of_Cfg_Files>
            <List_of_Inputs>
                <Input>
                    <Input_ID>FP_AGB_L2B</Input_ID>
                    <Alternative_ID>FP_AGB_L2B</Alternative_ID>
                    <List_of_Selected_Inputs>
                        <Selected_Input>
                            <File_Type>FP_AGB_L2B</File_Type>
                            <List_of_File_Names>
                                <File_Name>/data/BTK/L3_Delivery/TDS/TDS-BPS-
L3-001/l2b_agb/BIO_FP_AGB_L2B_I_G01_TS02E023_B001_02_D299IA</File_Name>
                                <File_Name>/data/BTK/L3_Delivery/TDS/TDS-BPS-
L3-001/l2b_agb/BIO_FP_AGB_L2B_I_G02_TS02E023_B001_02_D299IA</File_Name>
                                <File_Name>/data/BTK/L3_Delivery/TDS/TDS-BPS-
L3-001/l2b_agb/BIO_FP_AGB_L2B_I_G03_TS02E023_B001_02_D299IA</File_Name>
                                <File_Name>/data/BTK/L3_Delivery/TDS/TDS-BPS-
L3-001/l2b_agb/BIO_FP_AGB_L2B_I_G04_TS02E023_B001_02_D299IA</File_Name>
                            </List_of_File_Names>
                        </Selected_Input>
                    </List_of_Selected_Inputs>
                </Input>
                <Input>
                    <Input_ID>FP_FD__L2B</Input_ID>
                    <Alternative_ID>FP_FD__L2B</Alternative_ID>
                    <List_of_Selected_Inputs>
                        <Selected_Input>
                            <File_Type>FP_FD__L2B</File_Type>
                            <List_of_File_Names>
                                <File_Name>/data/BTK/L3_Delivery/TDS/TDS-BPS-
L3-001/l2b_fd/BIO_FP_FD__L2B_I_G01_TS02E023_B001_02_D299R0</File_Name>
                                <File_Name>/data/BTK/L3_Delivery/TDS/TDS-BPS-
L3-001/l2b_fd/BIO_FP_FD__L2B_I_G02_TS02E023_B001_02_D299R0</File_Name>
                                <File_Name>/data/BTK/L3_Delivery/TDS/TDS-BPS-
L3-001/l2b_fd/BIO_FP_FD__L2B_I_G03_TS02E023_B001_02_D299R0</File_Name>
                                <File_Name>/data/BTK/L3_Delivery/TDS/TDS-BPS-
L3-001/l2b_fd/BIO_FP_FD__L2B_I_G04_TS02E023_B001_02_D299R0</File_Name>
                            </List_of_File_Names>
                        </Selected_Input>
                    </List_of_Selected_Inputs>
                </Input>
                <Input>
                    <Input_ID>FP_FH__L2B</Input_ID>
                    <Alternative_ID>FP_FH__L2B</Alternative_ID>
                    <List_of_Selected_Inputs>
                        <Selected_Input>
                            <File_Type>FP_FH__L2B</File_Type>
                            <List_of_File_Names>
                                <File_Name>/data/BTK/L3_Delivery/TDS/TDS-BPS-
L3-001/l2b_fh/BIO_FP_FH__L2B_I_G01_TS02E023_B001_02_D29611</File_Name>
                                <File_Name>/data/BTK/L3_Delivery/TDS/TDS-BPS-
L3-001/l2b_fh/BIO_FP_FH__L2B_I_G02_TS02E023_B001_02_D29611</File_Name>
                                <File_Name>/data/BTK/L3_Delivery/TDS/TDS-BPS-
L3-001/l2b_fh/BIO_FP_FH__L2B_I_G03_TS02E023_B001_02_D29611</File_Name>
                                <File_Name>/data/BTK/L3_Delivery/TDS/TDS-BPS-
L3-001/l2b_fh/BIO_FP_FH__L2B_I_G04_TS02E023_B001_02_D29611</File_Name>
                            </List_of_File_Names>
```

```
                        </Selected_Input>
                    </List_of_Selected_Inputs>
                </Input>
                <Input>
                    <Input_ID>AUX_PP3___</Input_ID>
                    <Alternative_ID>AUX_PP3___</Alternative_ID>
                    <List_of_Selected_Inputs>
                        <Selected_Input>
                            <File_Type>AUX_PP3___</File_Type>
                            <List_of_File_Names>
                                    <File_Name>/data/BTK/L3_Delivery/CONF/CONF-
BPS-PP3-
001/BIO_AUX_PP3____20170101T000000_20991231T000000_01_BHDHC0</File_Name>
                            </List_of_File_Names>
                        </Selected_Input>
                    </List_of_Selected_Inputs>
                </Input>
            </List_of_Inputs>
            <List_of_Outputs>
                <Output>
                    <File_Type>FP_AGB_L3</File_Type>
                    <File_Name_Pattern>BIO_FP_AGB__L3_*</File_Name_Pattern>
                     <File_Dir>/data/BTK/L3_Delivery/working_dir_mcampan/</File
_Dir>
                    <Baseline>02</Baseline>
                </Output>
                <Output>
                    <File_Type>FP_FH__L3</File_Type>
                    <File_Name_Pattern>BIO_FP_FH___L3_*</File_Name_Pattern>
                     <File_Dir>/data/BTK/L3_Delivery/working_dir_mcampan/</File
_Dir>
                    <Baseline>02</Baseline>
                </Output>
            </List_of_Outputs>
            <List_of_Intermediate_Outputs/>
        </Task>
    </List_of_Tasks>
</Job_Order>
```

Sample execution results

L3 Processor does not generate intermediate data. Output products are stored in the path specified in the Job Order and log file is stored in the same path of the Job Order file.

5.3.3.6. L1 Framing Processor

The L1 Framing Processor (L1F_P) implements one processing chain (or task):

- L1 Framing Processing (L1F_P), aimed at generating a set of L1 Virtual Frame products (one per frame) starting from a L0 product (one slice, both standard and monitoring).

In addition to this, L1 Framing Processor requires in input for its execution also a set of auxiliary files and internal resources.

The complete list of inputs, and of corresponding outputs, is reported in the following tables, indicating also the criteria to be used for selecting each input and the destinations of each output inside the overall BPS processing chain.
Please refer to [AD1] and [AD2] for the complete description of each field of these tables (a summary is reported also in Sec. 9) and to [RD4] for the complete description of each auxiliary file and internal resource.

L1 Framing Processing (L1F_P) task: inputs and outputs

Product Type	Description	Card.	Origin	Instances	Start / Stop Offset	Overlap / Min Overlap	Metadata Names	Max. # of Files	Ordering
Sx_RAW__0S	Stripmap Level-0 – Standard product	1	External	Single	0 / 0	Yes / 100	/	1	Latest
Sx_RAW__0M	Stripmap Level-0 – Monitoring product	1	External	Single	0 / 0	Yes / 100	/	1	MaxTemporalOverlap, Latest
AUX_ORB___	Auxiliary Orbit	1	External	Single	0 / 0	Yes / 100	/	1	MaxTemporalOverlap, ClosestStart

Product Type	Description	Card.	Dest.	Notes
CPF_L1VFRA	L1 Virtual Frame product	F	External	Collected by PF, but not used directly by other BPS tasks Used by PF to retrieve parameters needed to trigger L1_P task

Job Order preparation

To trigger a L1 Framing Processor execution a dedicated Job Order has to be filled. Here below are reported the list of sections and fields, with the instructions on how to fill them:

- `Processor_Name`, that shall be L1F_P;
- `Task_Name`, that shall be L1F_P (only one task for each Job Order);
- `Processor_Version` / `Task_Version`, that shall contain the processor version (in the form MM.mp, with M for major, m for minor and p for patch);
- a section (`List_of_Inputs`) where to specify all the input products and auxiliary files paths, selected according to the rules listed in the tables above, i.e.:
  - Sx_RAW__0S: Stripmap mode Level-0 Standard (L0S) product;
  - Sx_RAW__0M: Stripmap mode Level-0 Monitoring (L0M) product;
  - AUX_ORB___: Auxiliary Orbit product, including input position and velocity state vectors (one per second);
- a section (`List_of_Outputs`) where to specify all the output products paths, i.e.:
  - CPF_L1VFRA: L1 Virtual Frame products (one per frame);
    - Note: each product contains the parameters describing the frame (i.e., start and stop times, frame ID and frame status), that shall be used to fill the corresponding field in L1 Processor Job Order and to

trigger the generation of the corresponding L1 frame (see Sec. 5.3.3.1);

- fields (`Stdout_Log_Level` and `Stderr_Log_Level`) to set the log level;
- fields (`Number_of_CPU_Cores`, `Amount_of_RAM` and `Disk_Space`) to set the hardware resources allocated for the processing.

For the complete description of Job Order structure and fields refer to [AD2] and for the details on how to fill them to [AD1].

Note that, in case of execution from Docker image, instructions reported in Sec. 5.3.2 shall be followed. For what concern the volumes mounting, all the paths in the Job Order file shall refer to the Docker container file system.

In the box below a sample L1 Framing Processor Job Order is reported.

```
<?xml version="1.0" encoding="UTF-8"?>
<Job_Order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
schemaName="BIOMASS CPF-Processor ICD" schemaVersion="String">
  <Processor_Configuration>
    <File_Class>File_Class</File_Class>
    <Processor_Name>L1F_P</Processor_Name>
    <Processor_Version>03.30</Processor_Version>
    <Processing_Node>Processing_Node</Processing_Node>
    <List_of_Stdout_Log_Levels>
      <Stdout_Log_Level>INFO</Stdout_Log_Level>
    </List_of_Stdout_Log_Levels>
    <List_of_Stderr_Log_Levels>
      <Stderr_Log_Level>INFO</Stderr_Log_Level>
    </List_of_Stderr_Log_Levels>
    <Intermediate_Output_Enable>false</Intermediate_Output_Enable>
    <Processing_Station>Processing_Station</Processing_Station>
    <Request>
      <TOI>
        <Start>2017-02-25T09:46:04.620000</Start>
        <Stop>2017-02-25T09:47:03.747000</Stop>
      </TOI>
    </Request>
  </Processor_Configuration>
  <List_of_Tasks>
    <Task>
      <Task_Name>L1F_P</Task_Name>
      <Task_Version>03.30</Task_Version>
      <Number_of_CPU_Cores>1</Number_of_CPU_Cores>
      <Amount_of_RAM>2048</Amount_of_RAM>
      <Disk_Space>4096</Disk_Space>
      <List_of_Proc_Parameters />
      <List_of_Cfg_Files />
      <List_of_Inputs>
        <Input>
          <Input_ID>Sx_RAW__0x</Input_ID>
          <Alternative_ID>0001</Alternative_ID>
          <List_of_Selected_Inputs>
            <Selected_Input>
              <File_Type>S1_RAW__0S</File_Type>
              <List_of_File_Names>
                <File_Name>path to L0S product</File_Name>
              </List_of_File_Names>
            </Selected_Input>
```

```
          </List_of_Selected_Inputs>
        </Input>
        <Input>
          <Input_ID>Sx_RAW__0x</Input_ID>
          <Alternative_ID>0001</Alternative_ID>
          <List_of_Selected_Inputs>
            <Selected_Input>
              <File_Type>S1_RAW__0M</File_Type>
              <List_of_File_Names>
                <File_Name>path to L0M product</File_Name>
              </List_of_File_Names>
            </Selected_Input>
          </List_of_Selected_Inputs>
        </Input>
        <Input>
          <Input_ID>AUX_ORB___</Input_ID>
          <Alternative_ID>0001</Alternative_ID>
          <List_of_Selected_Inputs>
            <Selected_Input>
              <File_Type>AUX_ORB___</File_Type>
              <List_of_File_Names>
                <File_Name>path to AUX_ORB product</File_Name>
              </List_of_File_Names>
            </Selected_Input>
          </List_of_Selected_Inputs>
        </Input>
      </List_of_Inputs>
      <List_of_Outputs>
        <Output>
          <File_Type>CPF_L1VFRA</File_Type>
          <File_Name_Pattern>BIO_????_CPF_L1VFRA_*</File_Name_Pattern>
          <File_Dir>path to output folder</File_Dir>
          <Baseline>01</Baseline>
        </Output>
      </List_of_Outputs>
      <List_of_Intermediate_Outputs />
    </Task>
  </List_of_Tasks>
</Job_Order>
```

Sample execution results

L1 Framing Processor does not generate intermediate data. Output products are stored in the path specified in the Job Order and log file is stored in the same path of the Job Order file.

5.3.4. Logging functionalities

The logging strategy adopted by BPS processors is the one described in [AD2].

In particular, log messages are displayed on OS stdout and stderr stream. The minimum levels of the log messages emitted to the standard output and standard error streams can be specified via Job Order. Only messages having level greater or equal to these values will be emitted.

The following table reports the various types of log message, with a brief description and some examples. The adopted strategy is that:

- Error messages (E) go to stderr;
- all the other messages go to stdout.

Log message type	Description	Example
Debug (D)	Basic software tracing messages	`[D] AzProcBandwidth = 849.0Hz`
Information (I)	It contains info on kind/status of operations	`[I] Biomass L1 Processor started`
Progress (P)	It contains info on progress of operations (waitbars)	`[P] L1CoreProcessor - RANGE FOCUSING: 10%`
Warning (W)	An error occurred, the processor was able to continue	`[W] L1CoreProcessor - The estimated hIono value (0) is not valid, fall back to the climatological model`
Error (E)	An error occurred, the processor was not able to continue	`[E] L1PreProcessor - Directory <…> does not exist.`

NOTE: Progress type is not used by L2 Processors; moreover L2B_FD_P and L2B_FH_P do not implement Warning type.

Tab.5 BPS processors log types and samples

5.3.5. Exit Codes

The BPS processors customize the exit codes policy described in [AD2] as summarized in the following table.

Value	Mnemonic	Description
0	OK	The task execution was successfully completed.
128	FAILURE	Error exit code. The task is considered failed.

Tab.6 BPS processors Exit Codes.

5.3.6. Intermediate data and internal configuration files management

The L1 and Stack Processors generate during processing a set of intermediate data. They include "checkpoints" where to recover an interrupted processing execution (and in this case they are indicated as “breakpoint files”) or elements to support debug and interactive analyses. All the other BPS processors work entirely in memory.

All these intermediate data are stored into a unique folder created and populated during processing. This folder has the structure and content described in the following tables.
The lists include also the internal configuration files generated during processing by L1_P and STA_P to store useful processing parameters contained in input auxiliary files (see Sec. 5.3.3) plus several other lower-level processing parameters. The complete specifications of these files can be found in [RD1] annex.

Intermediate data folder element	Content
l1_pre_processor_output	Folder containing the L1 Pre-Processor intermediate data
l1_core_processor_output	Folder containing the L1 Core-Processor intermediate data
BPSConf.xml	L1 Processor logger internal configuration file
L1PreProcessorInputFile.xml	L1 Pre-Processor internal input file
L1PreProcessorConf.xml	L1 Pre-Processor internal configuration file
L1CoreProcessorInputDCPoly.xml	Doppler Centroid forced by user (only in case of DC estimation method set to Fixed)
L1CoreProcessorInputFile.xml	L1 Core-Processor internal input file
L1CoreProcessorProcessingOptions.xml	L1 Core-Processor internal configuration file
L1CoreProcessorProcessingParameters.xml	L1 Core-Processor internal configuration file
BIO_L1_P_VV.VV_YYYYMMDDTHHMMSS.log	L1 Processor log file
BPSL1ProcessorStatusFile.json	L1 Processor status file

Tab.7 L1 processor intermediate data folder structure.

Intermediate data folder element	Content
stack_pre_processor_brk	Output of the stack preprocessor
stack_coreg_processor_brk	Output of the coregistrator processor
BPSConf.xml	Coreg Processor logger internal configuration file
stackCoregConfig.xml	Coreg Processor internal configuration for secondaries
stackCoregPrimaryConfig.xml	Coreg Processor internal configuration for primaries

stackInputFile_XY.xml	Coreg Processor input files
aux_pps_previous.xml	Cached AUX-PPS file for resuming execution
job_order_previous.xml	Cached Job order file for resuming execution
STACK_PRE_PROC_COMPLETE.txt	State file to flag the pre-processor as completed
STACK_COREG_PROC_COMPLETE.json	State file that store the results of coregistration

Tab.8 Stack processor intermediate data folder structure.

The approach followed for the management of intermediate data folder is the following:

- if, via Job Order, an intermediate data folder is specified and the corresponding flag is activated (see Sec. 5.3.3), processor creates it at the desired position, folders and files are created inside it and kept at the end of the processing;
- otherwise, intermediate data are stored only temporarily during processing, in a folder located in the same directory of the Job Order; delete-on-consume mode is activated, then files are deleted during processing as soon as they are no more needed by the remaining processing steps, in order to reduce as much as possible disk occupation; at the end of the processing none of these files is kept on disk except for the L1 processor log file and L1 processor status file.

Some specific considerations shall be made for the L1 Processor, that when it is used in default environment (i.e., through Docker image) makes use by design of shared memory to store folders and files other than output products. This introduces some constraints to be able to:

- keep intermediate data directory at the end of the processing;
- support stop-and-resume functionality.

In fact, in case of processing interruption or in any case at the end of the processing, all the intermediate data would be lost. To support these, when an intermediate data folder is specified in Job Order and the corresponding flag is activated:

- when the processing is interrupted, the processor automatically copies all the files from the shared memory to this directory on disk;
- when the processor is restarted, the processor checks for the presence of such directory at the same path specified in the Job Order. If nothing is found, processing starts from the beginning, while if the directory is found it is moved back to the shared memory and then the processing is correctly resumed.

Due to this specific way of working, when the L1 Processor is executed running its Docker image, it is required to mount the shared memory resource on a specific path, specified via Job Order (see Sec. 5.3.2 and 5.3.3).

5.3.7. Stop-and-resume functionality

The BPS processors foreseeing the usage of breakpoint files (i.e., L1 and Stack processors) implement the stop-and-resume functionality. This means that if the processing is interrupted for any reason, it can be restarted from the latest breakpoint file successfully generated. This feature is then made possible by the presence of these breakpoint files in the intermediate data folder (and this is also the reason why other BPS processors, not foreseeing them, don't support this functionality), which shall then be activated via Job Order. The processor is able to automatically identify the files that are needed to restart and complete the execution of the remaining processing steps.

In the tables below are reported the complete lists of breakpoint files for L1 and Stack processors, indicating the points in the processing chain where the processing can be stopped and resumed. For more details refer to [AD1] and [RD1]).

Breakpoint files and other intermediate data	Description
iRAW iRAWDynCal iPGP iDrift iTxTP iChirp iEstNoise iPatt2D_D1H iPatt2D_D1V iPatt2D_D2H iPatt2D_D2V iSlantDEM iChannelImbalance.xml iDelays.xml ssp_headers.csv	Data after pre-processing Other intermediate data: Internal calibration parameters, antenna patterns, back-geocoded DEM
iRAW_rfi_corrected iRFI_time_domain_mask iRFI_freq_domain_mask	Data after RFI mitigation Other intermediate data: RFI masks
iRGC iDC iDC_std	Data after range compression, radiometric correction and calibration, azimuth resampling and DC and FR estimation Other intermediate data: DC grids
iSLC	Data after azimuth Compression and antenna pattern compensation
iSLC_iono_corrected iFR iPhaseScreenBB iFRPlane iMLKCoherence iMLKRLLRPhase	Data after polarimetric and ionospheric calibration Other intermediate data: Faraday Rotation and related maps, ionosphere phase screen
iSLC_af_corrected iPhaseScreenAF	Data after autofocus

	Other intermediate data: autofocus phase screen
iSRD	Data after multilooking
iSRD_denoised iSLC_nesz_map	Data after denoising Other intermediate data: Denoising maps
iGRD	Data after ground-range projection

** NOTE: If optional steps are off, the corresponding intermediate data are not generated. If autofocus is off but ionospheric calibration is on, input of Multilooking step is iSLC_iono_corrected product.*

Tab.9 L1 processor breakpoint files (in **bold**) and other intermediate data (in regular fonts

Breakpoint files and other intermediate data	Description	Cardinality
<L1a_product_name>_PF	Pre-processed L1a input product	Stack size
<L1a_product_name>_XYZ	DEM product from pre-processor	1
<L1a_product_name>_PF_Cor	Coregistered frames	Stack size
<L1a_product_name>_PF_DSI	Synthetic interferograms	Stack size
<L1a_product_name>_PF_Kz	Vertical wavenumbers	Stack size
<L1a_product_name>_PF_DAPD	Absolute distances from reference	1
<L1a_product_name>_PF_IonoL1PhaseScreen	Shifted ionosphere from L1_P	Stack size
<L1a_product_name>_PF_Elev	Elevation angles	Stack size
<L1a_product_name>_PF_CorRg	Coreg shifts in range direction	Stack size
<L1a_product_name>_PF_CorAz	Coreg shifts in azimuth direction	Stack size
<L1a_product_name>_PF_CSA	Coregistration shift accuracy	Stack sizes

Tab.10 Stack processor breakpoint files. Note that the stack processor always generates intermediate files since they are needed to interface the coregistrator module with the stack orchestrator. Intermediate files are deleted once no longer useful if the user selects so via job-order configuration. The intermediate files with cardinality 1 are generated only for the coregistration primary. The _CSA product is generated only when the coregistrator is run with “Geometry and Data”.

5.4. Troubleshooting

All the BPS processors present functionalities that can be exploited for troubleshooting purposes. The main ones are the following:

- Tuning of the log level: properly setting the log level via Job Order interface (see Sec. 5.3.3), in particular increasing it up to DEBUG level (see Sec. 5.3.4), it is possible to display in the log messages very detailed information about the processing status and, in case of error, about its origin. This shall be the first strategy to adopt in case of issues;
- Activating the generation of intermediate data: in L1 and Stack Processor it is possible to activate via Job Order interface (see Sec. 5.3.3) the generation of intermediate data, so that they are not deleted at the end of the processing. In this way is will be possible to understand better at which stage of the processing a certain issue happens or if the data was presenting problems even before in the chain. The detailed description of intermediate data format is reported in [RD1] annex. This shall be the second strategy to adopt in case of issues;
- Exploiting the stop-and-resume mechanism (see Sec. 5.3.7): in L1 and Stack Processor, the processing can be stopped in any moment to investigate its status through e.g. the analysis of intermediate data (considering that all those required for a restart are left in the working directory). These can also be edited and then, when the processor is restarted, it uses the existing data to resume the processing.

If none of these works, a standard debugger shall be attached to the processor under analysis to debug it and identify the origin of the experienced issue.

6. APPENDIX A: Docker Files

In this section are provided the Docker files used to generate the BPS processors Docker images. In addition to these, it has to be noted that YAML files describing how to reproduce exactly the environment created in Docker images are included in BPS processors delivery package.

L1 Processor Docker file

```
FROM bps-prereq:latest

COPY bin/* /usr/local/bin/
COPY lib/* /usr/local/lib/
COPY pkgs /tmp/python-packages

ENV LD_LIBRARY_PATH /usr/local/lib

USER root
RUN bash -c "conda index --verbose /tmp/python-packages && \
    conda install -y --strict-channel-priority bps-l1_processor &&
\
    conda clean -afy" && \
    rm -fr /tmp/python-packages && \
    ln -s /opt/conda_folder/envs/env/bin/bps_l1_processor
/usr/local/bin/

USER nobody
CMD /bin/bash
```

Stack Processor Docker file

```
FROM bps-prereq:latest

COPY bin/* /usr/local/bin/
COPY lib/* /usr/local/lib/
COPY pkgs /tmp/python-packages

ENV NUMBA_CACHE_DIR "/tmp/numba"
ENV LD_LIBRARY_PATH /usr/local/lib

USER root
RUN mkdir /tmp/numba && \
    chmod 777 /tmp/numba && \
    bash -c "conda index --verbose /tmp/python-packages && \
    conda install -y --strict-channel-priority bps-stack_processor
&& \
    conda clean -afy" && \
    rm -rf /tmp/numba/* && \
    rm -fr /tmp/python-packages && \
    ln -s /opt/conda_folder/envs/env/bin/bps_stack_processor
/usr/local/bin/

USER nobody
```

```
CMD /bin/bash
```

L2A Processor Docker file

```
FROM bps-prereq:latest

COPY pkgs /tmp/python-packages

ENV NUMBA_CACHE_DIR "/tmp/numba"

USER root
RUN mkdir /tmp/numba && \
    chmod 777 /tmp/numba && \
    bash -c "conda index --verbose /tmp/python-packages && \
    conda install -y --strict-channel-priority bps-l2a_processor &&
\
    conda clean -afy" && \
    rm -rf /tmp/numba/* && \
    rm -fr /tmp/python-packages && \
    ln -s /opt/conda_folder/envs/env/bin/bps_l2a_processor
/usr/local/bin/

USER nobody
CMD /bin/bash
```

L2B Processor Docker file

```
FROM bps-prereq:latest

COPY pkgs /tmp/python-packages

ENV NUMBA_CACHE_DIR "/tmp/numba"

USER root
RUN mkdir /tmp/numba && \
    chmod 777 /tmp/numba && \
    bash -c "conda index --verbose /tmp/python-packages && \
    conda install -y --strict-channel-priority bps-l2b_fd_processor
&& \
    conda clean -afy" && \
    rm -rf /tmp/numba/* && \
    rm -fr /tmp/python-packages && \
    ln -s /opt/conda_folder/envs/env/bin/bps_l2b_fd_processor
/usr/local/bin/

USER nobody
CMD /bin/bash
```

```
FROM bps-prereq:latest

COPY pkgs /tmp/python-packages

ENV NUMBA_CACHE_DIR "/tmp/numba"
```

```
USER root
RUN mkdir /tmp/numba && \
    chmod 777 /tmp/numba && \
    bash -c "conda index --verbose /tmp/python-packages && \
    conda install -y --strict-channel-priority bps-l2b_fh_processor
&& \
    conda clean -afy" && \
    rm -rf /tmp/numba/* && \
    rm -fr /tmp/python-packages && \
    ln -s /opt/conda_folder/envs/env/bin/bps_l2b_fh_processor
/usr/local/bin/

USER nobody
CMD /bin/bash
```

```
FROM bps-prereq:latest

COPY pkgs /tmp/python-packages

ENV NUMBA_CACHE_DIR "/tmp/numba"

USER root
RUN mkdir /tmp/numba && \
    chmod 777 /tmp/numba && \
    bash -c "conda index --verbose /tmp/python-packages && \
    conda install -y --strict-channel-priority bps-
l2b_agb_processor && \
    conda clean -afy" && \
    rm -rf /tmp/numba/* && \
    rm -fr /tmp/python-packages && \
    ln -s /opt/conda_folder/envs/env/bin/bps_l2b_agb_processor
/usr/local/bin/

USER nobody
CMD /bin/bash
```

L3 Processor Docker file

```
FROM bps-prereq:latest

COPY pkgs /tmp/python-packages

ENV NUMBA_CACHE_DIR "/tmp/numba"

USER root
RUN mkdir /tmp/numba && \
    chmod 777 /tmp/numba && \
    bash -c "conda index --verbose /tmp/python-packages && \
    conda install -y --strict-channel-priority bps-
l2b_agb_processor && \
    conda clean -afy" && \
    rm -rf /tmp/numba/* && \
    rm -fr /tmp/python-packages && \
    ln -s /opt/conda_folder/envs/env/bin/bps_l3_processor
/usr/local/bin/
```

```
USER nobody
CMD /bin/bash
```

L1 Framing Processor Docker file

```
FROM bps-prereq:latest

COPY lib/* /usr/local/lib
COPY pkgs /tmp/python-packages

ENV LD_LIBRARY_PATH /usr/local/lib

USER root
RUN bash -c "conda index --verbose /tmp/python-packages && \
    conda install -y --strict-channel-priority bps-
l1_framing_processor && \
    conda clean -afy" && \
    rm -fr /tmp/python-packages && \
    ln -s /opt/conda_folder/envs/env/bin/bps_l1_framing_processor
/usr/local/bin/

USER nobody
CMD /bin/bash
```

Prereq Docker file

```
FROM registry.access.redhat.com/ubi8-minimal:latest

COPY pkgs /tmp/python-packages

ENV LANG en_US.utf8
ENV BASH_ENV "/etc/profile"
ENV PROJ_DATA "/opt/conda_folder/envs/env/share/proj"

USER root
RUN microdnf install libgomp libgfortran && \
    microdnf clean all && \
    cd /tmp && \
    curl -sSL https://github.com/conda-
forge/miniforge/releases/latest/download/Mambaforge-Linux-x86_64.sh
-o Mambaforge.sh && \
    chmod +x Mambaforge.sh && \
    ./Mambaforge.sh -b -p /opt/conda_folder && \
    /opt/conda_folder/bin/conda install -y -c conda-forge conda-
build && \
    /opt/conda_folder/bin/conda index --verbose /tmp/python-
packages && \
    /opt/conda_folder/bin/conda config --prepend channels conda-
forge && \
    /opt/conda_folder/bin/conda config --prepend channels
/tmp/python-packages && \
    /opt/conda_folder/bin/conda create -n env -y python=3.12 bps-
common bps-transcoder && \
    /opt/conda_folder/bin/conda clean -afy && \
    cp /opt/conda_folder/etc/profile.d/conda.sh
/etc/profile.d/conda.sh && \
```

```
    echo "conda activate env" >> /etc/profile.d/conda.sh && \
    rm -rf /tmp/*
```

7. APPENDIX B: Build procedure

In this section are described the procedures to build the BPS processors:

- Python conda packages
- corresponding C++ binaries;
- Docker images;

starting from corresponding source code repositories.

7.1. Overall build procedure

In order to build BPS processors, this four-steps procedure shall be followed:

1. Build the C++ executables and the shared libraries for L1PreProcessor, L1CoreProcessor, BPSStackProcessor, IonosphericHeightIRI20Estimator;
2. Build all the 23 conda packages;
3. Prepare the developer bundle;
4. Build the docker images.

In details:

1. Build the C++ executables and the shared libraries for L1PreProcessor, L1CoreProcessor, BPSStackProcessor, IonosphericHeightIRI20Estimator
   This step is described in Sec. 7.3 and generates the following list of items:
   - BiomassL0ImportPreProc
   - BPSL1CoreProcessor
   - BPSStackCoreProcessor
   - libareapplib_common.so
   - libareapplib_ionosphericCalibration.so
   - libareapplib_rfimitigator.so
   - libare_core.so
   - libare_focus.so
   - libare_parsing.so
   - libare_sarbase.so
   - IonosphericHeightIRI20Estimator
2. Build all the 23 conda packages
   This step is described in Sec. 7.2 and generates the following list of items:
   - arepytools-1.7.6-py_0.conda
   - arepyextras-runner-1.0.2-py_0.tar.bz2
   - bps-common-X.Y.Z-py_0.conda
   - bps-transcoder-X.Y.Z-py_0.conda
   - bps-l1_framing_processor-X.Y.Z-py312_0.conda
   - bps-l1_processor-X.Y.Z-py312_0.conda

- bps-l1_core_processor-X.Y.Z-py_0.conda
- bps-l1_pre_processor-X.Y.Z-py_0.conda
- bps-l2a_processor-X.Y.Z-py_0.conda
- bps-l2b_agb_processor-X.Y.Z-py_0.conda
- bps-l2b_fd_processor-X.Y.Z-py_0.conda
- bps-l2b_fh_processor-X.Y.Z-py_0.conda
- bps-l3_processor-X.Y.Z-py_0.conda
- bps-stack_processor-X.Y.Z-py312_0.conda
- bps-stack_cal_processor-X.Y.Z-py_0.conda
- bps-stack_coreg_processor-X.Y.Z-py_0.conda
- bps-stack_pre_processor-X.Y.Z-py_0.conda

3. Prepare the developer bundle

   The developer bundle contains the minimal set of items required to build the docker images and to install the processors (via bundle installation).

   To prepare a developer bundle this procedure shall be followed:

   a. Setup bundle structure;
   b. Download the EOCFI libs;
   c. Add the BPS conda packages to the bundle;
   d. Add the C++ binaries to the bundle;
   e. Verify the result;
   f. Compress the bundle.

   In details:

   a. Setup bundle structure

      Create an empty bundle with the following folder structure:

```
bundle/
├── l1_framing_processor
│   ├── lib
│   └── pkgs
│       ├── linux-64
│       └── noarch
├── l1_processor
│   ├── bin
│   ├── lib
│   └── pkgs
│       ├── linux-64
│       └── noarch
├── l2a_processor
│   └── pkgs
│       ├── linux-64
│       └── noarch
├── l2b_agb_processor
│   └── pkgs
│       ├── linux-64
│       └── noarch
├── l2b_fd_processor
│   └── pkgs
│       ├── linux-64
│       └── noarch
```

```
├── l2b_fh_processor
│   └── pkgs
│       ├── linux-64
│       └── noarch
├── l3_processor
│   └── pkgs
│       ├── linux-64
│       └── noarch
├── prereq
│   └── pkgs
│       ├── linux-64
│       └── noarch
└── stack_processor
    ├── bin
    ├── lib
    └── pkgs
        ├── linux-64
        └── noarch
```

b. Download the EOCFI libs;
Download the EOCFI libs (version: EOCFI-4.27-CPPLIB-LINUX64_LEGACY) from the official website (https://eop-cfi.esa.int/index.php/mission-cfi-software/eocfi-software, login required). Place the following files, located in libraries\LINUX64_LEGACY inside the .zip archive, in the lib folder of framing processor:
- libCfiLib.so
- libCfiOrbit.so
- libCfiPointing.so
- libgeotiff.so.2
- libproj.so.14
- libtiff.so.5
- libxml2.so.2
- libCfiDataHandling.so
- libCfiEECommon.so
- libCfiFileHandling.so

```
bundle/
└── l1_framing_processor
    └── lib
```

c. Add the C++ binaries to the bundle

```
l1_processor
├── bin
│   ├── BiomassL0ImportPreProc
│   ├── BPSL1CoreProcessor
│   └── IonosphericHeightIRI20Estimator
└── lib
    ├── libareapplib_common.so
    ├── libareapplib_ionosphericCalibration.so
    ├── libareapplib_rfimitigator.so
```

```
      ├── libare_core.so
      ├── libare_focus.so
      ├── libare_parsing.so
      └── libare_sarbase.so
stack_processor
├── bin
│   └── BPSStackProcessor
└── lib
      ├── libareapplib_common.so
      ├── libare_core.so
      ├── libare_parsing.so
      └── libare_sarbase.so
```

d. Verify the result

The developer bundle should have the following structure:

```
bundle/
├── l1_framing_processor
│   ├── lib
│   │   ├── libCfiDataHandling.so
│   │   ├── libCfiEECommon.so
│   │   ├── libCfiFileHandling.so
│   │   ├── libCfiLib.so
│   │   ├── libCfiOrbit.so
│   │   ├── libCfiPointing.so
│   │   ├── libgeotiff.so.2
│   │   ├── libproj.so.14
│   │   ├── libtiff.so.5
│   │   └── libxml2.so.2
│   └── pkgs
│       ├── linux-64
│       │   └── bps-l1_framing_processor-X.Y.Z-py312_0.conda
│       └── noarch
│           ├── arepyextras-runner-1.0.2-py_0.tar.bz2
│           ├── arepytools-1.7.6-py_0.conda
│           ├── bps-common-X.Y.Z-py_0.conda
│           └── bps-transcoder-X.Y.Z-py_0.conda
├── l1_processor
│   ├── bin
│   │   ├── BiomassL0ImportPreProc
│   │   ├── BPSL1CoreProcessor
│   │   └── IonosphericHeightIRI20Estimator
│   ├── lib
│   │   ├── libareapplib_common.so
│   │   ├── libareapplib_ionosphericCalibration.so
│   │   ├── libareapplib_rfimitigator.so
│   │   ├── libare_core.so
│   │   ├── libare_focus.so
│   │   ├── libare_parsing.so
│   │   └── libare_sarbase.so
│   └── pkgs
│       ├── linux-64
│       │   └── bps-l1_processor-X.Y.Z-py312_0.conda
│       └── noarch
│           ├── arepyextras-runner-1.0.2-py_0.tar.bz2
│           ├── arepytools-1.7.6-py_0.conda
│           ├── bps-common-X.Y.Z-py_0.conda
│           ├── bps-l1_core_processor-X.Y.Z-py_0.conda
```

```
│           ├── bps-l1_pre_processor-X.Y.Z-py_0.conda
│           └── bps-transcoder-X.Y.Z-py_0.conda
├── l2a_processor
│   └── pkgs
│       ├── linux-64
│       └── noarch
│           ├── arepyextras-runner-1.0.2-py_0.tar.bz2
│           ├── arepytools-1.7.6-py_0.conda
│           ├── bps-common-X.Y.Z-py_0.conda
│           ├── bps-l2a_processor-X.Y.Z-py_0.conda
│           └── bps-transcoder-X.Y.Z-py_0.conda
├── l2b_agb_processor
│   └── pkgs
│       ├── linux-64
│       └── noarch
│           ├── arepyextras-runner-1.0.2-py_0.tar.bz2
│           ├── arepytools-1.7.6-py_0.conda
│           ├── bps-common-X.Y.Z-py_0.conda
│           ├── bps-l2b_agb_processor-X.Y.Z-py_0.conda
│           └── bps-transcoder-X.Y.Z-py_0.conda
├── l2b_fd_processor
│   └── pkgs
│       ├── linux-64
│       └── noarch
│           ├── arepyextras-runner-1.0.2-py_0.tar.bz2
│           ├── arepytools-1.7.6-py_0.conda
│           ├── bps-common-X.Y.Z-py_0.conda
│           ├── bps-l2b_fd_processor-X.Y.Z-py_0.conda
│           └── bps-transcoder-X.Y.Z-py_0.conda
├── l2b_fh_processor
│   └── pkgs
│       ├── linux-64
│       └── noarch
│           ├── arepyextras-runner-1.0.2-py_0.tar.bz2
│           ├── arepytools-1.7.6-py_0.conda
│           ├── bps-common-X.Y.Z-py_0.conda
│           ├── bps-l2b_fh_processor-X.Y.Z-py_0.conda
│           └── bps-transcoder-X.Y.Z-py_0.conda
├── l3_processor
│   └── pkgs
│       ├── linux-64
│       └── noarch
│           ├── arepyextras-runner-1.0.2-py_0.tar.bz2
│           ├── arepytools-1.7.6-py_0.conda
│           ├── bps-common-X.Y.Z-py_0.conda
│           ├── bps-l3_processor-X.Y.Z-py_0.conda
│           └── bps-transcoder-X.Y.Z-py_0.conda
├── prereq
│   └── pkgs
│       ├── linux-64
│       └── noarch
│           ├── arepyextras-runner-1.0.2-py_0.tar.bz2
│           ├── arepytools-1.7.6-py_0.conda
│           ├── bps-common-X.Y.Z-py_0.conda
│           └── bps-transcoder-X.Y.Z-py_0.conda
└── stack_processor
    ├── bin
    │   └── BPSStackProcessor
    ├── lib
    │   ├── libareapplib_common.so
```

```
        │   ├── libare_core.so
        │   ├── libare_parsing.so
        │   └── libare_sarbase.so
        └── pkgs
            ├── linux-64
            │   └── bps-stack_processor-X.Y.Z-py312_0.conda
            └── noarch
                ├── arepyextras-runner-1.0.2-py_0.tar.bz2
                ├── arepytools-1.7.6-py_0.conda
                ├── bps-common-X.Y.Z-py_0.conda
                ├── bps-stack_cal_processor-X.Y.Z-py_0.conda
                ├── bps-stack_coreg_processor-X.Y.Z-py_0.conda
                ├── bps-stack_pre_processor-X.Y.Z-py_0.conda
                └── bps-transcoder-X.Y.Z-py_0.conda
```

e. Compress the bundle

```
$ tar -czf bps-bundle-vX.Y.Z.tar.gz bundle
```

4. Build the docker images
   See also Sec. 7.4.

```
$ ./build_docker_from_bundle.sh --bundle=bps-bundle-vX.Y.Z.tar.gz
```

7.2. Build Python conda packages

These are the development instructions for manual conda packages generation.

Repository setup

1. Clone the bps Python source code repository (https://bps.service.aresys.it/bps/bps);
2. Clone also the arepytools and arepyextras-runner repositories (https://bps.service.aresys.it/bps/arepytools, https://bps.service.aresys.it/bps/arepyextras-runner)

Conda packages generation

Environment setup, create a build environment with

- these build tools installed:
  - conda-build
  - build
- these conda channels available:
  - conda-forge

```
$ conda create -n build_environment python=3.12
```

```
$ conda activate build_environment

$ conda config -env --prepend channels conda-forge

$ conda install conda-build build
```

1. Create a local conda channel

```
$ mkdir -p local_bps_channel/linux-64 && mkdir -p
local_bps_channel/noarch

$ conda index local_bps_channel

$ conda config --env --prepend channels ${PWD}/local_bps_channel
```

2. Build the source distribution for arepytools, arepyextras-runner and for each BPS package. For instance, the bps-common package is built by:

```
$ cd bps-common

$ python -m build --sdist

$ cd ..
```

The source distribution files are inside each package folder in the **dist** folder;

3. For each package, build the conda package:

```
$ cd bps-common

$ conda build recipe --output-folder ../local_bps_channel

$ cd -
```

To build each conda package, conda needs to find all the package dependencies. This is a possible source of issues only when looking for other bps packages. For instance, when building the bps-transcoder package conda will look for the bps-common package. It is therefore necessary to build the bps packages in a proper order to guarantee that conda will find all the required dependencies when building each package. Those dependencies are listed in section 4.4.1. A possible ordering is the following:

- `arepytools`
- `arepyextras-runner`
- `bps-common`
- `bps-transcoder`
- `bps-l1_framing_processor`

- `bps-l1_pre_processor`
- `bps-l1_core_processor`
- `bps-l1_processor`
- `bps-stack_pre_processor`
- `bps-stack_coreg_processor`
- `bps-stack_cal_processor`
- `bps-stack_processor`
- `bps-l2a_processor`
- `bps-l2b_fd_processor`
- `bps-l2b_fh_processor`
- `bps-l2b_agb_processor`
- `bps-l3_processor`

Note that bps-l1_framing_processor, bps-l1_processor and bps-stack_processor are architecture dependent packages. By default, they are built for python 3.12. A different python version can be specified thorough the `--python` option of the conda build command.

7.3. Build C++ executables

Three different source packages are provided:

- bps source package with binary core libraries
  - this package contains the source code only for the functionalities not covered by BIPR and developed specifically for the BPS project
- bps source package
  - this package contains the entire source code, including the functionalities covered by BIPR and developed outside BPS project
- IonosphericHeightIRI20Estimator

The difference between the first two packages is that the first one is fully open source and there are no limitations on its distribution, while the second one shall undergo the BPS contractual constraints.
The instructions to build them are in the following three sections.
Please note that, following these instructions starting from the second package, the binaries contained in the first one can be re-generated.

7.3.1. L1PreProcessor, L1CoreProcessor, BPSStackProcessor (from bps-binaries package)

These are the development instructions for manual C++ executables generation starting from package containing BPS source code + core libraries.

Software prerequisites

- BPS C++ executables binaries code package (bps-binaries-vM.m.p.tar.gz)
- C++14-compliant compiler
- CMake (version equal or greater than 3.20)
- make

Build

1. We assume that the BPS package has been extracted to a folder named "bps_binaries_srcbin".
   From its parent folder, create a new build folder and move to it:

```
$ mkdir build

$ cd build
```

2. To configure CMake and build the BPS, run the following CMake commands:

```
$ cmake ../bps_binaries_srcbin \

 -DBrickSAR_EXT_SDK_PATH=../bps_binaries_srcbin/extern

$ cmake --build .
```

Install

1. To install the BPS, run the following CMake command from the build folder:

```
$ cmake --install . \

 --prefix /path/to/bps/installation --component Runtime
```

2. Copy the core libraries dynamic dependencies to the install dir:

```
$ cp ../bps_binaries_srcbin/lib/*.so /path/to/bps/installation/lib
```

3. To run the executables, add `/path/to/bps/installation/bin` to the PATH environment variable and `/path/to/bps/installation/lib` to the LD_LIBRARY_PATH environment variable.

The expected content of the installation folder is

```
├── bin
│   ├── BPSL1CoreProcessor
```

```
│   ├── BPSStackProcessor
│   └── BiomassL0ImportPreProc
└── lib
    ├── libare_core.so
    ├── libare_focus.so
    ├── libare_parsing.so
    ├── libare_sarbase.so
    ├── libareapplib_common.so
    ├── libareapplib_ionosphericCalibration.so
    └── libareapplib_rfimitigator.so
```

7.3.2. L1PreProcessor, L1CoreProcessor, BPSStackProcessor (from bps-source package)

These are the development instructions for manual C++ executables generation from package containing full BPS source code.

Software prerequisites

- BPS C++ executables source code package (bps-source-vM.m.p.tar.gz)
- C++14-compliant compiler
- CMake (version equal or greater than 3.
- make

Build

1. We assume that the BPS package has been extracted to a folder named "bps_binaries_src".
   From its parent folder, create a new build folder and move to it:

```
$ mkdir build

$ cd build
```

2. To configure CMake and build the BPS, run the following CMake commands:

```
$ cmake ../bps_binaries_src \

 -DARE_EXT_SDK_PATH=../bps_binaries_src/extern

$ cmake --build .
```

Install

1. To install the BPS, run the following CMake command from the build folder:

```
$ cmake --install . --prefix /path/to/bps/installation --component
Runtime
```

2. To run the executables, add `/path/to/bps/installation/bin` to the PATH environment variable and `/path/to/bps/installation/lib` to the LD_LIBRARY_PATH environment variable.

The expected content of the installation folder is

```
├── bin
│   ├── BPSL1CoreProcessor
│   ├── BPSStackProcessor
│   └── BiomassL0ImportPreProc
└── lib
    ├── libare_core.so
    ├── libare_focus.so
    ├── libare_parsing.so
    ├── libare_sarbase.so
    ├── libareapplib_common.so
    ├── libareapplib_ionosphericCalibration.so
    └── libareapplib_rfimitigator.so
```

7.3.3. IonosphericHeightIRI20Estimator

This project provides bindings to the International Reference Ionosphere (IRI) model and easier access to its core function through a C interface.

Build requirements

- `gcc >= 7.3.1`
- `gfortran >= 7.3.1`
- CMake (version equal or greater than 3.14)
- make
- wget
- patch
- unzip

Build instructions

The project build is based on CMake.
A utility script to simplify configuration and build process is provided inside the 'proj' folder.
CMake downloads the IRI source files from IRI model website https://irimodel.org/IRI-2016/00_iri.tar and applies the patch available in the repository at ./patch/IRI-2016-addPath.patch.
To build the executable run the following

```
$ bash ./proj/tools/build.sh \

   --build-dir=build_dir --build-type=Release \
```

```
   --package=IonosphericHeightIRI20EstimatorPkg.tar.gz

$ tar -xvzf IonosphericHeightIRI20EstimatorPkg.tar.gz

$ tree IonosphericHeightIRI20Estimator
IonosphericHeightIRIEstimator
└── bin
    └── IonosphericHeightIRI20Estimator
```

It is also possible to use a local 00_iri.tar file instead of letting cmake download it.

```
$ bash ./proj/tools/build.sh            \

   --build-dir=build_dir --build-type=Release \

   --iri-model-archive=/path/to/00_iri.tar \

   --package=IonosphericHeightIRI20EstimatorPkg.tar.gz

$ tar -xvzf IonosphericHeightIRI20EstimatorPkg.tar.gz

$ tree IonosphericHeightIRI20Estimator

IonosphericHeightIRIEstimator

└── bin

    └── IonosphericHeightIRIEstimator
```

Type:

```
$ bash ./proj/tools/build.sh -help
```

to see other build and packaging options.

7.3.4. External libraries

In this section are reported for reference the build instructions for external libraries.

To build EXT SDK the following tools are required:

- CMake 3.20 (or more recent)
- Ninja 1.8 (or more recent)
- GNU GCC 8(or more recent)

Also, the following commands need to be available in the PATH:

- autoconf
- base64

- diff
- install
- libtool
- make
- patch
- scanelf
- tar
- zstd

Once the required dependencies are installed, the fetch.sh and build.sh scripts in proj/tools can be used to build EXT SDK on Linux.
The two scripts will install EXT SDK in the destination directory and create a deploy archive.

Usage is as follows:

```
$ ./proj/tools/fetch.sh <SrcURL>
$ ./proj/tools/build.sh <DstDir> <Archive>.sh
```

fetch.sh downloads the needed pre-built libraries, either from a remote URL or a local path (in which case the prefix file:// is required).

In both cases <SrcURL> content must be the following:

```
.
├── eocfi
│   └── EOCFI-4.28-CLIB-LINUX64.zip
└── intel
    ├──  ipp-2022.1.0-linux.tar.xz
    └──  mkl-2025.1.0-linux.tar.xz
```

These three libraries cannot be downloaded with a direct link as done for all the others.
EOCFI can be downloaded from https://eop-cfi.esa.int/index.php/mission-cfi-software/eocfi-software .

Intel libraries should be downloaded from the intel official website:

- https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html
- https://www.intel.com/content/www/us/en/developer/tools/oneapi/ipp.html

These versions are no longer available on the intel website since their policy is to remove the libraries yearly.
To overcome this problem, we have delivered, inside the source package, the redistributable part of those archives.
In order to update those libraries, two extract scripts are available at https://bps.service.aresys.it/bps/bps-ext-sdk and can be adapted to support newer version of those libraries available on the intel website.

Once <Archive>.sh is generated, it can be run like a regular script to deploy EXT SDK. Usage is as follows:

```
$ <Archive>.sh <DeployDir>
```

7.4. Build Docker images

Requirements

- linux machine with docker and xz (compression tool) installed
- internet connection

Procedure

In order to generate the docker files, you need to:

1. download the delivery bundle archive (no need to extract it)
2. execute the build script:

```
$ cd bps-dockerfiles

$ ./build_docker_from_bundle.sh --bundle=/path/to/bps-bundle-
vM.m.p.tar.gz
```

To build the docker images with the same environment, different dockerfiles needs to be used w.r.t. those described in Sec. 6. Each of them differs from the reference one by two aspects:

- copy (and final removal) of the environment file;
- conda creation/install step instruction replaced to exploit environment file.

L1 Processor Docker file

```
FROM bps-prereq:latest

COPY bin/* /usr/local/bin/
COPY lib/* /usr/local/lib/
COPY pkgs /tmp/python-packages
COPY environment.yml /tmp/environment.yml

ENV LD_LIBRARY_PATH /usr/local/lib

USER root
RUN bash -c "conda index --verbose /tmp/python-packages && \
    conda env update --file /tmp/environment.yml && \
    conda clean -afy" && \
    rm -fr /tmp/python-packages && \
    rm -fr /tmp/environment.yml && \
    ln -s /opt/conda_folder/envs/env/bin/bps_l1_processor
/usr/local/bin/

USER nobody
```

```
CMD /bin/bash
```

Stack Processor Docker file

```
FROM bps-prereq:latest

COPY bin/* /usr/local/bin/
COPY lib/* /usr/local/lib/
COPY pkgs /tmp/python-packages
COPY environment.yml /tmp/environment.yml

ENV NUMBA_CACHE_DIR "/tmp/numba"
ENV LD_LIBRARY_PATH /usr/local/lib

USER root
RUN mkdir /tmp/numba && \
    chmod 777 /tmp/numba && \
    bash -c "conda index --verbose /tmp/python-packages && \
    conda env update --file /tmp/environment.yml && \
    conda clean -afy" && \
    rm -rf /tmp/numba/* && \
    rm -fr /tmp/python-packages && \
    rm -fr /tmp/environment.yml && \
    ln -s /opt/conda_folder/envs/env/bin/bps_stack_processor
/usr/local/bin/

USER nobody
CMD /bin/bash
```

L2A Processor Docker file

```
FROM bps-prereq:latest

COPY pkgs /tmp/python-packages
COPY environment.yml /tmp/environment.yml

ENV NUMBA_CACHE_DIR "/tmp/numba"

USER root
RUN mkdir /tmp/numba && \
    chmod 777 /tmp/numba && \
    bash -c "conda index --verbose /tmp/python-packages && \
    conda env update --file /tmp/environment.yml && \
    conda clean -afy" && \
    rm -rf /tmp/numba/* && \
    rm -fr /tmp/python-packages && \
    rm -fr /tmp/environment.yml && \
    ln -s /opt/conda_folder/envs/env/bin/bps_l2a_processor
/usr/local/bin/

USER nobody
CMD /bin/bash
```

L2B Processor Docker file

```
FROM bps-prereq:latest

COPY pkgs /tmp/python-packages
COPY environment.yml /tmp/environment.yml

ENV NUMBA_CACHE_DIR "/tmp/numba"

USER root
RUN mkdir /tmp/numba && \
    chmod 777 /tmp/numba && \
    bash -c "conda index --verbose /tmp/python-packages && \
    conda env update --file /tmp/environment.yml && \
    conda clean -afy" && \
    rm -rf /tmp/numba/* && \
    rm -fr /tmp/python-packages && \
    rm -fr /tmp/environment.yml && \
    ln -s /opt/conda_folder/envs/env/bin/bps_l2b_fd_processor
/usr/local/bin/

USER nobody
CMD /bin/bash
```

```
FROM bps-prereq:latest

COPY pkgs /tmp/python-packages
COPY environment.yml /tmp/environment.yml

ENV NUMBA_CACHE_DIR "/tmp/numba"

USER root
RUN mkdir /tmp/numba && \
    chmod 777 /tmp/numba && \
    bash -c "conda index --verbose /tmp/python-packages && \
    conda env update --file /tmp/environment.yml && \
    conda clean -afy" && \
    rm -rf /tmp/numba/* && \
    rm -fr /tmp/python-packages && \
    rm -fr /tmp/environment.yml && \
    ln -s /opt/conda_folder/envs/env/bin/bps_l2b_fh_processor
/usr/local/bin/

USER nobody
CMD /bin/bash
```

```
FROM bps-prereq:latest

COPY pkgs /tmp/python-packages
COPY environment.yml /tmp/environment.yml

ENV NUMBA_CACHE_DIR "/tmp/numba"

USER root
RUN mkdir /tmp/numba && \
    chmod 777 /tmp/numba && \
    bash -c "conda index --verbose /tmp/python-packages && \
    conda env update --file /tmp/environment.yml && \
    conda clean -afy" && \
```

```
    rm -rf /tmp/numba/* && \
    rm -fr /tmp/python-packages && \
    rm -fr /tmp/environment.yml && \
    ln -s /opt/conda_folder/envs/env/bin/bps_l2b_agb_processor
/usr/local/bin/

USER nobody
CMD /bin/bash
```

L3 Processor Docker file

```
FROM bps-prereq:latest

COPY pkgs /tmp/python-packages
COPY environment.yml /tmp/environment.yml

ENV NUMBA_CACHE_DIR "/tmp/numba"

USER root
RUN mkdir /tmp/numba && \
    chmod 777 /tmp/numba && \
    bash -c "conda index --verbose /tmp/python-packages && \
    conda env update --file /tmp/environment.yml && \
    conda clean -afy" && \
    rm -rf /tmp/numba/* && \
    rm -fr /tmp/python-packages && \
    rm -fr /tmp/environment.yml && \
    ln -s /opt/conda_folder/envs/env/bin/bps_l3_processor
/usr/local/bin/

USER nobody
CMD /bin/bash
```

L1 Framing Processor Docker file

```
FROM bps-prereq:latest

COPY lib/* /usr/local/lib/
COPY pkgs /tmp/python-packages
COPY environment.yml /tmp/environment.yml

ENV LD_LIBRARY_PATH /usr/local/lib

USER root
RUN bash -c "conda index --verbose /tmp/python-packages && \
    conda env update --file /tmp/environment.yml && \
    conda clean -afy" && \
    rm -fr /tmp/python-packages && \
    rm -fr /tmp/environment.yml && \
    ln -s /opt/conda_folder/envs/env/bin/bps_l1_framing_processor
/usr/local/bin/

USER nobody
CMD /bin/bash
```

Prereq Docker file

```
FROM registry.access.redhat.com/ubi8-minimal:latest

COPY pkgs /tmp/python-packages
COPY environment.yml /tmp/environment.yml

ENV LANG en_US.utf8
ENV BASH_ENV "/etc/profile"
ENV PROJ_DATA "/opt/conda_folder/envs/env/share/proj"

USER root
RUN microdnf install libgomp libgfortran && \
    microdnf clean all && \
    cd /tmp && \
    curl -sSL https://github.com/conda-
forge/miniforge/releases/latest/download/Mambaforge-Linux-x86_64.sh
-o Mambaforge.sh && \
    chmod +x Mambaforge.sh && \
    ./Mambaforge.sh -b -p /opt/conda_folder && \
    /opt/conda_folder/bin/conda install -y -c conda-forge conda-
build && \
    /opt/conda_folder/bin/conda index --verbose /tmp/python-
packages && \
    /opt/conda_folder/bin/conda config --prepend channels conda-
forge && \
    /opt/conda_folder/bin/conda config --prepend channels
/tmp/python-packages && \
    /opt/conda_folder/bin/conda env create -f /tmp/environment.yml
&& \
    /opt/conda_folder/bin/conda clean -afy && \
    cp /opt/conda_folder/etc/profile.d/conda.sh
/etc/profile.d/conda.sh && \
    echo "conda activate env" >> /etc/profile.d/conda.sh && \
    rm -rf /tmp/*
```

8. APPENDIX C: Instructions for developers

As described in [RD1], the L1_P processor is designed to integrate dynamically the following libraries:

- RFI library: dynamic library for Interference mitigation;
- Ionosphere library: dynamic library for ionospheric artefacts estimation and correction.

These are provided as dynamic libraries in order to allow replacement, additions or removal of the corresponding algorithms without changing the architecture or structure of the L1 Processor software. In this section are described the interface constraints allowing a seamless replacement of the two libraries.

RFI library

The RFI library shall implement the following interface:

.hpp

```
#include "are_apps/Common/AresysApplication.hpp"

namespace are_apps
{
class RFIMitigatorApp : public AresysApplication
{
public:
    void run(const are_inputfile::Step& i_step) override;
};
}
```

.cpp

```
#include "are_apps/Common/aresys_generic_inputfile.hpp"

namespace are_apps
{
void RFIMitigatorApp::run(const are_inputfile::Step& i_step)
{
    const auto& step = i_step.RFIMitigator().get();

    // Inputs/outputs retrieval
    const std::string inputProduct = step.InputProduct();
    const std::string configurationFileName =
step.ConfigurationFileName();
    const std::string outputProduct = step.OutputProduct();
    const std::string outputRFITDMask =
step.OutputTimeDomainInterferenceMaskProduct();
    const std::string outputRFIFDMask =
step.OutputFrequencyDomainInterferenceMaskProduct();

    // Implementation
    [...]
```

```
}
}
```

where:

- InputProduct: path to the intermediate product received in input by the RFI mitigation step (iRAW in [RD1]);
- ConfigurationFileName: path to the configuration file received in input by the RFI mitigation step (L1CoreProcessorProcessingParameters.xml in [RD1]);
- OutputProduct: path to the intermediate product provided in output by the RFI mitigation step (iRAW_rfi_corrected in [RD1]);
- OutputTimeDomainInterferenceMaskProduct: path to the intermediate product provided in output by the RFI mitigation step and containing RFI Time-Domain masks (iRFI_time_domain_mask in [RD1]);
- OutputFrequencyDomainInterferenceMaskProduct: path to the intermediate product provided in output by the RFI mitigation step and containing RFI Frequency-Domain masks (iRFI_freq_domain_mask in [RD1]);

Ionosphere library

The ionosphere library shall implement the following interface:

.hpp

```
#include "are_apps/Common/AresysApplication.hpp"

namespace are_apps
{
class IonosphericCalibrationApp : public AresysApplication
{
public:
    void run(const are_inputfile::Step& i_step) override;
};
}
```

.cpp

```
#include "are_apps/Common/aresys_generic_inputfile.hpp"

namespace are_apps
{
void IonosphericCalibrationApp::run(const are_inputfile::Step& i_step)
{
    const auto& step = i_step.IonosphericCalibration().get();

    // Inputs/outputs retrieval
    const std::string inputProduct = step.InputProduct();
    const std::string inputGMF = step.GeomagneticFieldModel();
    const std::string inputTEC = step.InputTecMapProduct();
    const std::string inputIonosphereHeight = step.HeightModelProduct();
    const std::string configurationFileName = step.ConfigurationFile();
    const std::string outputProduct = step.OutputProduct();
    const std::string outputFR = step.OutputFaradayRotationProduct();
    const std::string outputPhaseScreen =
step.OutputPhaseScreenProduct();
```

```
    // Implementation
    [...]
}
}
```

where:

- InputProduct: path to the intermediate product received in input by the ionosphere calibration step (iSLC in [RD1]);
- ConfigurationFileName: path to the configuration file received in input by the ionosphere calibration step (L1CoreProcessorProcessingParameters.xml in [RD1]);
- GeomagneticFieldModel: path to the product received in input by the ionosphere calibration step and containing the Geomagnetic Field Model (GMF in [RD1]);
- InputTecMapProduct: path to the product received in input by the ionosphere calibration step and containing the TEC (AUX_TEC in [RD1]);
- HeightModelProduct: (optional) path to the product received in input by the ionosphere calibration step and containing the ionosphere height information;
- OutputProduct: path to the intermediate product provided in output by the ionosphere calibration step (iSLC_iono_corrected in [RD1]);
- OutputFaradayRotationProduct: path to the intermediate product provided in output by the ionosphere calibration step and containing Faraday Rotation (iFR in [RD1]);
- OutputPhaseScreenProduct: path to the intermediate product provided in output by the ionosphere calibration step and containing phase screen (iPhaseScreenBB in [RD1]);

9. APPENDIX D: Processors inputs selection criteria

The tables reported in Sec. 5.3.3 describe the inputs of each BPS processor, indicating also the criteria to be used for selecting each of them. In this section are reported more details about the meaning of each entry of these tables. Note that this section is intended just as a summary, while the complete description can be found in [AD2] and in [AD3].

Product Type	Description	Card.	Origin	Instances	Start / Stop Offset	Overlap / Min Overlap	Metadata Names	Max. # of Files	Ordering

For each input, the tables specify:

- **Cardinality / Max. # of Files**: number of products of this type that can be selected and used to feed the processor (cardinality specifies the nominal range, while maximum number of files specifies the maximum number accepted, even considering potential contingency cases);
- **Origin**: product source, always set to “External” for BPS processors, meaning that the product is generated by another processor in the suite;
- **Instances**: mode of execution of the processor, always set to “Single” for BPS processors, meaning that a single instance is executed even if more products of this type are provided in input.

In addition, different criteria can be used to select these inputs:

- **Temporal selection criteria**: in this case the selection is based on a desired time interval, identified by a start and a stop times:
  - **Start / Stop Offset**: offsets (in seconds) to be subtracted/added to the start/stop of the desired time interval before executing the input selection according to the rule specified by the following fields. Always set to “0 / 0” for BPS processors;
  - **Overlap / Min Overlap**: specify if the time validity (reported also in each product name) of the input data has to overlap the desired time interval and which is the minimum overlap required (in percentage);
- **Geospatial selection criteria**: in this case the selection is based on geospatial criteria. Not used by BPS processor;
- **Metadata-based selection criteria**: in this case the selection is based on a specific set of product metadata, that are used to filter them:
  - **Metadata Names**: list of product metadata to be used for the selection.

Last information is the **ordering**: once that all the products have been selected following the criteria described above, they are ordered according to the specified criteria. Then only

the first “Max. # of Files” shall be used for processing. Here below a summary of what reported in [AD3] to describe the ordering criteria.
Assuming that:

- $[T_0, T_1]$ is the desired time interval;
- Δt_0 and Δt_1 are the start/stop offsets;
- $[V_0, V_1]$ is the product validity interval;
- *AOI* is the desired area, identified by a polygon;
- *AOI'* is the desired area modified by the geospatial offset;
- *G* is the geometry of the product identified by a polygon;
- *GI* is the intersection between *AOI'* and *G*, identified by a polygon (if there is no intersection, *GI* is undefined)

the ordering criteria are:

- *Latest*: the input data are ordered by file creation date. The list of input data is ordered by descending values, i.e. from largest to smallest;
- *MaxTemporalOverlap*: the input data are ordered by temporal overlap *TO*:
$$T0 = \max\left(0, \min(T_1 + \Delta t_1, V_1) - \max(T_0 - \Delta t_0, V_0)\right)$$
The list of input data is ordered by descending values, i.e. from largest *TO* to smallest;
- *MaxGeospatialOverlap*: the input data are ordered by geospatial overlap *GO* which is the area of *GI*. If *GI* is undefined (i.e. no intersection), *GO* is zero. The list of input data is ordered by descending values, i.e. from largest *GO* to smallest;
- *ClosestStart*: the input data are ordered by $|T_0 - \Delta t_0 - V_0|$. The list of input data is ordered by ascending values, i.e. from smallest to largest;
- *ClosestStop*: the input data are ordered by $|T_1 + \Delta t_1 - V_1|$. The list of input data is ordered by ascending values, i.e. from smallest to largest;
- *BestTimeCentered*: the input data are ordered by $|(T_1 + \Delta t_1 + T_0 - \Delta t_0)/2 - (V_1 + V_0)/2|$. The list of input data is ordered by ascending values, i.e. from smallest to largest.