Open Simulation Framework

openSF

SYSTEM USER'S MANUAL

Code :	OPENSF-DMS-SUM
Issue :	3.18
Date :	12/03/2020

	Name	Function	Signature
Prepared by	Carlos Pérez Sancha João Malés Javier Martín Gonzalo Vicario Rui Mestre Enrique del Pozo	Project Engineer Project Engineer Project Engineer Project Engineer Project Engineer Project Engineer	
Reviewed by	Federico Letterio	Project Manager	
Approved by	Federico Letterio	Project Manager	
Signatures and approvals on original			

DEIMOS Space S.L.U.
Ronda de Poniente, 19, Edificio Fiteni VI, 2-2ª
28760 Tres Cantos (Madrid), SPAIN
Tel.: +34 91 806 34 50 / Fax: +34 91 806 34 51
E-mail: deimos@deimos-space.com

© DEIMOS Space S.L.U., 2019

This page intentionally left blank

© DEIMOS Space S.L.U., 2019

Document Information

Contract Data	
Contract Number:	22852/09/NL/FF
Contract Issuer:	ESA/ESTEC

Internal Distribution		
Name	Unit	Copies
Mariano Sánchez Nogales	Head of the Flight Systems Business Unit	1
Internal Confidentiality Level (DMS-COV-POL05)		
Unclassified ☑	Restricted ☐	Confidential ☐

External Distribution		
Name	Organization	Copies
Michele Zundo	ESA/ESTEC	1
Marcos Bento	ESA/ESTEC	1
Montserrat Piñol	ESA/ESTEC	1

Archiving	
Word Processor:	MS Word 2016
File Name:	OPENSF-DMS-SUM-318.doc

Document Status Log

Issue	Change description	Date
1.0	First issue of this document	21/12/09
1.1	Version of this document after openSF AR1 ❑ Installation details section completed ❑ New chapter describing the openSF web page. ❑ New chapter 5 tutorial for defining an E2E simulation in openSF ❑ Product tools section updated, now includes a list of popular product tools (section 4.5.4)	15/03/10
1.2	New version in response to ESA assessment for openSF version 1.1 ❑ Section 3.4.1.1 updated clarifying that the openSF installation mechanism for Linux platforms is the same as the Windows one. Installation of JRE under Linux completed. ❑ Updated section 0. Bin folder reference removed. OSFI folder added ❑ Section 4.5.4 updated. Tools for MacOS issue. ❑ Added Annex A detailing how to build openSF from sources files. ❑ Updated chapter 5 with latest changes for the web site. ❑ Updated chapter 6. Folder structure guidelines.	20/04/10
1.3	Minor corrections: ❑ Clarifications on section 5.2 ❑ typos in Annex A	26/04/10
1.4	New version for the openSF v2 acceptance ❑ Section 4.5.2 added: Management of databases	22/09/10
1.5	Added Annex B detailing Parameter Editor functionality.	15/10/10

© DEIMOS Space S.L.U., 2019

Issue	Change description	Date
1.6	Update after openSF AR 2 meeting: ❑ Added IDL and Matlab windows for the Linux installation ❑ Updated functioning of databases in the multi-repository: Independence between databases. ❑ Added IDL requirements for Linux installation: Problems with installation path and different types of licenses ❑ Added Matlab requirements for Linux installation: licenses ❑ Updated introduction sentence in Annex A, section 6. Instructions to build the framework. ❑ Removed import capabilities in Simulation creation ❑ Updated obsolete screenshots. ❑ Added new section for module developers.	12/11/10
2.0	New version including extended capabilities for openSF 2.2: ❑ Parameter Perturbation plug-in (from SEPSO) ❑ Parameter Editor integration ❑ Tool management extension ❑ Check output generation ❑ MATLAB errors inclusion ❑ Import/Export capability ❑ Extended log capabilities ❑ Keyboard shortcuts ❑ HMI Isolation	16/02/12
2.1	New version after version 2.2 acceptance meeting ❑ Added IDL version selection to openSF installer (section 3.4.1.1) ❑ Plot perturbation capabilities	02/03/12

Issue	Change description	Date
3.0	New version including extended capabilities for openSF V3: ❑ Updated framework pre-requisites (section 3.3.3); ❑ Updated installation instructions, leaving only references to the supported operating system – Linux (section 3.4.1.1); ❑ Added references to OSFEG libraries; ❑ Added section on migration from previous versions of openSF to openSF V3 (section 3.5); ❑ Updated framework figures throughout section 4; ❑ Added reference to new system configuration parameter to control module parallelisation (section 4.5.1); ❑ Added section on importing an XML database definition (section 4.5.2.5); ❑ Added section on the CPU core usage view that supports module parallelisation (section 4.5.6); ❑ Added copy capabilities for several openSF elements (from section 4.7 to 4.11); ❑ Updated the module chain management in a simulation according to the openSF framework revision (section 4.11); ❑ Added capability for removal of intermediate output files (section 4.11.2.8.1); ❑ Added capabilities for simplified module management: ○ Switching a module version (section 4.11.5.1); ○ Bypass/Switch-off module execution (section 4.11.5.2); ○ Run from a given point in the module chain (section 4.11.5.3); ❑ Added capabilities for parallelisation of module execution (section 4.6); ❑ Added capability for exporting/importing module of an executed simulations (sections 4.11.9.3 and 4.11.9.4);	22/11/13

Issue	Change description	Date
3.1	New version answering the comments generated by ESA on the openSF V3 AR documentation package. Implementation of the following RIDs: ❑ OSF-AR3-05: Update semantics of maximum number of threads parameter with associated warning message (section 4.5.1); ❑ OSF-AR3-06: Updates database view related figures 4-14 and 4-20 (section 4.5.2); ❑ OSF-AR3-07: Added clarification on the log functionality in the case of module parallelisation (section 4.11.6.1); ❑ OSF-AR3-10: Clarification on simplification of simulation directory name (section 4.13). ❑ OSF-AR3-11: Renamed section 4.19 to “Table of keyboard shortcuts”; ❑ OSF-AR3-RF-01: Updated change log to list sections changed for openSF v3; ❑ OSF-AR3-RF-02: Updated Applicable and Reference documents (section 2.1 and 2.2) including also document versions; added section 1.1.1 identifying the changes from openSF V2.2 to V3; ❑ OSF-AR3-RF-03: Revised Tutorial (section 6) according to openSF V3 and added a reference to the training material (course handouts); ❑ OSF-AR3-CE-01: Corrected the supported IDL versions (section 3.3.3.3); ❑ OSF-AR3-CE-02: updated pre-requisites section to appear chronologically before framework installation (section 3.3.3). ❑ OSF-AR3-CE-03: Updated section to clarify typical definition of OPENSF_HOME; ❑ OSF-AR3-CE-04: Added a clarification regarding the installation an execution of openSF in the appropriate machine architecture (section 3.4.1.1). The implementation of these RIDs closes action ACT-AR3-03 from the AR3.	15/01/14
3.2	New version including the integration of Python modules in openSF.	04/04/14
3.3	Updated after review comments from ESA: implemented RIDS OPENSF_v3.2_RID_01 and OPENSF_v3.2_RID_07 by updating section 3.3.3.5.	30/04/14
3.4	New version with updated installation procedure including also the porting to OSX.	03/06/15
3.5	New version including extended capabilities for openSF V3.4: Time Based Scenario Orchestration. Included special conditions for installation in OSX.	02/12/15
3.6	OPENSF-AN-004: The “iterate parameters” functionality is able to import parameter iteration definitions from file OPENSF-AN-017: Updated SUM to use "modules" instead of "models". Added section on execution of script modules and special conditions applicable to OSX 10.11.	16/03/16

Issue	Change description	Date
3.7	Updated with extended capabilities for openSF V3.5: ❑ OPENSF-AN-003: Remote orchestration; ❑ OPENSF-AN-019: Automatic openSF version checking; ❑ OPENSF-AN-030: Select subset of parameters to monitor; ❑ Miscellanea HMI corrections and simplifications. ❑ Descriptor syntax clarifications.	16/05/16
3.8	Overall review based on ESA comments	06/06/16
3.9	Updated installation requirements.	20/01/17
3.10	Updated with HMI revamping for Eclipse RCP.	09/06/17
3.11	Updated with COTS requirement (Table 3-1 and Sec 3.3.2)	28/08/17
3.12	Remove OSFI from framework pre-requisites Update MySQL tools path configuration during installation Database created during installation selected by the user Update folder structure Small updates on database connection, deletion and backup due to bug fixes OpenSF log messages moved to simulation dedicated log files Move ParameterEditor appendix to dedicated SUM	15/12/17
3.13	Update for openSF version 3.7.2 ❑ §3: Removal of the V2 upgrade path. ❑ §3.1.1 and §3.3: Update of the supported platforms. ❑ Reorganization of §3.3.3, splitting off the module-specific requirements into §3.3.4. ❑ §3.5: Update of the command line arguments section. ❑ §4.9.5: New dialog for missing configuration files. ❑ §4.9.10: Update simulation export/import files, dialog ❑ §4.5.1, §4.12, §4.18: add per-module log files. ❑ §6: Update of build instructions	15/06/18

© DEIMOS Space S.L.U., 2019

Issue	Change description	Date
3.14	Update for openSF version 3.7.3: ❑ §3 and §4: Added Windows as a supported platform. ❑ §3.4.2: Updated the folder structure of the OpenSF installation folder. ❑ §3.4.2, §4.11.5.4 and §4.13: Documented new feature for grouping timeline and iteration/perturbation simulations. ❑ §3.3, §3.4, $4.5, §4.18: Updated the dependency from MySQL, since starting on v3.7.3 the client tools are not necessary. ❑ §4.5.1: XSD Validation and other changes in Preferences dialog. ❑ §All sections: Updated several images to comply with the newly delivered version. ❑ §3.4.3: updated licencing scheme ❑ Various minor changes	14/12/18
3.15	Update for openSF version 3.8.0: ❑ General revision and update of the document. ❑ §2: Update the description of the installation process. ❑ §4.9.2.11: New section on iterations on parameter sets. ❑ §4.13: Include the concept of the "User Roles" and explain them. ❑ §4.15 Addition of some guidelines on how to implement a Monte Carlo study in openSF.	06/06/19
3.16	Update for openSF version 3.8.1: ❑ §3.3.3.1 Modification of supported versions to align them with the pre-requisites listed under §3.3.3. ❑ §4.5.1 Added description of the new preferences added to the Application Settings preferences page.	17/07/19
3.17	Update for openSF version 3.9.0: ❑ Removal of the Stage and Simulation concepts. ❑ Renaming of the previous "Session" concept to "Simulation". ❑ Clarify the setup of the $INSTALL4J_JAVA_HOME environment variable. ❑ Update all images to the latest openSF HMI. ❑ Removed "product" concept.	27/11/19
3.18	Update for openSF version 3.9.2 ❑ New feature: Python-like format for parameter value edition.	12/03/20

Table of Contents

List of Figures

© DEIMOS Space S.L.U., 2019

1. INTRODUCTION

This document has been produced by DEIMOS within the frame of the openSF project and represents the Software User Manual for the openSF platform.

- Chapter 1, this present chapter, talks about the document, giving a description and settling the basis to understand it.
- Chapter 2 links this document with information from other sources.
- Chapter 3 details the procedures for setting up the openSF system.
- Chapter 4 describes one by one all the different functionalities of the openSF system.
- Chapter 5 gives a detailed description of the openSF web site.
- Chapter 6 details the procedures for creating an E2E simulation from scratch.

Reading the chapters in this order will help users to fully understand the use of the system.

1.1. Scope

This document applies to openSF v3.9.2 and is aimed to support two different kinds of users:

- Users that want to take advantage of the application.
- Developers that want to implement new modules intended to be used within openSF.

1.2. Acronyms and Abbreviations

Acronym	Description
AD	Applicable Document
API	Application Programming Interface
CFI	Customer Furnished Item
COTS	Commercial Off-The-Shelf
DBMS	Database Management System
DMS	DEIMOS Space
E2E	End to end simulation
GUI	Graphical User Interface
HMI	Human-Machine Interface
I/O	Input/Output
ICD	Interface Control Document
L1PP	Level 1 Processor Prototype

Acronym	Description
L2PP	Level 2 Processor Prototype
LCF	Local Configuration File
MC	Monte Carlo
RD	Reference Document
SEPSO	Statistical End-To-End Performance Simulator for Optical Imaging Sensors
SUM	System User Manual
TBC	To Be Confirmed
TBD	To Be Defined / Decided
TN	Technical Note
UML	Unified Modelling Language

1.3. Definitions

Definition	Meaning
Batch mode	It is the capability of the simulator to perform consecutive runs without a continuous interaction with the user. Batch mode checks the agreement or not between the output of a given module and the input by the next one in the sequence of the simulation. Several modes of executions can be performed: ❑ Iteratively, executing one or more simulations ❑ Iteratively, executing the same simulation several times depending on the parameters' configuration ❑ Same as above but by executing a batch script.
Configuration File	An XML file that contains parameters necessary to execute a module. A configuration file instance must comply with the corresponding XML schema defined at module creation time. A special case is the global configuration file that defines the configuration parameters that are common to all modules.
Descriptor	The descriptors define the set of input and output files used to connect modules in simulation runs. Each module has two descriptors associated, one for inputs and other for outputs, and they define the number and location of each of the IO files.
Framework	Software infrastructures designed to support and control the simulation definition and execution. It includes the GUI, domain and database capabilities that enable to perform all the functionality of the simulator.

Definition	Meaning
Module	Executable entity that can take part in a simulation. A module can be understood, broadly speaking, also as an “algorithm”. Basically, it contains the recipe to produce products function of inputs. A module contains also several rules to define the input, output and associated formats. Furthermore, its behaviour is controlled by one configuration file. Overall, the architecture of a module consists of: ❑ The source code and its binary compiled counterpart (or interpretable script) ❑ A configuration file with its parameters ❑ An input file that characterizes its inputs ❑ An output file that characterizes its outputs Note: Although sometimes the word Model appears instead of Module, they have the same meaning in this scope
Parameter	A constant whose value characterizes a given particularity of a module. Parameters are user-configurable, they are fixed before launching a module and, for practical reasons, not all of them shall be accessible from the HMI.
Simulation	A simulation is defined as an execution of a set of modules (either a unique execution or an iterative one with different parameter values). The restriction of how to concatenate these modules and the order on which they are executed is based on the logic imposed by the relation between their descriptors.
Time-Based Execution	The Time-Based scenario execution implements the notion of time driven execution of a simulation whereby each simulation module is invoked in a sequence of time segments.
Tool	A tool is an external executable file that performs a given action to a certain group of files. Used in openSF platform and associated to a certain file extension these tools can be called to perform off-line operations to products output in simulations.

2. RELATED DOCUMENTS

This section details the list of applicable and reference documents used for the generation of this document, as well as the standards that have been applied. Note that the latest issue and dates of the documents can be found on the openSF website (http://eop-cfi.esa.int/index.php/opensf).

2.1. Applicable Documents

The following table specifies the applicable documents that were compiled during the project development.

Table 2-1: Applicable documents

Reference	Code	Title	Issue
[AD-ICD]	openSF-DMS-ICD-001	openSF Interface Control Document	3.0.1
[AD-ADD]	openSF-DMS-ADD-001	openSF Architecture Design Document	2.2
[AD-E2E]	PE-ID-ESA-GS-464	ESA generic E2E simulator Interface Control Document	1.2.5

2.2. Reference Documents

The following table specifies the reference documents that shall be taken into account during the project development.

Table 2-2: Reference documents

Reference	Code	Title	Issue
[RD-OSFI-DM]	OSFI-DMS-TEC-DM	openSF Integration Libraries Developers Manual	1.18
[RD-OSFEG-DM]	OSFEG-DMS-TEC-DM	openSF Error Generation Libraries Developers Manual	1.32
[RD-TM]	openSF-3.2-Training	openSF Training Course	3.1
[RD-PE]	OPENSF-DMS-PE-SUM	ParameterEditor Software User’s Manual	1.4

2.3. Standards

The following table specifies the standards that were complied with during project development.

Table 2-3: Standards

Reference	Code	Title
[E40C]	ECSS-E-40C	Software development Standard
[XML]	(www.w3.org/TR/xml11/)	Extensible Markup Language (XML) 1.1

3. GETTING STARTED

3.1. Introduction

During the concept and feasibility studies for the ESA Earth Observation activities, the mission performance up to the final data scientific products needs to be predicted by means of end-to-end (E2E) simulators. The observing system characteristics that impact data quality need to be determined in order to achieve the scientific goals. On subsequent implementation phases, these mission end-to-end simulators become a coherent test bed for L1PP and L2PP and to support the verification of space segment performance and associated sensitivity analysis.

A mission E2E simulator is able to reproduce all significant processes, design and steps that impact the mission performance as well as output simulated data products.

Commonalities in the structure of these E2E simulators highlighted the need for a common modular framework. openSF is an open software framework to support a standardised set of end-to-end mission simulation capabilities allowing the assessment of the science goals and engineering requirements with respect to the mission objectives.

Scientific models and product exploitation tools can be plugged in the system platform with ease using a well-defined integration process.

3.1.1. Quick installation Recommended approach

Detailed System Requirements are presented in section 3.3. For a quick installation strategy, the recommended approach is the following:

- ❑ Ubuntu 18.04 LTS (or higher), macOS 10.14 or higher, or Windows 10;
- ❑ Oracle Java 8;
- ❑ MySQL server 5.7.x[1].

3.2. Conventions used in this Manual

This chapter lists all the conventions used throughout this Software User Manual.

3.2.1. OPENSF_HOME

All through the contents of this User Manual, a "variable" called OPENSF_HOME is exhaustively used as a placeholder. This variable value points to the root folder that contains the openSF installation. Typically, this folder could be similar to this (in a UNIX like operating system):

[1] The latest available version can be found in https://dev.mysql.com/downloads/mysql/5.7.html#downloads and the corresponding installations for the "compressed tar archive" Linux binary distributions in http://dev.mysql.com/doc/refman/5.7/en/binary-installation.html

```
/home/<user>/openSF
```

This variable matches with one "environment variable" defined in the SW.

3.2.2. Data types

The data types supported by openSF configuration files are described in [AD-E2E].

3.3. System Requirements

The openSF framework is developed and runs on the Eclipse Rich Client Platform (RCP). The current version (3.9.2) of openSF uses Eclipse 2019-03 (4.11), for which the target platforms officially supported by the Eclipse project can be found in the Eclipse project plan page.

3.3.1. Hardware requirements

Hardware must at least fulfil the following requirements:

- 64-bit 2GHz processor
- 4 GB of RAM memory installed
- 200MB of free space to install.

3.3.2. Operating system requirements

Not all the platforms targeted by the Eclipse platform are officially supported by openSF. Binary distributions are currently provided for the following platforms:

- Linux x86-64: in particular, openSF has been tested with Ubuntu 18.04.
- macOS x86-64, version 10.14 or higher.
- Windows 10 x86-64

3.3.3. Framework pre-requisites

Pre-requisite	Purpose	Licensing	Distribution site
Oracle Java (TM) Runtime Environment, Standard Edition 1.8 (64-bit)	openSF runs within this execution environment.	GNU GPL / Java Community Process	http://www.java.com/en/download/
MySQL community server 5.7.x [2]	openSF stores information in this relational database	GPL or Proprietary License	http://dev.mysql.com/downloads/mysql

[2] It's not mandatory to have the MySQL server installed in computer, since it can be accessed over the network.

Pre-requisite	Purpose	Licensing	Distribution site
MySQL user with DB creation/modification privileges	openSF needs DB creation/modification privileges	GPL or Proprietary License	http://dev.mysql.com/downloads/mysql
GTK+ v3.20 or higher (Linux only)	openSF uses GTK+ as the graphics library in Linux Note: GTK known bug with Table/Tree editing that affects versions < 3.20.	GNU LGPL	https://www.gtk.org/download/index.php
mpstat (Linux only)	openSF uses this library to assess the CPU core usage statistics	GNU GPL	https://linux.die.net/man/1/mpstat
sshfs	openSF uses this library for remote execution	GNU GPL	https://osxfuse.github.io/ (OSX) https://github.com/libfuse/sshfs (Linux)

It is recommended to use a Java Runtime Environment from Oracle, although the corresponding IBM or openJDK versions should work as well. All the openSF software pre-requisites are freely downloadable, the links for them can be found in the openSF website.

3.3.3.1. MySQL Installation Guide

openSF currently requires MySQL server version 5.7.x

openSF present version ensures full compatibility with MySQL v5.7.x.

Note that the framework relies on a default installation of MySQL in the same machine (e.g. the expected port is 3306). In case the server is on a network machine or has a non-standard installation, it needs to be specified when launching openSF through the command line:

```
~/openSF$ ./openSF --dbadress server:3307
```

Refer to section 3.5.1 for further details on Database-related arguments.

3.3.3.1.1. Linux Installation

The most popular Linux distributions (SUSE, Ubuntu, RedHat, Debian) include an automatic system for downloading and installing software. The MySQL installation method in case the Linux distribution provides a software package manager may be as simple as using the particular package manager (Yast in OpenSUSE case for example) and searching for the package for the MySQL server.

In case the Linux distribution does not provide an online package manager it is suggested to visit the MySQL website and look for the download for the distribution in question, or as a last resort the “Linux-Generic” packages, in particular the “compressed TAR archive for x86-64”.

3.3.3.1.2. macOS Installation

MySQL installation package for macOS can be obtained from MySQL official website. It is recommended to download the DMG archive. MySQL installation is based on a GUI installer with default configuration, for which detailed instructions can be found in the MySQL web documentation.

As with Linux, there is also a "compressed TAR archive" that can be used simply by uncompressing the downloaded file and following the MySQL documentation. This may be useful if the system user does not have administrative privileges, required for the DMG-based installation.

3.3.3.1.3. Windows Installation

MySQL installation package for Windows can be obtained from MySQL official website. It is recommended to download the Windows installer package. MySQL installation is based on a GUI installer with default configuration, for which detailed instructions can be found in the MySQL web documentation.

There is also a "compressed ZIP archive" that can be used simply by uncompressing the downloaded file and following the MySQL documentation. This may be useful if the system user does not have administrative privileges, required for the MSI-based installation.

3.3.3.2. JRE Installation Guide

3.3.3.2.1. JRE OracleTM Linux Installation

Most popular Linux distributions include a Java Runtime Environment as part of its default package. Usually this JRE is the one included in the Open Java Development Kit (http://openjdk.java.net/). The openSF framework is only officially supported with the OracleTM JRE, so it may be needed to install it and set it as default JRE instead of the openJDK one.

OracleTM Java Runtime Environment is included in the software repositories of the most popular distributions. Please check the distribution documentation about how to install new software from official repositories. There is also a standalone JRE OracleTM version for Linux operating systems available. It consists in a tarball file, generic for all distributions, whose installation method is as simple as uncompressing it in the desired destination folder. It includes all the necessary files to run java applications through the Java Virtual Machine. This package can be downloaded from the official Java OracleTM web page.

Note that after the installation of the Oracle Java the JAVA_HOME environment variable must point to the folder location of the JRE (e.g. `export JAVA_HOME=/usr/java/jdk1.8.0_131`).

3.3.3.2.2. JRE macOS Installation

Some old versions of Mac OSX include a Java 6 runtime provided by Apple. Use of a Java 8 Runtime Environment from Oracle is required. For further details please refer to the official Java OracleTM web page: https://www.java.com/ .

3.3.3.2.3. JRE Windows Installation

Windows 10 does not come with any Java Runtime Environment installation. Thus, the user shall download and install Java from within the OracleTM web page: https://www.java.com/ .

3.3.3.3. Remote execution installation guide

Remote execution in openSF relies on mounting a remote file system through sshfs. To enable this solution some pre-requisite software needs to be installed before openSF remote execution orchestration.

3.3.3.3.1. Sshfs Linux installation

The sshfs installation method in case the Linux distribution provides a software package manager consists of installing the following packages: sshfs, fuse-utils. In case the Linux distribution does not provide an online package manager it is suggested to visit sshfs website (https://github.com/libfuse/sshfs) and look for alternative installation methods.

3.3.3.3.2. Sshfs macOS Installation

Sshfs installation package for macOS can be obtained from OXS Fuse official website (https://osxfuse.github.io/). It is recommended to download the DMG archive. The stable releases of both OSXFuse and SSHFS should be installed. Installation is based on a GUI installer with default configuration.

3.3.3.3.3. SSH access permission installation

To ease the access to remote file system thru sshfs it is required to enable access by sharing ssh keys (so it is not required writing the password every time the connection is established).

The following commands implement the sharing of ssh keys between the participating computers:

```
~/$> ssh-keygen -t dsa
```

followed by:

```
~/$> ssh-copy-id -i .ssh/id_rsa.pub <user>@<machine>
```

For the above configuration the following packages are required: ssh-keygen, ssh-copy-id. Note that ssh-copy-id is not an officially OSX supported package so either an unofficial installer can be used (e.g. brew) or the public key needs to be copied manually.

Note that the Windows OpenSF does not support remote execution.

3.3.4. Module pre-requisites

In most cases, openSF runs modules by invoking them as binaries from the simulations folder. This means that it is the responsibility of the module developer/integrator to provide any dependencies (e.g. libraries, interpreters, etc.) and to perform the setup needed for such modules to work correctly. The means are varied and depend on the type of module (compiled binary, script, etc.).

For example, a module written in C++ may link statically against all its dependencies so that the resulting executable does not depend on any dynamic libraries that would have to be found and loaded on start, although this may not always be possible depending on the library and the system. Alternatively, dependencies can be provided in a way that may be located by the module, so e.g. a Python module that makes use of OSFI-Python or other libraries may access them by:

- ❑ Installing them in the site-packages directory of Python,
- ❑ Including them in the PYTHONPATH environment variable, or
- ❑ Distributing them alongside with the module, making the script itself look for them in a known location relative to the module (e.g. `../libs/OSFI/Python`).

For more information on the matter of module development and deployment, look at the documents mentioned in §3.6.

3.3.4.1. IDL

Note: IDL module execution is deprecated and not under further development.

To execute modules in IDL with openSF it is necessary to have installed IDL software on the computer.

openSF has been tested with the following versions of this software: version 7.1, 8.0 and 8.1.

If the user has a previous version, the application may not work eventually. It is recommended to have installed at least IDL 7.1, and whenever possible version 8.0 or later.

An important requirement for the correct functioning is that IDL is installed in the default path, because if not some features of the OSFI library will not work properly. This problem is related with ConFM module, which uses some internal classes of IDL that must be in the default path, because otherwise the application does not find them. This is caused because IDL looks for these classes only in the default directory, and if it does not find them generates an error.

For IDL 7.1 the default path is '/usr/local/itt/idl' and for IDL 8.x the default path is '/usr/local/itt/idl/idl'.

Furthermore, IDL provides three types of licenses according to the user needs, as can be seen below:

- ❑ IDL development: Full license for IDL that allows to the user to use all its functionalities. Users can access to the IDL Development Environment, the IDL command line, and having the ability of compiling and executing IDL .pro files and executing .sav files.
- ❑ IDL runtime: Allows executing IDL programs precompiled and saved as .SAV files, or .pro files without any type of restriction.
- ❑ IDL virtual machine: It is a free license that allows to the user to execute IDL programs precompiled and saved as .SAV files, or .pro files. This kind of license has a few restrictions, like displaying a splash screen on start-up, callable IDL applications are not available.

To execute a .sav or a .pro file without any type of restriction it is necessary to have installed the development license or the runtime license on the computer. If user wants to generate .sav files by compiling .pro files, it is mandatory to have the development license. If the user only has the virtual machine license, he can execute .pro and .sav files but with restrictions, as many functionalities are not available for this type of license.

3.3.4.2. Matlab

To execute modules in Matlab with openSF, Matlab software must be installed on the computer. The only requirement is that Matlab version must be R2009a or later, with the corresponding license.

3.3.4.3. Python and other scripts

The framework also executes script-based modules, such as Python or shell scripts. Currently, they are invoked as normal programs, so they must be marked with execution permissions. The choice of interpreter must be somehow recognizable for the system e.g. with the customary hash-bang line:

```
#!/bin/sh                                             (POSIX shell scripts)
#!/bin/bash                                                 (BASH scripts)
#!/usr/bin/env python                        (Python, system default version)
#!/usr/bin/env python2/3                   (Python, script choice of a version)
#!/opt/bin/PowerShellCore6/pwsh             (Other, custom script interpreters)
```

Since .sh/.bat files are executed directly, Windows users cannot run POSIX shell modules and Linux/Mac users cannot run CMD batch modules. However, the system is extensible and new “interpreted file” types can be added in future versions.

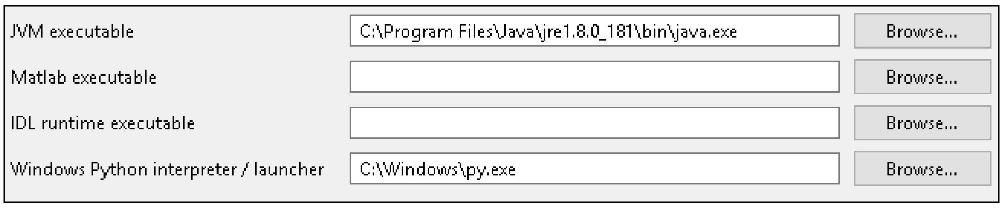
3.3.4.3.1. Python scripts execution in Windows

Due to the Python2/Python3 ambiguity (some scripts are compatible between both versions and some are not), a method to choose between the correct interpreter for each script is necessary. Giving the version of the interpreter to be used via shebang lines in the beginning of the Python script solves the issue.

Since Windows does not allow shebang lines to choose between which Python version to use, the solution is to use a launcher[3] for aiding in the location and execution of different Python versions, allowing the scripts to indicate a preference for a specific Python version.

Thus, in Windows, the Python interpreter must be specified to run Python modules. If available, the user shall use the aforementioned launcher/chooser “py.exe” instead of a specific interpreter version i.e. “python.exe”.

This Python interpreter, as well as others interpreters from the compatible languages, can be chosen by accessing Systems->Preferences->Application Settings, as depicted in Figure 3-1.



JVM executable	C:\Program Files\Java\jre1.8.0_181\bin\java.exe	Browse...
Matlab executable		Browse...
IDL runtime executable		Browse...
Windows Python interpreter / launcher	C:\Windows\py.exe	Browse...

Figure 3-1: Windows interpreters chooser within OpenSF.

3.4. How to Install the Framework

Provided that every pre-requisite is fulfilled, users can now proceed to install the application.

The openSF distribution package consists of an installer for each target platform. The installer will be in charge of the system deployment and the pre-requisites checking.

The installer, together with the deployment of the framework, shall also execute a SQL script in order to create the default MySQL database. Additionally, an extra database could be also installed if the user desires to install the test suite.

The system pre-requisites checking performed by installer are the followings:

- ❑ Oracle Java JRE 1.8: installer check the presence of the Java Runtime Environment 1.8 or later.
- ❑ MySQL server 5.7.X.

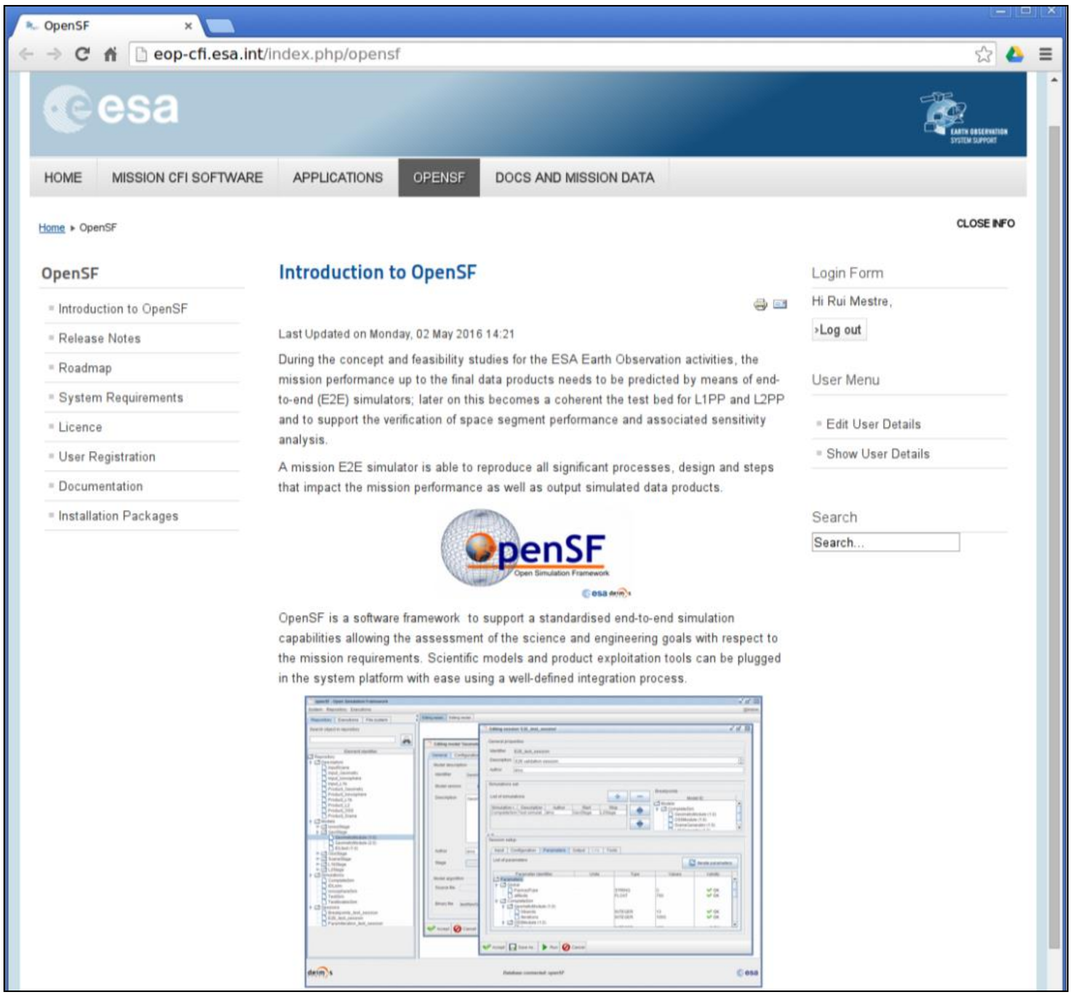
[3] https://docs.python.org/3/using/windows.html#python-launcher-for-windows

3.4.1. Installer Guide Setup

Installers are built with the Install4J software. Install4J is a multi-platform GUI installer designed to be completely cross-platform.

To download the openSF software and documentation perform the following steps:

1. If not already done, register as a user on http://eop-cfi.esa.int/ (see "Create an account" link in right pane). The registration is free;
2. If not already done, register as openSF user at http://eop-cfi.esa.int/index.php/openSF/openSF-registration (see Figure 3-2);
3. Download the Software at http://eop-cfi.esa.int/index.php/openSF/download-installation-packages



OpenSF
eop-cfi.esa.int/index.php/opensf

HOME | MISSION CFI SOFTWARE | APPLICATIONS | OPENSF | DOCS AND MISSION DATA

Home ▸ OpenSF

CLOSE INFO

OpenSF
- Introduction to OpenSF
- Release Notes
- Roadmap
- System Requirements
- Licence
- User Registration
- Documentation
- Installation Packages

Introduction to OpenSF

Last Updated on Monday, 02 May 2016 14:21

During the concept and feasibility studies for the ESA Earth Observation activities, the mission performance up to the final data products needs to be predicted by means of end-to-end (E2E) simulators; later on this becomes a coherent the test bed for L1PP and L2PP and to support the verification of space segment performance and associated sensitivity analysis.

A mission E2E simulator is able to reproduce all significant processes, design and steps that impact the mission performance as well as output simulated data products.

OpenSF is a software framework to support a standardised end-to-end simulation capabilities allowing the assessment of the science and engineering goals with respect to the mission requirements. Scientific models and product exploitation tools can be plugged in the system platform with ease using a well-defined integration process.

Login Form

Hi Rui Mestre,

›Log out

User Menu
- Edit User Details
- Show User Details

Search

Search...

Figure 3-2: openSF Downloads Web

3.4.1.1. Linux Installation

openSF software is available for all the Linux distributions. After downloading the installer corresponding to the machine architecture, users will execute it (either by double clicking on it or by opening a terminal window and executing it) and follow the instructions that appear on the screen.

If the system pre-requisite Java is not met the following error message will be shown.

```
$ ./openSF_linux64_3.0.9.sh
No suitable Java Virtual Machine could be found on your system.
The version of the JVM must be at least 1.8.
Please define INSTALL4J_JAVA_HOME to point to a suitable JVM.
```

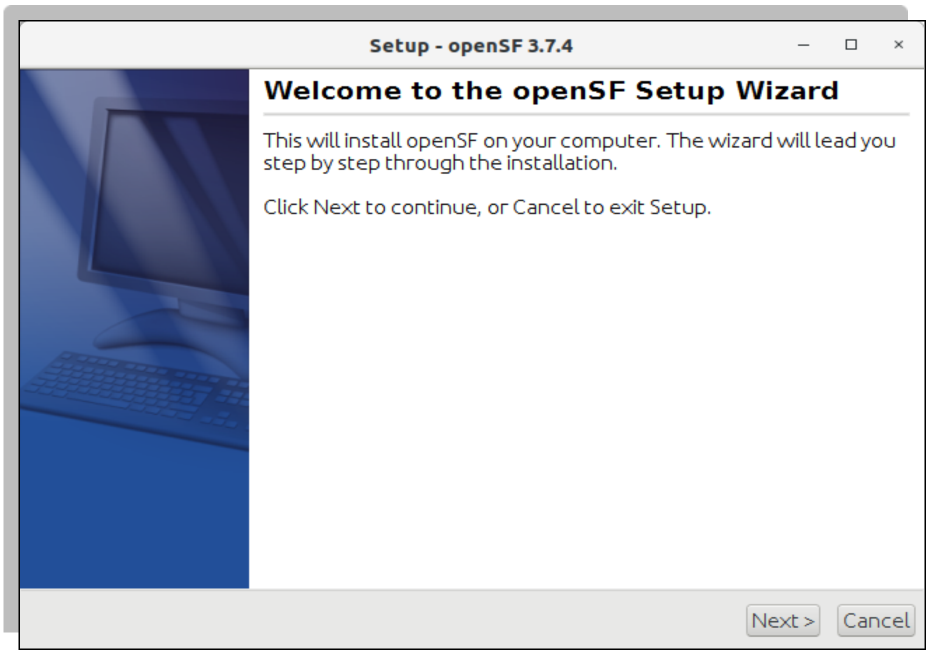
The installer checks the presence of a Java Virtual Machine in the locations defined by any of the following environment variables $JAVA_HOME, $JDK_HOME or $INSTALL4J_JAVA_HOME. Therefore, this error could be fixed by either setting any of those variables to the correct location of the JVM in the system with the *export* command and then running the installer

```
$ export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
$ ./openSF_linux64_3.9.2.sh
Starting Installer ...
```

or by prepending the definition of any of those variables to the installer execution command.

```
$ JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/ ./openSF_linux64_3.9.2.sh
Starting Installer ...
```

If the JVM location is correctly set, the initial window will open when running the installer.


Setup - openSF 3.7.4

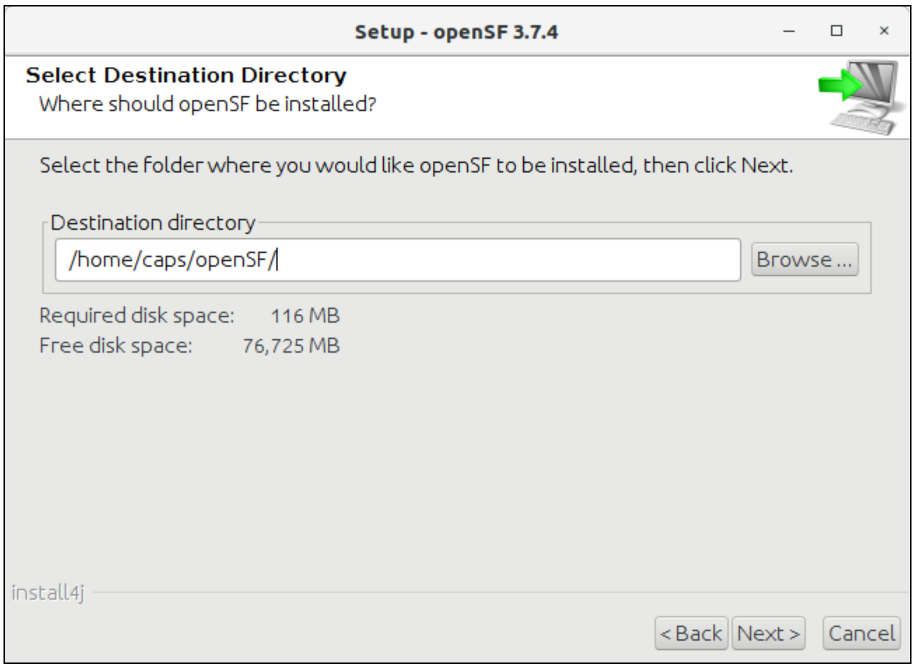
Welcome to the openSF Setup Wizard

This will install openSF on your computer. The wizard will lead you step by step through the installation.

Click Next to continue, or Cancel to exit Setup.

Figure 3-3: Installation confirmation screen

Once the installer has checked the system pre-requisites the user shall select the destination folder to hold the openSF software. The default installation folder is the user home one (/home/<user>/openSF).


Setup - openSF 3.7.4

Select Destination Directory
Where should openSF be installed?

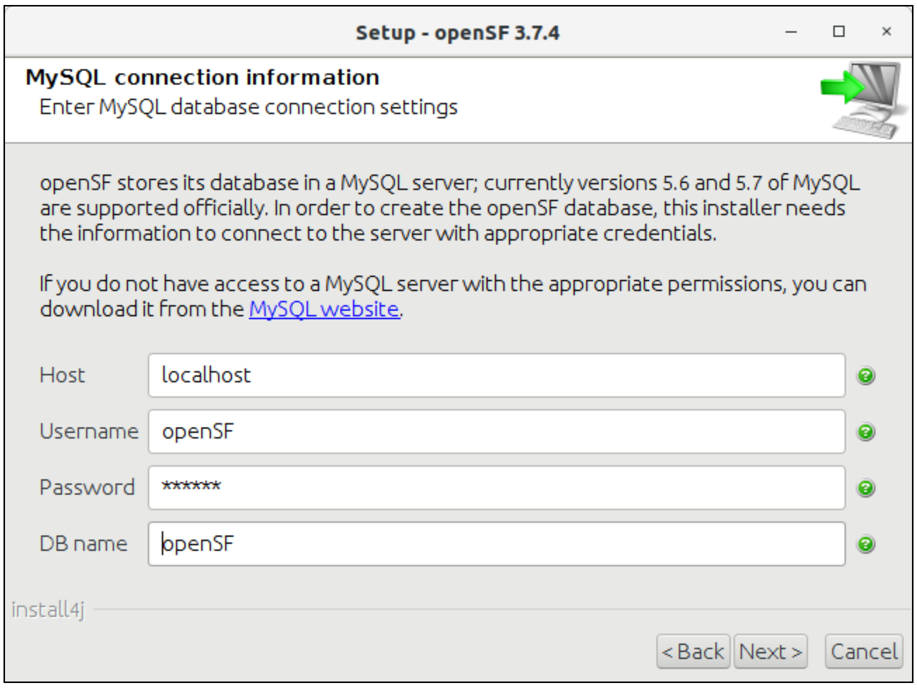
Select the folder where you would like openSF to be installed, then click Next.

Destination directory
/home/caps/openSF/

Required disk space: 116 MB
Free disk space: 76,725 MB

Figure 3-4: Installer folder selection window

After that, the installer needs the connection settings to access the MySQL server. The MySQL user needs to have database-creation privileges and use the MySQL native authentication protocol (use "*mysql_native_password*" as authentication plugin). It shall be noted that MySQL must be also active during the installation and the future use of the tool.


Setup - openSF 3.7.4

MySQL connection information
Enter MySQL database connection settings

openSF stores its database in a MySQL server; currently versions 5.6 and 5.7 of MySQL are supported officially. In order to create the openSF database, this installer needs the information to connect to the server with appropriate credentials.

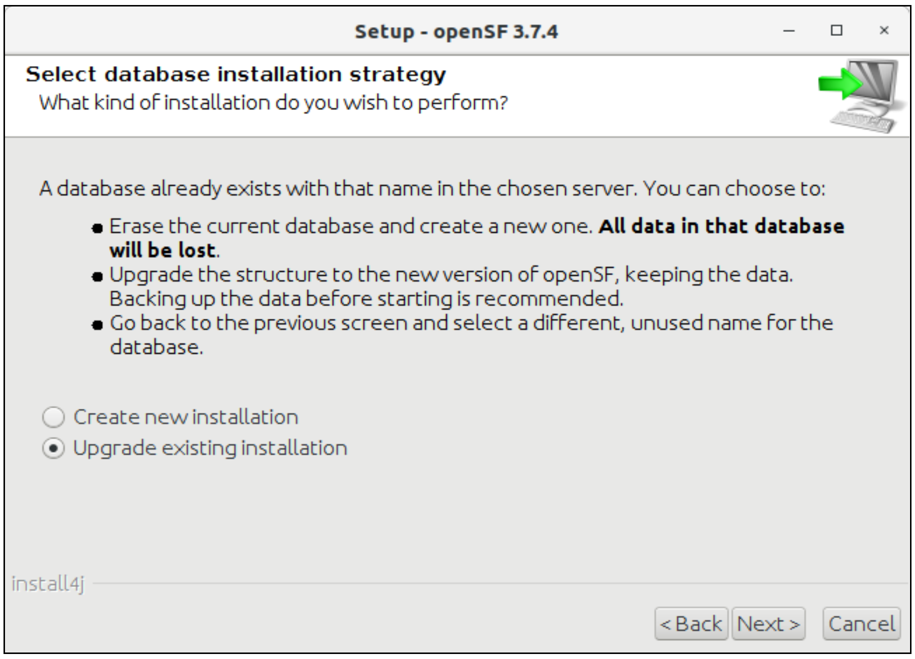
If you do not have access to a MySQL server with the appropriate permissions, you can download it from the MySQL website.

Host	localhost
Username	openSF
Password	******
DB name	openSF

Figure 3-5: Database Login screen

Once the MySQL username and password are checked the user shall input a database name. If the database already exists, the user shall select the installation strategy:

1. Create new installation: this option creates the database structure (tables, columns etc…) from scratch cleaning any existing database contents. It is recommended to back up the database content.
2. Upgrade existing installation: this option preserves the existing database contents. This is the preferred option when upgrading an existing installation of openSF.


Setup - openSF 3.7.4

Select database installation strategy
What kind of installation do you wish to perform?

A database already exists with that name in the chosen server. You can choose to:

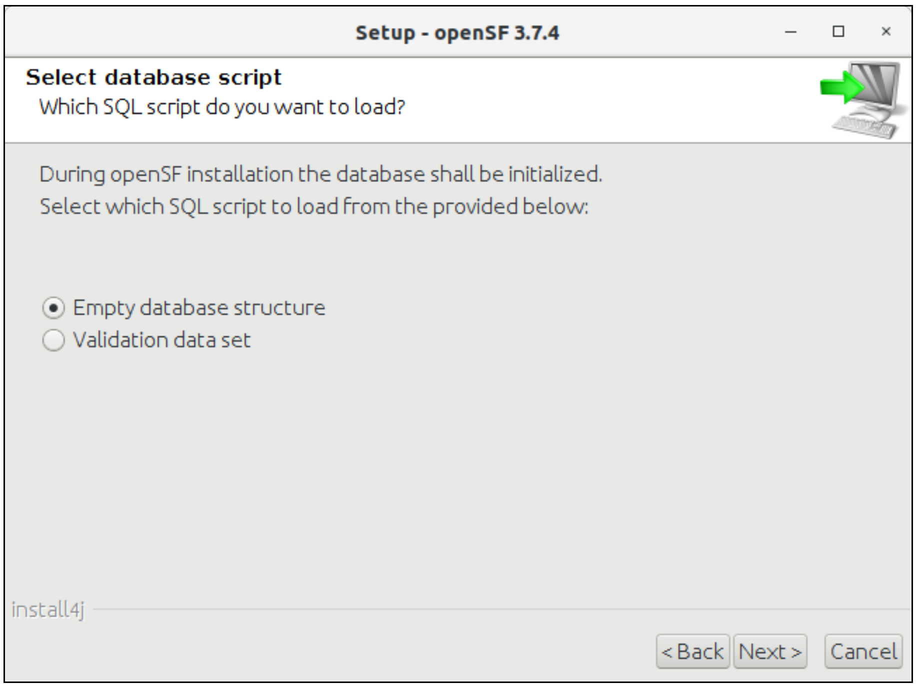
- Erase the current database and create a new one. **All data in that database will be lost**.
- Upgrade the structure to the new version of openSF, keeping the data. Backing up the data before starting is recommended.
- Go back to the previous screen and select a different, unused name for the database.

Create new installation [✓ Upgrade existing installation]

Figure 3-6: Database Installation Strategy

Once the database creation strategy is defined, in case a new database installation is to be created or if the selected database does not exist, the user shall select the initial database structure:

1. Empty database structure: this option creates the database structure (tables, columns etc…) with no content. This is the preferred option if the user is ready to create his first simulation scenario.
2. Validation data set: this option creates the database structure filling the database with a dummy simulation scenario for testing or learning purposes.


Setup - openSF 3.7.4

Select database script
Which SQL script do you want to load?

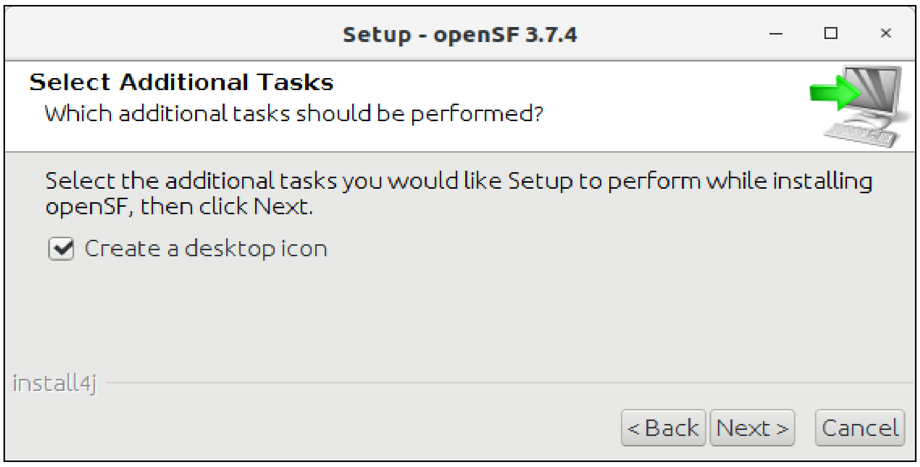
During openSF installation the database shall be initialized.
Select which SQL script to load from the provided below:

[✓ Empty database structure]
Validation data set

install4j

Figure 3-7: Database structure selection dialog

In the next window users shall check that the information is correct and click next to proceed with the software installation.


Setup - openSF 3.7.4

Select Additional Tasks
Which additional tasks should be performed?

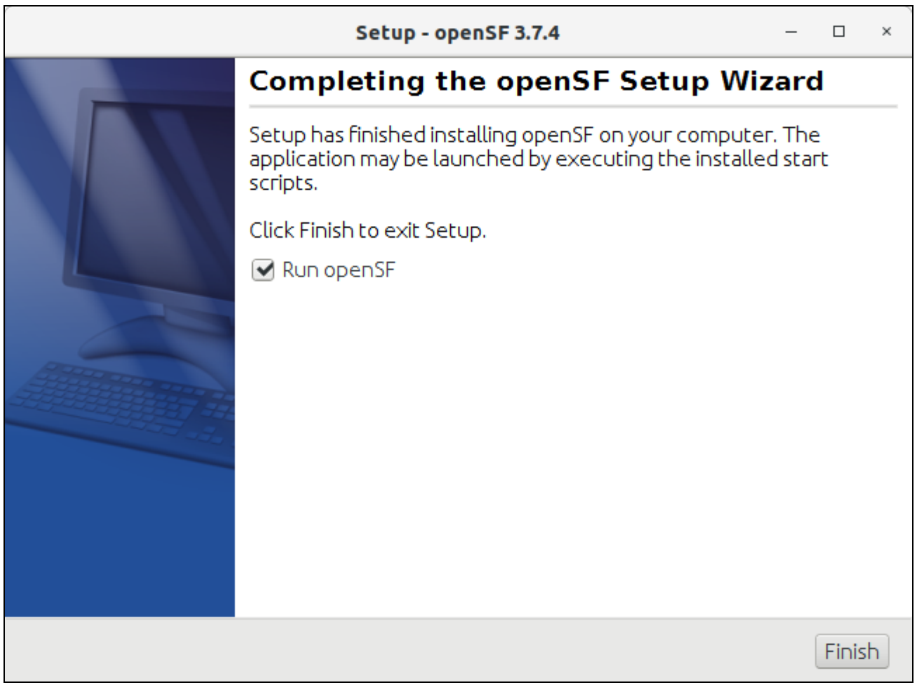
Select the additional tasks you would like Setup to perform while installing openSF, then click Next.

[✓ Create a desktop icon]

install4j

Figure 3-8: Installation icon window

If the installation process has been successful an “Install Complete” dialog shall appear allowing to automatically launch the openSF software.


Setup - openSF 3.7.4

Completing the openSF Setup Wizard

Setup has finished installing openSF on your computer. The application may be launched by executing the installed start scripts.

Click Finish to exit Setup.

Figure 3-9: Installation successful screen

To launch openSF users can either: (a) double-click on the openSF desktop icon[4] or (b) open a terminal window, go to the openSF installation folder and run openSF manually (for further details on launching openSF refer to section 3.5.

3.4.1.2. macOS and Windows Installation

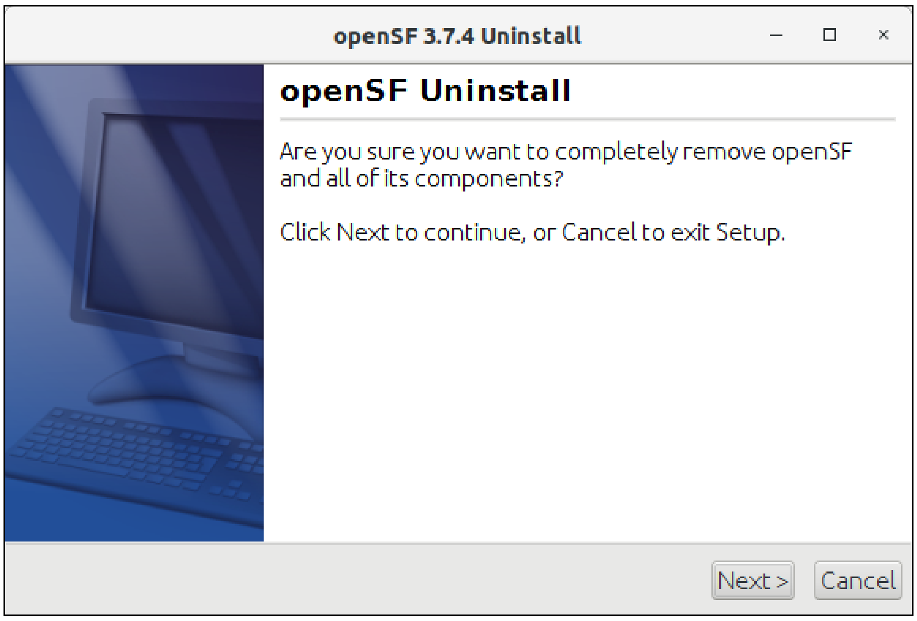
OpenSF installation package for macOS and Windows is provided as a DMG archive and an EXE installer respectively. After opening the respective installer file, openSF installation is based on a GUI installer that guides the user in installing the software. The steps to follow in the installation are the same as for Linux installation (described in section 3.4.1.1 just above).

3.4.1.3. Uninstalling openSF

The installation process places an uninstaller application in the Applications start-up menu.

Note that the uninstall mechanism only removes the application files, the database and user simulation folders are not removed.

[4] The openSF desktop icon is only available if the user so chooses during the installation process


openSF 3.7.4 Uninstall

openSF Uninstall

Are you sure you want to completely remove openSF and all of its components?

Click Next to continue, or Cancel to exit Setup.

Figure 3-10: Uninstall confirmation screen under Linux

3.4.1.4. Folder structure

This section provides a general description of the openSF folder structure and its contents.

Folder name (indented)	Contents
❑ .	openSF home root ❑ "openSF". Starting-up script. ❑ "openSF_updater". Script that updates openSF ❑ "uninstall". Uninstaller ❑ "openSF.properties". The configuration file. ❑ LICENCE file. ❑ Only in Windows distribution: eclipsec.exe, openSF.ini, artifacts.xml.
❑ configuration	Folder that contains Eclipse related files with fixed configuration
❑ data	Folder with a global configuration file template as well as a database template. Inside *xml/import* folder there are xsl files intended to transform old formatted XML files into OpenSF databases.
❑ features, plugins, p2	Folder that contains Eclipse related files
❑ ParameterEditor	Folder that contains the ParameterEditor executable
❑ workspace	Eclipse related folder with user configuration
❑ simulations	openSF simulations root folder

Folder name (indented)	Contents
❑ <simulation_name>	Simulation folder. Every simulation, once executed, has one directory structured as this one. However, if the execution is of type Timeline, Iteration or Batch, there will be a parent folder with several sub-simulation executions located under it. Each normal simulation folder (or timeline/iteration subfolder) will have the simulation script and, if generated, the simulation report. This folder will also have the input and configuration files used by the modules and their outputs.
❑ resources/documentation	Default folder for framework documentation.
❑ test	Test folder for openSF validation scenario. This folder contains the modules binaries for running the test simulation. ❑ *lib*: shared libraries ❑ *bin*: modules binaries ❑ *data*: files used in the examples of the validation database. ○ *batch*: example of configuration file for batch simulations. ○ *conf*: examples of configuration files. This includes local configuration files and a template for the global configuration file the user needs to mandatorily supply manually. ○ *database*: validation database in SQL and XML formats. ○ *perturbations*: sample input files of parameters' perturbations. ○ *simulations*: the configuration and input files used by each of the simulations provided in the validation database. The files are arranged in subfolders with the same name as the simulation that uses them. ○ *timeline*: example timeline scenario file.

Note that on macOS, all *Eclipse related* files are not deployed in the root directory; they are in the app's contents package.

Users can also find useful the guidelines about how to organize the folder structure of a simulation project integrated in openSF. These guidelines are described in section 5.3.

3.4.2. Licensing scheme

openSF uses a licensing scheme that allows integrating it in any kind of third-party developments.

Core library is distributed under the terms of the 'ESA Software Community License - Type 3' as published by the European Space Agency; either version 1.1 of the License, or (at your option) any later version.

A copy of the 'ESA Software Community License - Type 3' is distributed with openSF, or can be found at http://eop-cfi.esa.int/index.php/docs-and-mission-data/licensing-documents.

3.5. How to Start the Application

The openSF system can be launched (under Windows, macOS and Linux) by: (a) double clicking on the openSF desktop icon[5] or (b) using a command line interface and executing the following command.

```
$ $OPENSF_HOME/openSF                                                  (Linux)
$ open -a $OPENSF_HOME/openSF.app                                     (Mac OS)
$ $OPENSF_HOME\openSF.exe                                            (Windows)
```

If openSF is launched with no parameters, the GUI will show up normally. This behaviour can be modified providing the following parameters:

- ❑ *--execute <simulation_identifier>*. The framework will launch the execution of a previously defined simulation with the defined parameter values. The simulation identifier shall correspond to the one stored into the openSF database.

An example to illustrate the execution in batch mode of a “Radar” simulation is as follows:

```
$ $OPENSF_HOME/openSF --execute Radar                                  (Linux)
$ open -a $OPENSF_HOME/openSF.app --args --execute Radar              (Mac OS)
$ $OPENSF_HOME\openSF.exe -execute Radar                             (Windows)
```

This form will execute the openSF in text mode and find a simulation named “Radar”. Then it will execute it, intercepting all the events and storing the results in the database. Then, the system will stop.

3.5.1. Database-related arguments

If the openSF is launched with no parameters it will use the default configuration of openSF connection to MySQL, which is the one that was last used successfully.

The following command line arguments can be used to alter this configuration:

- ❑ *--database*. The MySQL database supporting openSF;
- ❑ *--user*. The MySQL server user name for accessing openSF supporting data;
- ❑ *--password*. The MySQL server password for accessing openSF supporting data;
- ❑ *--dbaddress*. The MySQL server location in the network (e.g. localhost or 100.100.0.23:3307).

Note that command-line options **must** use double dashes. Since openSF is built on the Eclipse RCP technology, options with a single dash *may* be intercepted by the Eclipse launcher code, not reaching the openSF application code at all. This is the case for e.g. the “user” option.

3.5.2. Environment variables

After the openSF installation has been successfully performed, at the first openSF execution the user may optionally be required to access the “System Configuration” module and adapt the default environment variables to match the distribution.

[5] The openSF desktop icon is only available if the user so chooses during the installation process

3.6. Developing modules for openSF

This section is aimed to module developers that are looking for further information about openSF module integration.

Below these lines the list of openSF documents with relevant information about module developing and the topics covered by each one.

- openSF Interface Control Document [AD-ICD]
  - openSF interface issues
  - Module development guidelines
  - Module development process
- OSFI Developer's Manual [RD-OSFI-DM]
  - Integration libraries reference manual for each programming language.
- OSFEG Developer's Manual [RD-OSFEG-DM]
  - Error Generation Libraries reference manual.
- openSF Architecture Design Document [AD-ADD]
  - OSFI architecture
  - Interaction between modules and openSF

4. OPENSF - REFERENCE MANUAL

This section provides a detailed description of all the elements that conforms the openSF graphical user-interaction. Note that all the screenshots are shown only for the Linux platform.

4.1. Main window

In this section the look-and-feel, operational behaviour and design features common to the openSF HMI are presented.

The HMI accepts input via devices such as the computer keyboard and mouse and provides articulated graphical output on the display. The openSF HMI approach has been chosen because of its flexibility, as it lets users organize the layout of the information as desired, showing only relevant windows and in the way users want. The layout consists of a "parent" container that can host inside several "internal frames". These internal frames are intended to present independent modules of the simulator. For example, each time the user wishes to perform operations with the list of modules of the system, a "module manager" frame will pop-up inside the bounds of the main window listing the list of modules currently available within openSF.

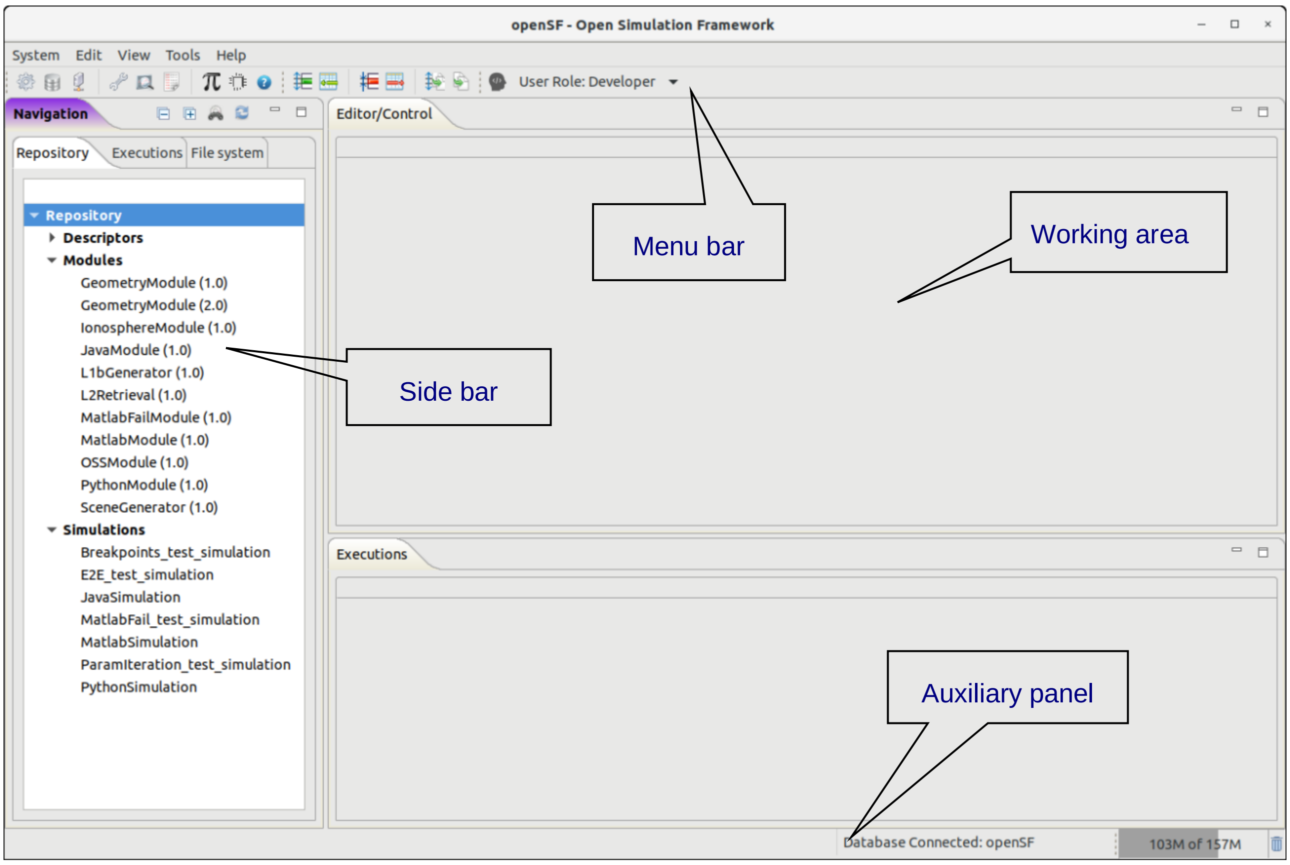


Figure 4-1: Main window appearance

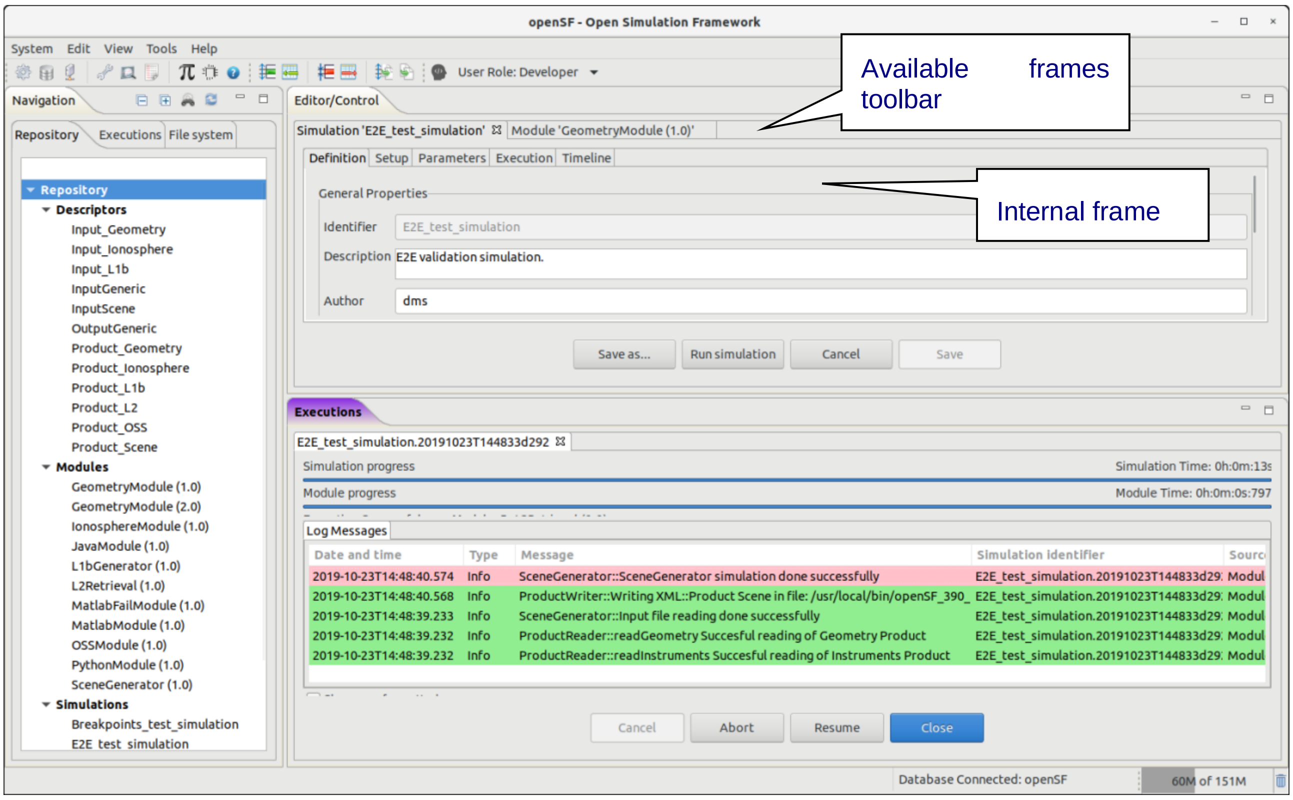
All the windows have common operations to help their usability: main window, internal frames or dialogues can be closed, resized, maximized or minimized to fit the user's needs.

This main window shown in Figure 4-1 includes a menu bar to provide keyboard and mouse access to the simulator main functions as well as functions regarding frames management and application basis.

Occupying the central and main region there is a working area. This area is where all internal frames are going to be created and main interaction is held. Besides that, this working area implements a "scrollable" panel in order to easily navigate through frames surpassing its bounds.

At the left side of the working area there is a system objects navigator, a "side bar" aiming to provide a quick access method to every item known by the system: repository of descriptors, modules and simulations, the list of simulation execution results and also a file system browser to navigate through the contents of the application's directory.

The main window's footer area shows application status information.



Identifier: E2E_test_simulation
Description: E2E validation simulation.
Author: dms

Simulation progress — Simulation Time: 0h:0m:13s
Module progress — Module Time: 0h:0m:0s:797

Date and time	Type	Message	Simulation identifier	Sourc
2019-10-23T14:48:40.574	Info	SceneGenerator::SceneGenerator simulation done successfully	E2E_test_simulation.20191023T144833d29	Modul
2019-10-23T14:48:40.568	Info	ProductWriter::Writing XML::Product Scene in file: /usr/local/bin/openSF_390_	E2E_test_simulation.20191023T144833d29	Modul
2019-10-23T14:48:39.233	Info	SceneGenerator::Input file reading done successfully	E2E_test_simulation.20191023T144833d29	Modul
2019-10-23T14:48:39.232	Info	ProductReader::readGeometry Succesful reading of Geometry Product	E2E_test_simulation.20191023T144833d29	Modul
2019-10-23T14:48:39.232	Info	ProductReader::readInstruments Succesful reading of Instruments Product	E2E_test_simulation.20191023T144833d29	Modul

Database Connected: openSF   60M of 151M

Figure 4-2: Main window appearance showing internal frames and scroll bars

The HMI provides a menu bar (Figure 4-3) at the upper side of the main frame to show some capabilities of the system. Below is also a toolbar for quick access to critical actions.

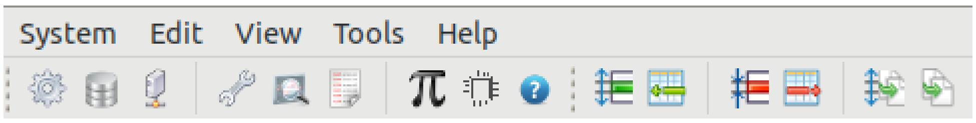


Figure 4-3: Detail of main menu bar

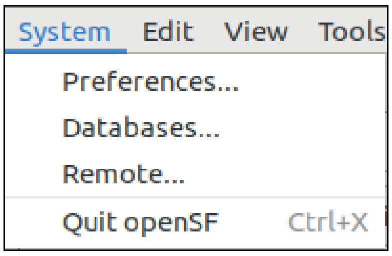


Figure 4-4: Detail of a menu, showing menu items

It is shown in Figure 4-4 that a menu item is composed of an icon which graphically describes the function, the name of the function and a "quick access" key combination. Users can quickly access this functionality pressing this key combination or the letter underscored in the function name while the menu is rolled down.

There are also some contextual or "pop-up" menus that users can access by clicking the right button of the mouse while over certain controls[6]. These "pop-up" menus have the same appearance as menus rolling down the menu bar.

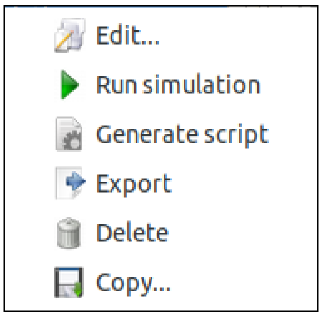


Figure 4-5: Detail of a contextual menu

It can be seen in Figure 4-5 that a pop-up menu acts exactly like a menu at the main frame. They also provide mouse and keyboard access to certain capabilities.

4.2. Generic Functions, dialogues and displays

This section is meant to describe the design of generic functions, dialogs and displays used by the HMI.

There are some functionalities of the HMI that show a "file chooser" dialog as shown in Figure 4-6.

[6] In macOS the contextual menu behaviour may depend on setting the correct gesture for Bluetooth mouse and track pad: the "Secondary click" gesture (e.g. "Click with two fingers" in track pad or 'Click on right side" in Bluetooth mouse) should be applied to allow using the options in the contextual menus.

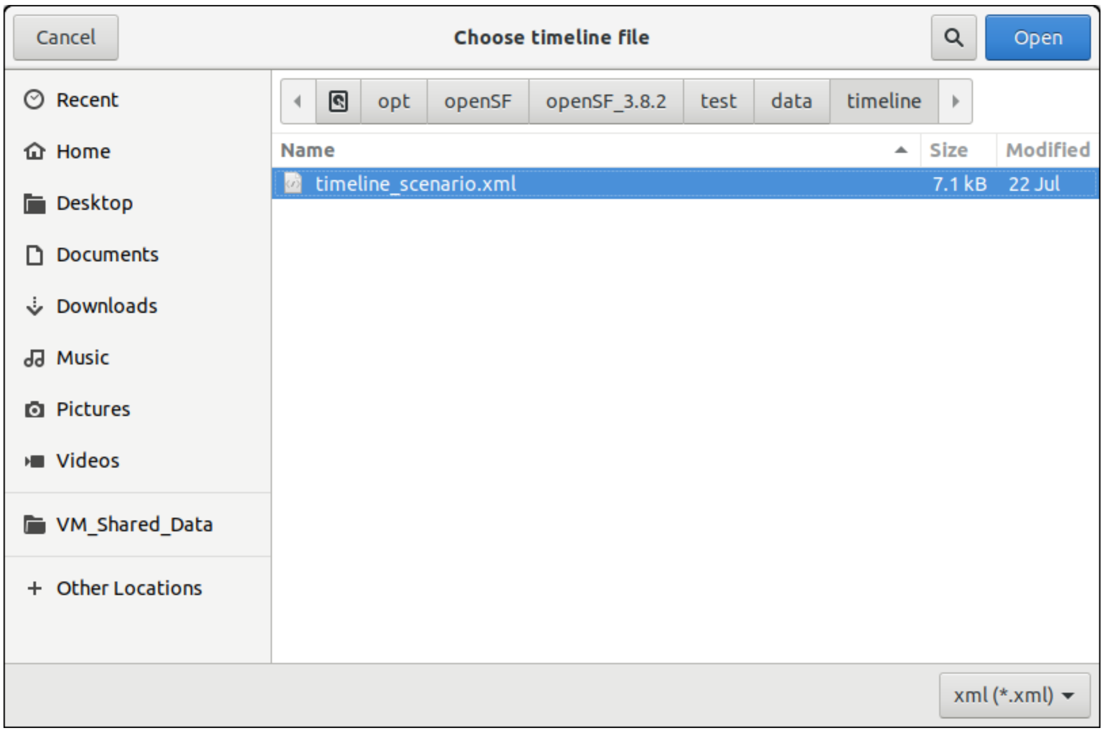
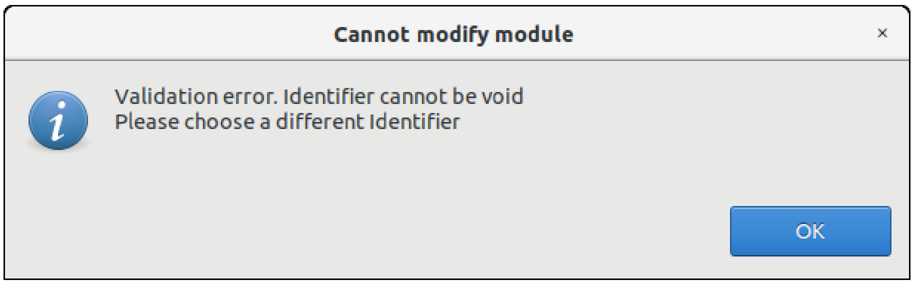


Figure 4-6: File chooser dialog

This dialog allows the user to browse the system directory to select a certain file or list of files. It provides sorting, filtering and file operations.

Throughout the openSF HMI some functionality may show information to the user and might ask for some input in response to an answer. The HMI will present some "modal" dialogs that will get the system focus until the user provides an answer. These modal dialogs will block the input to other areas of the application until a response is given.


Cannot modify module

Validation error. Identifier cannot be void
Please choose a different Identifier

Figure 4-7: Dialog example

These dialogs will typically provide a message with an "OK" button or give a yes-or-no question or another question with different options. The dialogs will provide information with a clear description of the event.

4.3. Frame management

Accessing to the “Window” menu of the main menu bar the user can find all the functionality provided for the frame management. Option ‘Reset Views’ allows restoring the original window properties (as defined when installing openSF)

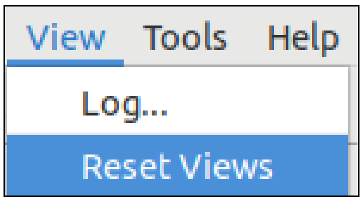


Figure 4-8: Frame management menu

Other frame management functionalities can be found in the header of every main frame (not in dialogues).

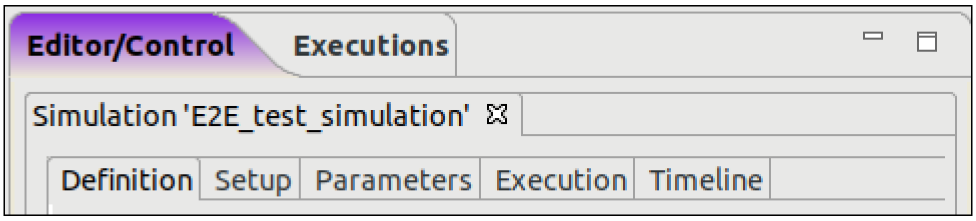


Figure 4-9: Internal frame header

Note in Figure 4-9 two little icons at the right border of the header. Those are the “minimize” and “maximize” operations.

If the user minimizes a frame it disappears from the working area but it can be restored from the “available frames toolbar”.

4.4. Side bar

On the left side of the main frame there is side bar, grouping different views of the openSF areas: Repository, Executions, and File system.

As can be seen at the right-upper corner of Figure 4-11, the usual buttons are available to minimize or maximize the side bar. The dotted bar can also be dragged to dynamically change its width.

© DEIMOS Space S.L.U., 2019

4.5. System

Selecting the "System" menu the users can access to the following functionalities:

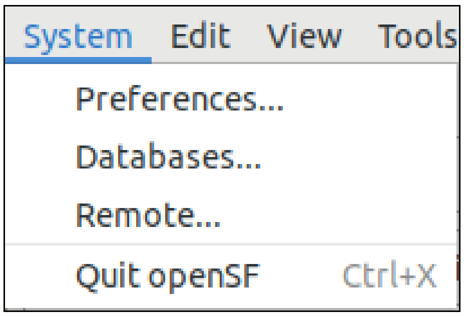


Figure 4-10: System menu

These functionalities are given to control the general characteristics of the whole openSF system.

The repository will show only the elements containing a given substring of characters written in this text box.

Elements are structured into the repository first by element type (descriptors, modules and simulations) and then by family.

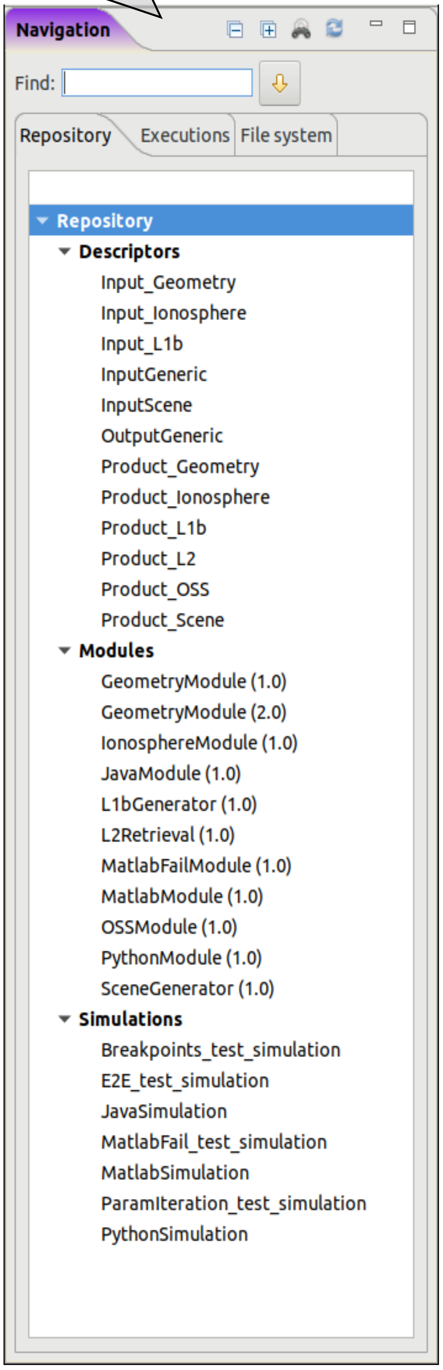
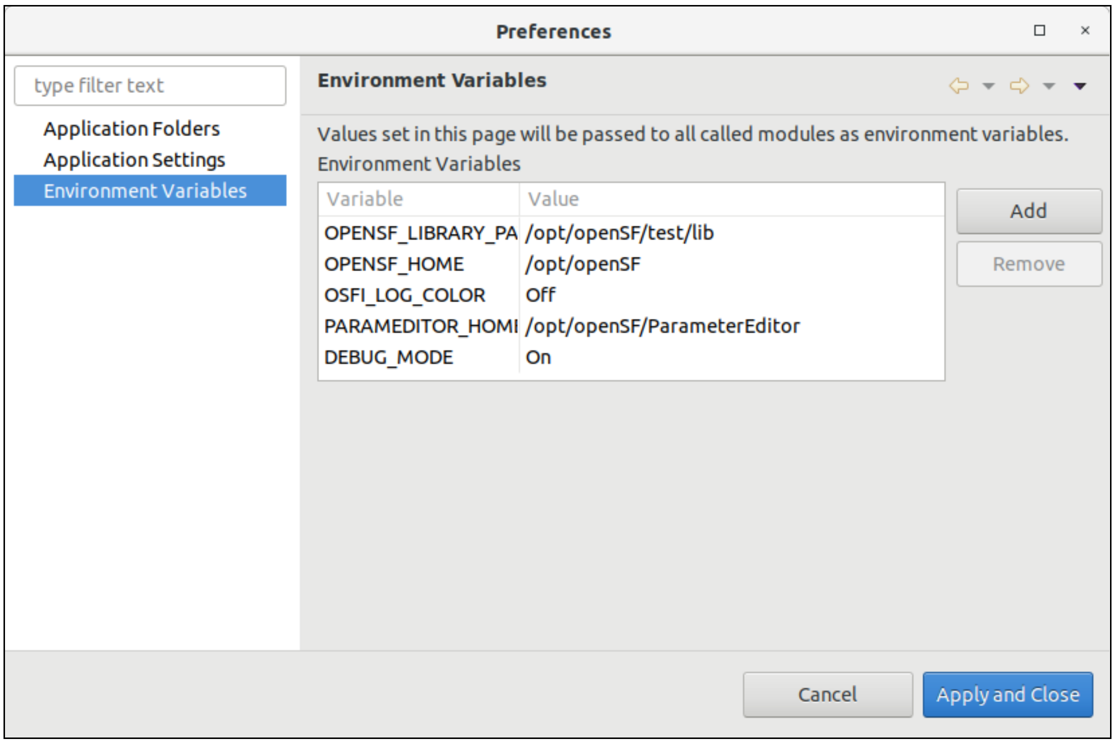


Figure 4-11: Side bar

4.5.1. Preferences

Selecting this menu option, the dialog shown in Figure 4-12 will show up.



Environment Variables

Values set in this page will be passed to all called modules as environment variables.

Environment Variables

Variable	Value
OPENSF_LIBRARY_PA	/opt/openSF/test/lib
OPENSF_HOME	/opt/openSF
OSFI_LOG_COLOR	Off
PARAMEDITOR_HOMI	/opt/openSF/ParameterEditor
DEBUG_MODE	On

Figure 4-12: System preferences - Environment Variables

In this dialog users can modify the following characteristics of the system:

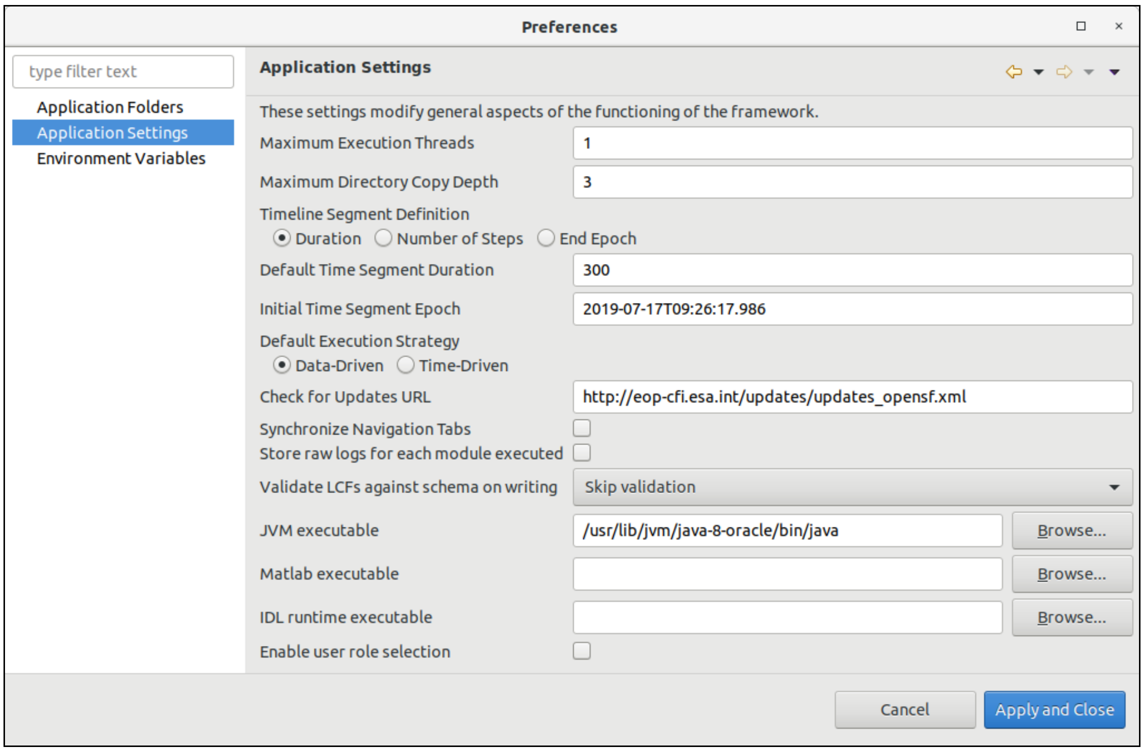
- **Environment variables (Figure** 4-12**)**. A list of environment variables and associated values are shown in this table. Once a module or a product tool is being executed, they can access these variables if they need them because the system makes them available to the execution process. Users can “add” or “remove” an environment variable using the given buttons. Double-clicking on an already existing variable the user can edit its name and value.

  One mandatory variable is $OPENSF_HOME (as explained previously). This variable points to the openSF base folder location. There is an environment variable recognized by OSFI called “DEBUG_MODE” that controls the verbosity of some module executions. Setting to “On” or “Off” can enable or disable this output.

  OpenSF handles in a special way all environment variables used to customize the search for dynamic libraries. For %PATH% (on Windows), $LD_LIBRARY_PATH (on Linux) and $DYLD_LIBRARY_PATH (on macOS), OpenSF prepends the value specified in openSF preferences to the value exported externally in the environment.

  macOS users must be aware of the limitations in using environment variables in script execution enforced by Apple with the introduction of the System Integrity Protection (SIP) security feature in OS X 10.11 (El Capitan), which does not allow critical environment variables (such as $DYLD_LIBRARY_PATH) to be passed in a cascading shell script call.

  For all other environment variables, if a value is specified in openSF preferences, then it overrides the one defined in the system.



These settings modify general aspects of the functioning of the framework.

Maximum Execution Threads	1
Maximum Directory Copy Depth	3
Timeline Segment Definition	[✓ Duration] Number of Steps End Epoch
Default Time Segment Duration	300
Initial Time Segment Epoch	2019-07-17T09:26:17.986
Default Execution Strategy	[✓ Data-Driven] Time-Driven
Check for Updates URL	http://eop-cfi.esa.int/updates/updates_opensf.xml
Synchronize Navigation Tabs	☐
Store raw logs for each module executed	☐
Validate LCFs against schema on writing	Skip validation
JVM executable	/usr/lib/jvm/java-8-oracle/bin/java
Matlab executable	
IDL runtime executable	
Enable user role selection	☐

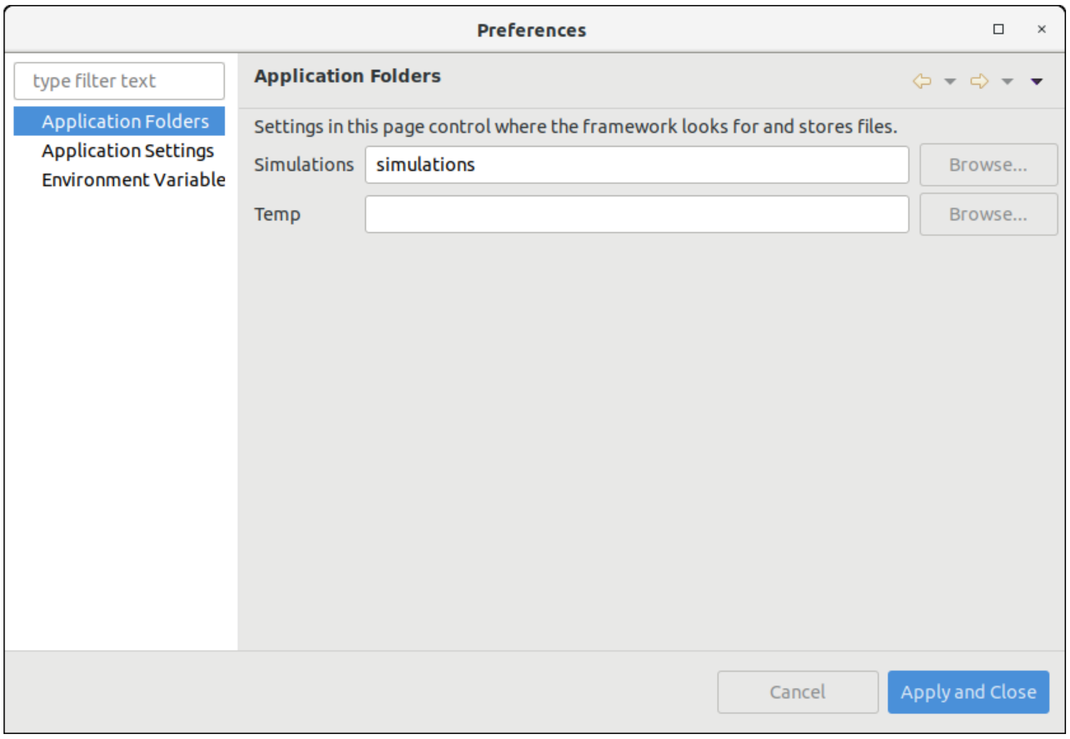
Figure 4-13: System preferences - Application Settings

- **Application settings (**Figure 4-13**).** Users can change the default system parameters:
  - *Maximum Execution Threads*. Sets the maximum number of modules that can be executed in parallel during a simulation execution. The recommended value corresponds to the number of cores of the machine were openSF is installed. Possible values are: 0 (number of cores of the machine), 1 (no parallelisation enable) or N (the number of modules that may be executed in parallel). Keep in mind that it is allowed to insert a maximum number of execution threads higher than the number of cores of the machine and the impact is that a core may have to deal with more than one thread. Nevertheless, a warning message will be shown to the user whenever entering a value higher than the machine's number of cores.
  - *Maximum Directory Copy Depth*: Defines the maximum depth of the directory tree used when copying files and folders before a simulation is executed. This copy operation typically includes copying input and configuration files/directories into the simulation folder. The use of this value is widespread and is designed to avoid possible infinite loops caused by symbolic links.
  - *Timeline Segment definition*.
  - *Default Time Segment duration:* Default duration of the new Time Segments in seconds.
  - *Initial Time Segment Epoch:* Initial Epoch for new Timeline Scenarios. This field shall be compliant with the CCSDS ASCII Time Code A format (`YYYY-MM-DDThh:mm:ss.ddd`).
  - *Default execution strategy*.
  - *Check for Updates URL*. The configurable URL where openSF looks for software updates.
  - *Store raw logs for each module executed*: if disabled, a single log file will be generated for each executed simulation. The file contains general status messages, plus all OSFI-formatted messages from all modules. However, if this option is enabled, a log file will be created in the

same folder containing the "raw" output (both OSFI and non-OSFI formatted) for each module in the simulation. This may be useful for debugging some modules.

- *Validate LCFs against schema on writing*: The user can validate the Local Configuration Files (LCFs) against a schema file. "Skip validation" is selected by default, but the user can select "Validate but only generate warnings" or "Validate and fail the simulation on errors". As the description indicates, if the user chooses "Validate but only generate warnings", the LCFs will be validated against the XSD schema file chosen in the Module menu (section 4.8) and warnings will be generated in the simulation log. If instead the user selects "Validate and fail the simulation on errors", the validation against the XSD schema file will raise errors which will force immediate termination the simulation's execution.
- *Executables for the several compatible languages*: In order to run possible Java, Matlab, etc, executables within the available simulations, the user needs to specify the system path for each of the language's executable for OpenSF to find.
- *Enable user role selection*: When enabled, a custom menu is shown in the toolbar to allow switching between "Normal" user mode and "Developer" user mode (see section 4.13).

- **Application folders**. Users can change the default locations for these directories:
  - *Simulations*. This is the place where all the files associated to simulation executions can be found. Execution scripts, report files and, by default, configuration and output files generated are going to be stored here.
  - *Temporal*. Some intermediate files are going to be stored temporarily in this directory.



Settings in this page control where the framework looks for and stores files.
Simulations: simulations
Temp:

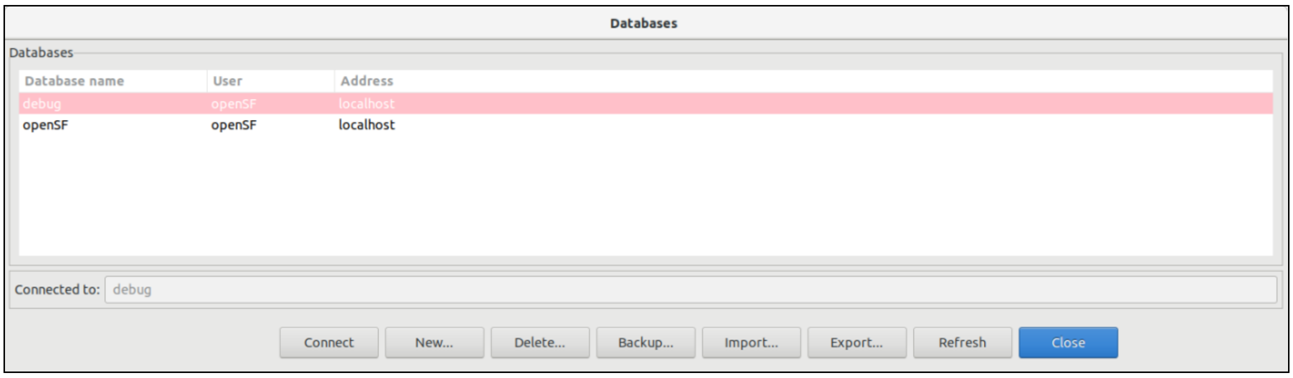
Figure 4-14: System preferences - Application Folders

All around this configuration dialog and the rest of the openSF system (except those places where something else is specified), the user can choose to input the absolute path of a certain file or folder or the path relative to $OPENSF_HOME.

4.5.2. Databases

openSF has the ability of working with a multi-repository. This implies that users can create different databases in the database server, so that all of them are independent and do not share any table. Users can work with different input data by selecting only the corresponding database in the repository. If the user changes input data, only the database that is load in that moment is modified, as all databases are independent. Thus, openSF can hold more than one database in one single instance.

When the user selects the “Databases” option from the menu ‘System’, the window shown in the Figure 4-15 will show up.



Database name	User	Address
debug	openSF	localhost
openSF	openSF	localhost

Connected to: debug

Figure 4-15: Database Management window

In the top of the window there are eight buttons, which allow users to perform different actions over the MySQL server. Users can connect to a database that exists in the MySQL server, create a new database, delete an existing database, backup an existing database, import database definitions specified in XML format or refresh the available database list.

The central area of the window shows a list with the databases stored in the MySQL server. Users can obtain information about the database name, the user who created the database, and the location of the server where the database is stored.

The bottom of the window shows a label with the database currently connected to the MySQL server. In this case, the application is connected to the ‘openSF’ database, so all configuration associated with its descriptors, modules and simulations is loaded by the system.

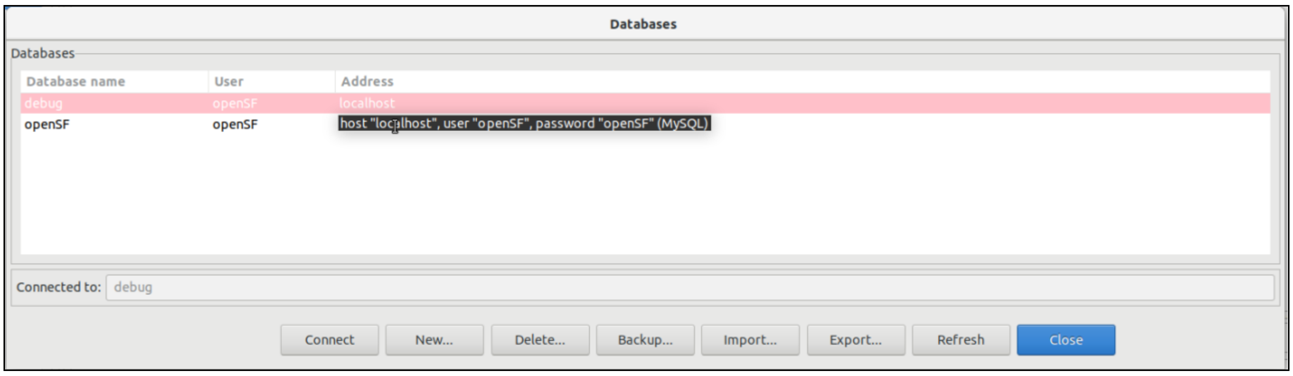
Finally, there is an ‘Close’ button to return to the main window of the application.

4.5.2.1. Connect to a database

Users can switch between the different databases allocated in the repository of the application. For this, the user has to select a database from the list, and click on the “Connect” button.

Automatically the system is connected to it, and then, the name of the selected database is shown beside the label “Connected to”.

An example of this procedure is shown below:


Databases

Databases

Database name	User	Address
debug	openSF	localhost
openSF	openSF	host "localhost", user "openSF", password "openSF" (MySQL)

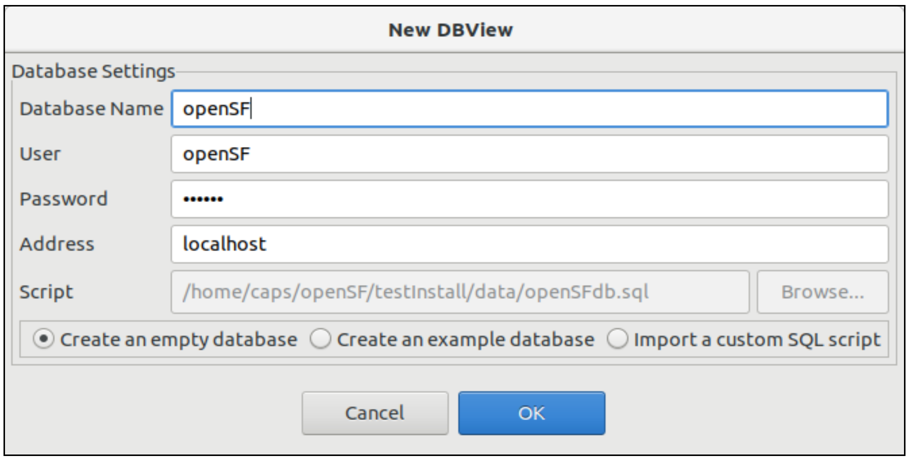
Connected to: debug

Figure 4-16: Connect to a database

The window shows that the system is connected to the 'debug database. If the user selects the 'openSF' database from the list, and clicks on the 'Connect' button, the system automatically connects to this database, as it is shown in the corresponding label. If an error occurs during the procedure and the connection cannot be performed, openSF shall show a message reporting the error.

4.5.2.2. Create a new database

If the user wants to create a new database in the MySQL server, only has to click on the new button, and a dialog will be shown by the application, as it is can see in the Figure 4-17.


New DBView

Database Settings

Database Name	openSF
User	openSF
Password	••••••
Address	localhost
Script	/home/caps/openSF/testInstall/data/openSFdb.sql

[✓ Create an empty database] Create an example database Import a custom SQL script

Figure 4-17: Create new database

A database has five characteristics fields:

1. **Database name**: Is the name of the database;
2. **User**: Is the user who connects to the databases server (requires privileges to create a database);
3. **Password**: Is the password of the user;
4. **Address**: Is the hostname where the databases server is located;
5. **Script**: Is the file used to create the database, a sequence of MySQL commands (.sql file). Besides the possibility to use a custom SQL script generated by the user, the scripts suggested during the

initial openSF installation, see Figure 3-7, which allow the creation of an empty database or a validation database with examples, can be selected through the available radio buttons.

All the fields have a default value, except the database name. Users can use these default values, or set others.

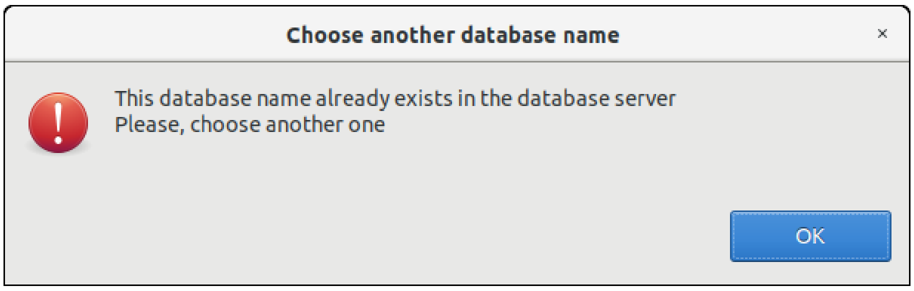
The default values are:

- User: ‘openSF’ (created during the installation of the application)
- Password: ‘openSF’ (password for the user ‘openSF’)
- Address: ‘localhost’
- Script: ‘openSFdb.sql’ (default script provided with the application)

It is mandatory that all fields have a correct value, except the password, that can be empty if the user wants.

When the user enters all the information correctly and clicks on the ‘OK’ button, the new database is created in the MySQL server, and automatically openSF is connected to it.

In case some field has been entered incorrectly (as for example an already existing database name, or the user or password to connect with the server are incorrect) openSF shall shows a message reporting the error as shown in Figure 4-18, and forces the user to enter the correct information.


Choose another database name

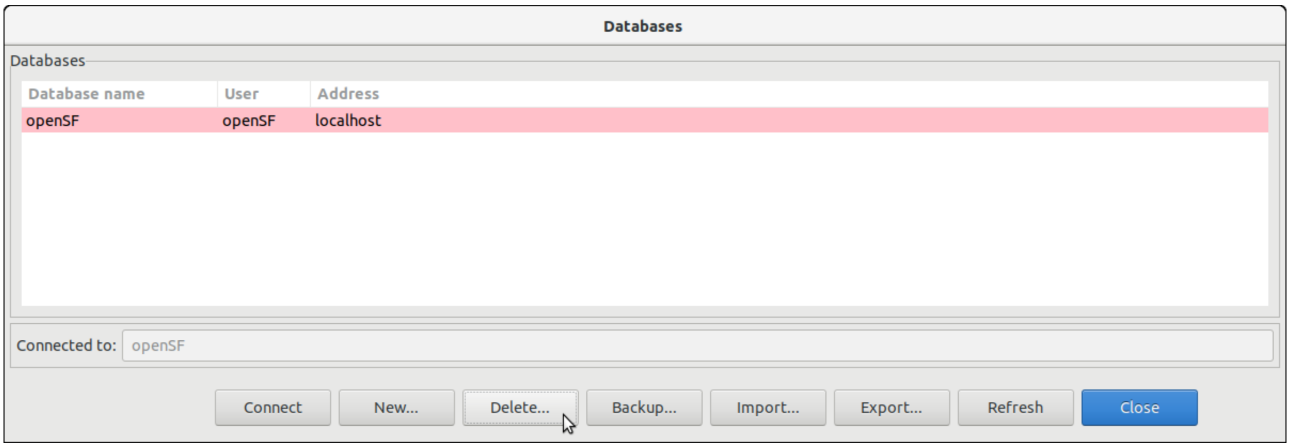
This database name already exists in the database server
Please, choose another one

Figure 4-18: Database name is wrong

If the user clicks on the ‘Cancel’ button on the databases window, no action is performed.

4.5.2.3. Delete a database

To remove a database from the databases server, the user has to select the database to remove from the list, and click on the ‘Delete’ button, as shown in the Figure 4-19.

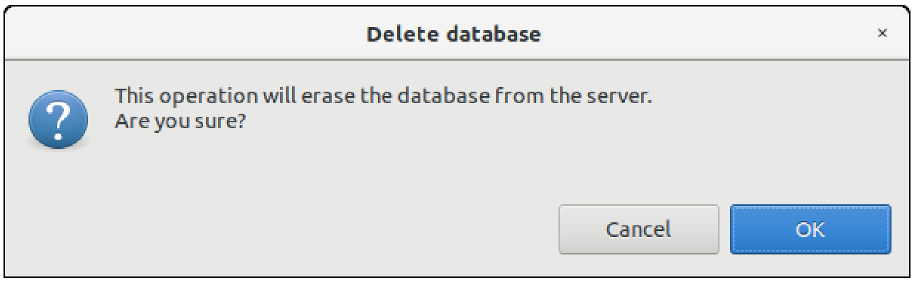


Database name	User	Address
openSF	openSF	localhost

Connected to: openSF

Figure 4-19: Delete a database

A new dialog is shown to confirm the action. If the user clicks on 'Yes, delete', the database is deleted.



This operation will erase the database from the server.
Are you sure?

Figure 4-20: Confirm deletion operation

openSF needs to be always connected to an existing database. Thus, the user cannot delete the connected database, and the system shall provide an error message. Therefore, the user must connect to another database to delete the current one.

4.5.2.4. Backup a database

The capability to perform a database backup is accessible from the Database Management window, presented in section 4.5.2. Once the database to backup has been selected the user only needs to click on the "Backup" button appearing at the bottom of the window.

The outcome of the operation is an SQL file containing all SQL statements that reproduce the status of the selected openSF database. This SQL script is the file used for the Database creation task, described in section 4.5.2.2.

4.5.2.5. Import a database

The capability to import elements definition specified using XML (refer to [AD-ICD] for the definition of the expected XML format) is accessible from the Database Management window, presented in section 4.5.2. The user only needs to click on the "Import" button appearing at the bottom of the window and select the XML file containing the database definitions.

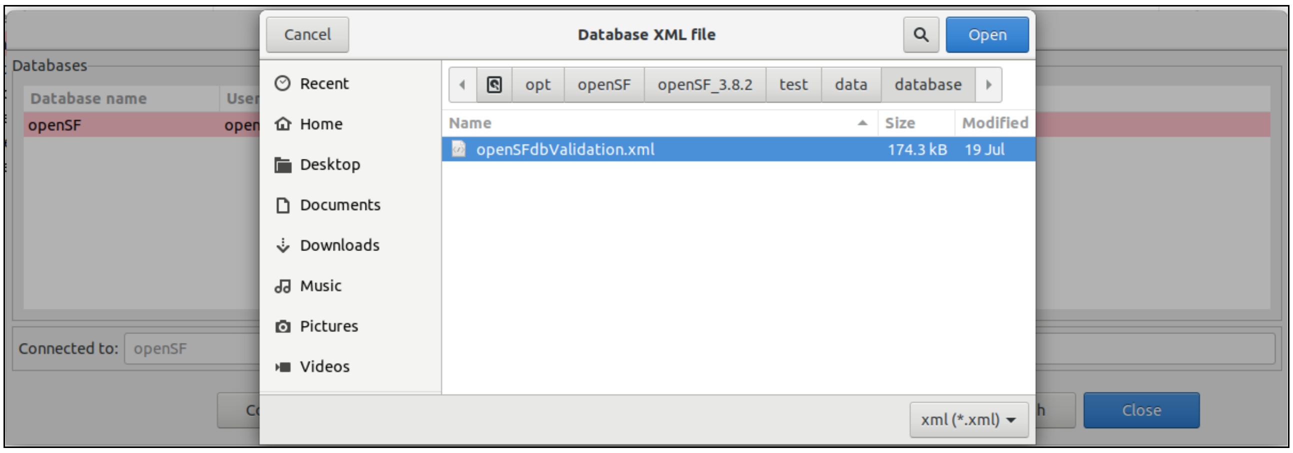
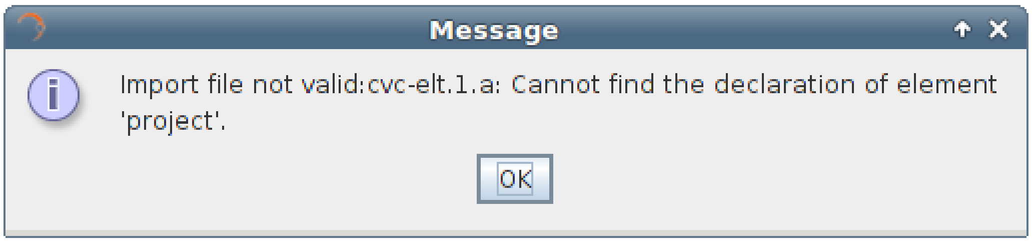


Figure 4-21: Database import

In the case the specified file is invalid a warning message is given and the import is cancelled.

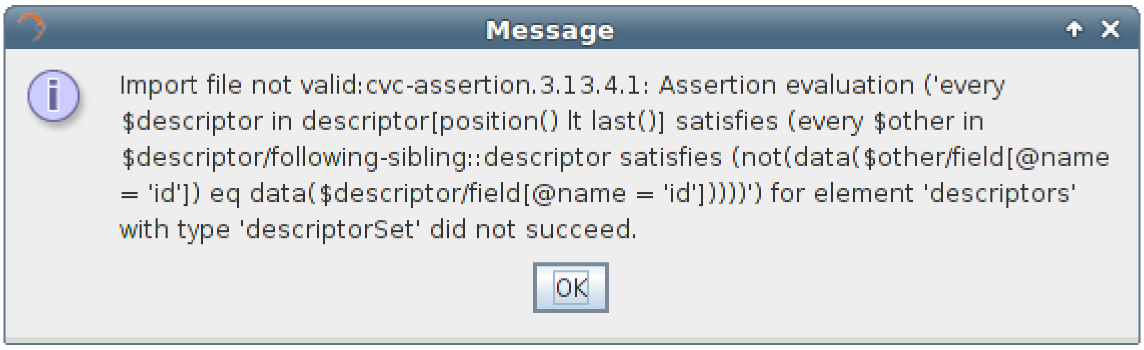

Message

Import file not valid:cvc-elt.1.a: Cannot find the declaration of element 'project'.

OK

Figure 4-22: Database import file error

If the file is valid but contains invalid definitions (e.g. non unique entities) a warning message is given and the import is cancelled.


Message

Import file not valid:cvc-assertion.3.13.4.1: Assertion evaluation ('every $descriptor in descriptor[position() lt last()] satisfies (every $other in $descriptor/following-sibling::descriptor satisfies (not(data($other/field[@name = 'id']) eq data($descriptor/field[@name = 'id']))))') for element 'descriptors' with type 'descriptorSet' did not succeed.

OK

Figure 4-23: Database import XML definition error

The outcome of the operation is the elements included in the XML file have been created and are part of the openSF repository.

4.5.2.6. Refresh database list

The database management dialog provides the capability to refresh the list of openSF databases available in the MySQL server to which openSF is connected to. This functionality identifies among the available databases the ones that are compatible with openSF. This capability is applicable in situations as manual migration of openSF database or automatic database upgrade, both occurring typically when upgrading to a more recent version of openSF.

4.5.3. Multinode simulation

openSF has the ability of orchestrating the remote execution of one (or several) module in a simulation run.

For the sample test simulation scenario shown in Figure 4-24 the user can choose which machine to use for each module to execute. Note that the remote execution in openSF relies on mounting a remote file system shared by all instances of openSF executing modules of a same simulation.

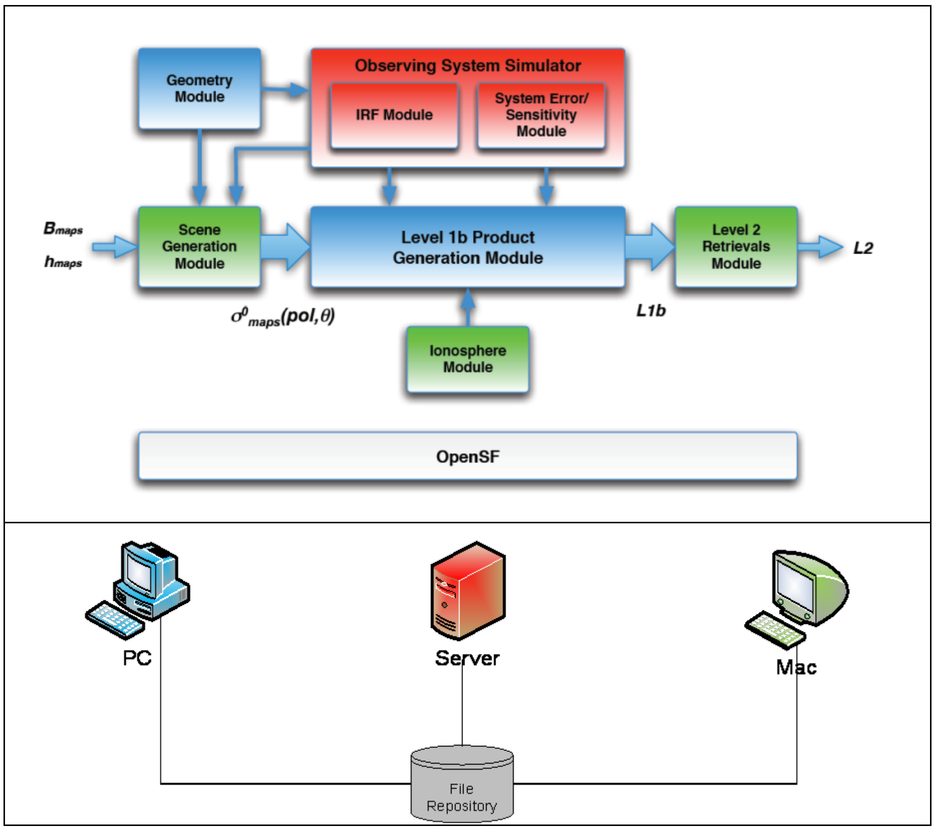
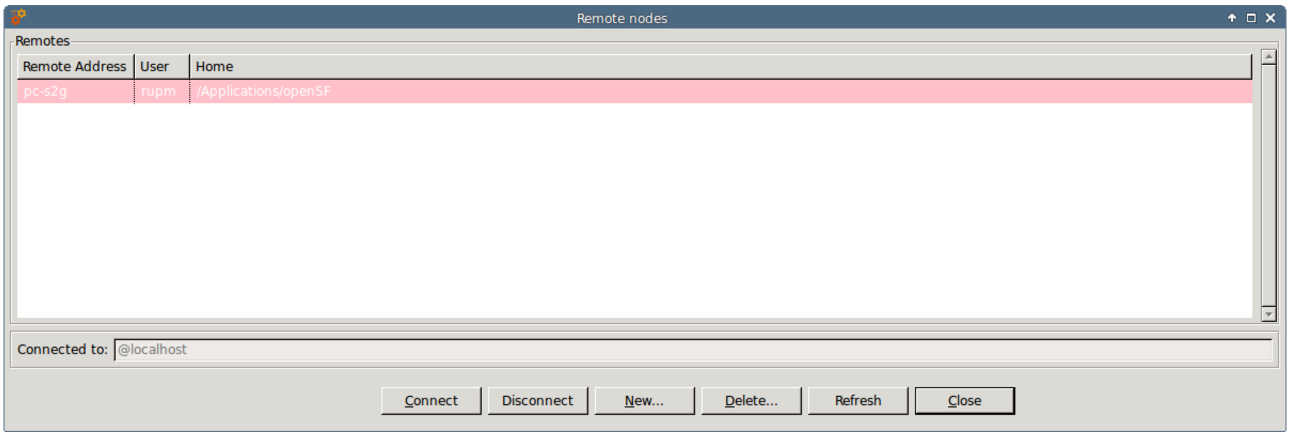


Figure 4-24: Outline of a simulation scenario

4.5.3.1. Remote machine management

For this purpose, a set of remote machines can be configured and managed in the system configuration.

When the user selects the “Remote” option from the menu ‘System’, the window shown in the Figure 4-25 will show up.



Remote Address	User	Home
pc-s2g	rupm	/Applications/openSF

Connected to: @localhost

Figure 4-25: Remote Machines Management window

In the bottom of the window there are five buttons, which allow users to perform different actions over the remote machine configuration. Users can connect or disconnect from a remote machine, create a remote machine reference, delete a remote machine configuration or refresh the list of available remote machines.

The central area of the window shows a list with the remote machine configured in openSF. Users can obtain information about the remote machine address, the user that connects to the machine and the remote path where openSF instance is installed.

The bottom of the window shows a label with the remote machine currently connected. In this case, the application is not connected to any remote machine so the local installation of openSF is used for storing the execution products (identified by <localhost> label).

Finally, there is an ‘Close’ button to return to the main window of the application.

4.5.3.2. Connect to a remote machine

Users can configure openSF to produce the simulation execution products in the file system of a remote machine. For this, the user has to select a remote machine from the list, and click on the “Connect” button. Automatically the file system is connected to it, and then, the name of the selected machine is shown beside the label “Connected to”.

Connecting to a remote machine file system means that the Simulation system folder used by openSF is located in a remote machine instead of in the local one.

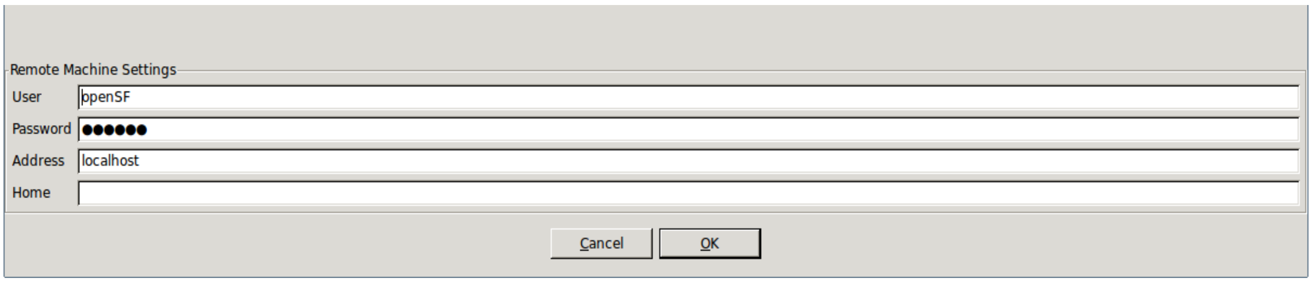
4.5.3.3. Disconnect from a remote machine

Users can disconnect from a remote machine a rely on the local machine file system for simulation execution. For this, the user has to click on the “Disconnect” button. In case openSF is already connected to a remote machine automatically the file system is disconnected from it, and then, the label <localhost> is shown beside the label “Connected to”.

Disconnecting from a remote machine file system means that the Simulation system folder used by openSF is the one in the local machine.

4.5.3.4. Declare a new remote machine

If the user wants to create a new remote machine reference, only has to click on the new button, and a dialog will be shown by the application, as it is can see in the Figure 4-26.



Remote Machine Settings

User	openSF
Password	●●●●●●
Address	localhost
Home	

Figure 4-26: Create new remote machine

A remote machine has four characteristics fields:

1. **Machine Address**: Is the address (IP or verbose) of the remote machine
2. **User**: Is the user who connects to the remote machine
3. **Password** *(optional)*: Is the password of the user. In case the ssh key has been setup this field is not required.
4. **Home**: Is the location in the remote machine where openSF instance is installed

Fields User and Home have a default value. Users can use these default values, or set others.

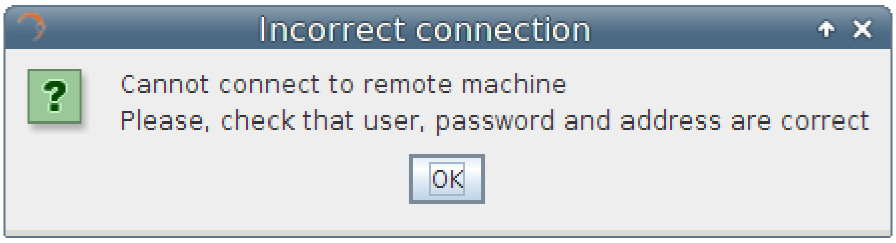
The default values are:

- ❑ User: ‘openSF’ (created during the installation of the application)
- ❑ Home: ‘/home/openSF/openSF’ (created during the installation of the application)

It is mandatory that all fields have a correct value, except the password, that can be empty if the user wants.

When the user enters all the information correctly and clicks on the ‘OK’ button, openSF attempts to connect to the remote machine using the configuration supplied.

In case some field has been entered incorrectly (as for example the user or password to connect with the server are incorrect) openSF shall shows a message reporting the error as shown in Figure 4-27, and forces the user to enter the correct information or cancel the creation.



Incorrect connection

Cannot connect to remote machine
Please, check that user, password and address are correct

Figure 4-27: Remote Machine is unreachable

If the user clicks on the ‘Cancel’ button on the databases window, no action is performed.

© DEIMOS Space S.L.U., 2019

4.5.3.5. Delete a remote machine

To remove a remote machine configuration, the user has to select the remote machine to remove from the list, and click on the 'Delete' button, as shown in the Figure 4-28. A new dialog is shown to confirm the action. If the user clicks on 'Yes, delete', the entry is deleted.

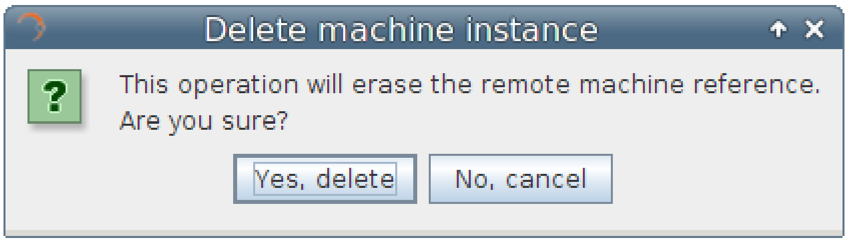


Figure 4-28: Confirm deletion operation

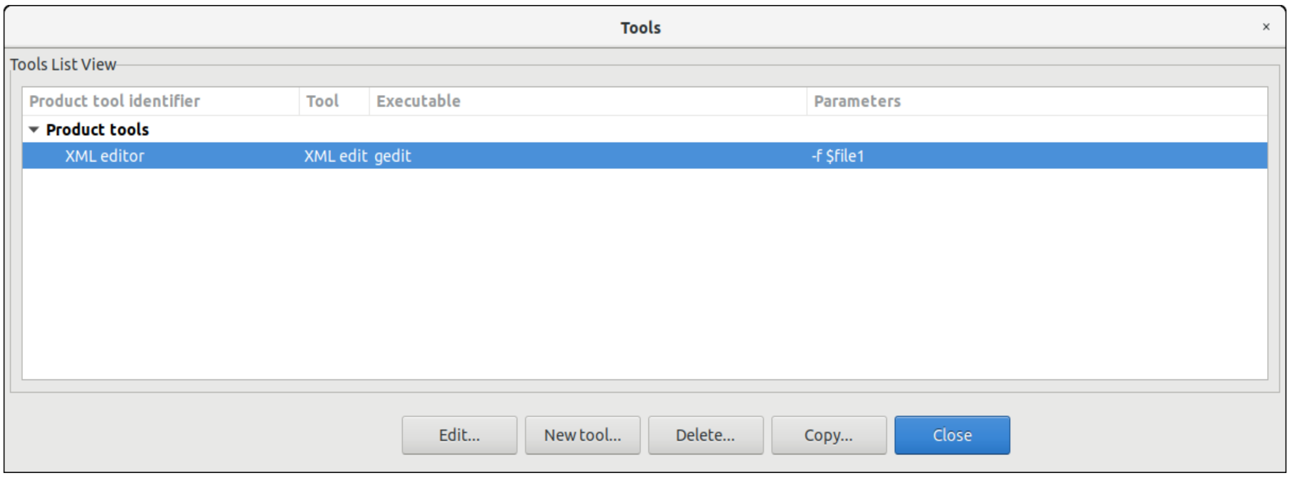
4.5.3.6. Refresh remote machine list

The remote machine management dialog provides the capability to refresh the list of remote machines to which openSF can be connected to. This capability is applicable in situations when recovering existing configuration upon upgrading to a new version of openSF software.

4.5.4. Product Tools

As explained in section 1.3, a tool is an external executable file that performs a given action to a certain group of files. Used as part of the openSF framework and associated to a certain file extension, these tools can be called to perform off-line operations to products involved in simulations.

Tools are classified as "internal", if they are part of the openSF distribution and are located in the tools' directory, or "external" in other cases. Currently openSF distribution package does not include any tool so all the product tools considered within this document are external and consequently there is no tools directory in openSF installation directory. Accessing to this functionality, the user is able to manage the openSF product tools.



Product tool identifier	Tool	Executable	Parameters
▾ Product tools			
XML editor	XML edit	gedit	-f $file1

Figure 4-29: Tools list view

A list of tools, showing its identifier, action, executable and parameters is given. Tools are definable by the user. Thus, new tools can be added by clicking the “New tool” button.

4.5.4.1. New tool

Accessing to this functionality, a new window appears to let the user create a new tool.

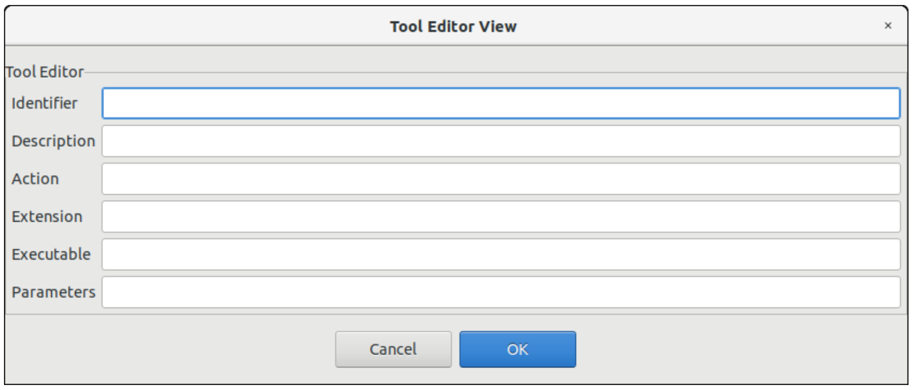


Figure 4-30: Tool. Creation

Users can define the following attributes:

Attribute name	Format	Purpose	Sample
Identifier	Medium string	This is a unique identifier of the tool.	“XML editor”
Description	Long string	A brief description of what this tool will do and need.	“This tool will open an XML file for editing”
Action	Medium string	What the tool is going to do.	“edit”
Extension	Short string.	The type of files that this tool is going to be applied to.	“xml”
Executable	Long string	Location of the executable file that is going to be called to execute the product.	“gedit”
Parameters	Long string	The list of parameters that will follow the executable. No variables can be passed from the HMI.	“-f $file1”

4.5.4.2. Edit tool

Selecting this functionality, the user can access and edit all the attributes of the selected tool.

4.5.4.3. Delete tool

Selecting this functionality, the user can delete the selected tool.

4.5.4.4. Product Tool Execution

The process to execute an external data exploitation application is described in other sections:

- Tool execution from the file system view, section 4.17.1.
- Tool execution from the simulation edition view and execution results view (see Figure 4-21). The execution view is the same as the simulation view with two particularities, Log Tab is enabled and the status is completed (successfully or not). Note that in the execution view the status of the output files is shown as *available* if the simulation chain has been successfully executed while in the simulation editing the status is shown as *pending* (see section 4.9.2.4).
- Scheduled execution over simulation data products, section 4.9.2.9.

For a real example of product tool execution there is more information in chapter 5.4.

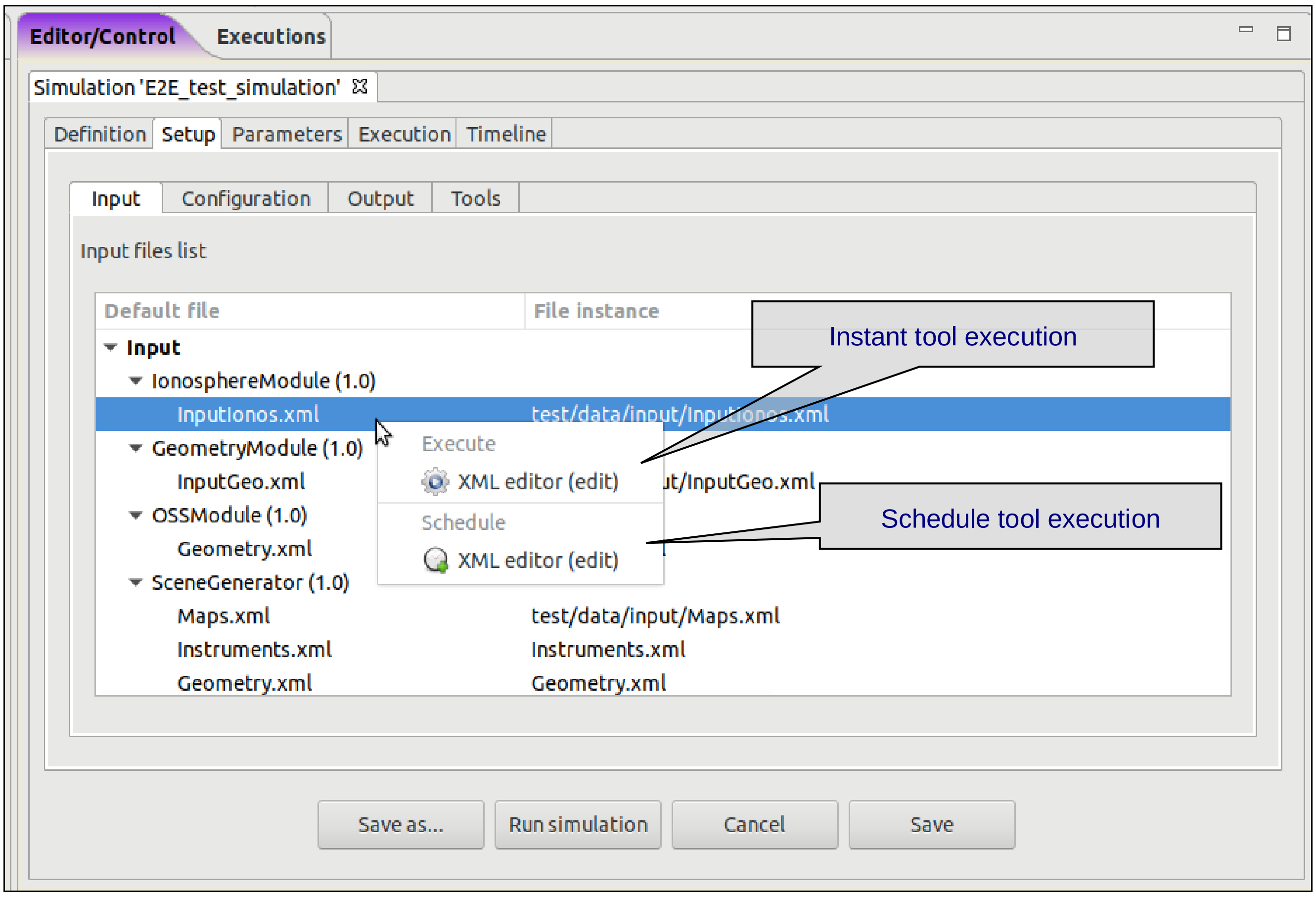


Figure 4-31: Tool Execution/Schedule from Simulation Edition View

4.5.4.5. Popular Product tools

During the integration of openSF in E2E simulation projects the development team has identified a set of **product tools** widely used and that are freely available on the web. For every listed tool the operating system compatibility is also specified (Linux, Multi-platform…).

4.5.4.5.1. Image Processing Tools

Below these lines are listed a set of tools for viewing and editing image files. The applications listed support a large number of image formats.

Image Viewers

- IrfanView - Multiformat Image Viewer - Microsoft Windows
- Eye of Gnome - Gnome Image Viewer - Linux (GNOME)
- Gwenview - KDE Image Viewer - Linux (KDE)
- Okular - KDE Document and Image Viewer - Linux (KDE)

Image Editors

- GIMP - The free Adobe Photoshop alternative - Multi-platform
- Inkscape - Image editor with vector graphics support - Multi-platform

4.5.4.5.2. Text Editors

- Notepad
- Emacs - GNU Editor - Multi-platform
- Notepad++ - Full-featured - Microsoft Windows

4.5.4.5.3. Scientific Data Formats

NetCDF

Network Common Data Form is a set of interfaces for array-oriented data access and a freely-distributed collection of data access libraries for C, Fortran, C++, Java, and other languages.

- Panoply - NetCDF data plotting - Multi-platform
- ncBrowse - NetCDF file browser - Multi-platform

HDF

Hierarchical Data Format, commonly abbreviated HDF, HDF4, or HDF5 is the name of a set of file formats and libraries designed to store and organize large amounts of numerical data.

- HDFView - HDF File viewer (images, tables...) - Multi-platform

4.5.4.5.4. Browsers

A web browser is a software application for retrieving, presenting, and traversing information resources on the World Wide Web. This kind of applications has a lot of possible functionalities as openSF product tools.

© DEIMOS Space S.L.U., 2019

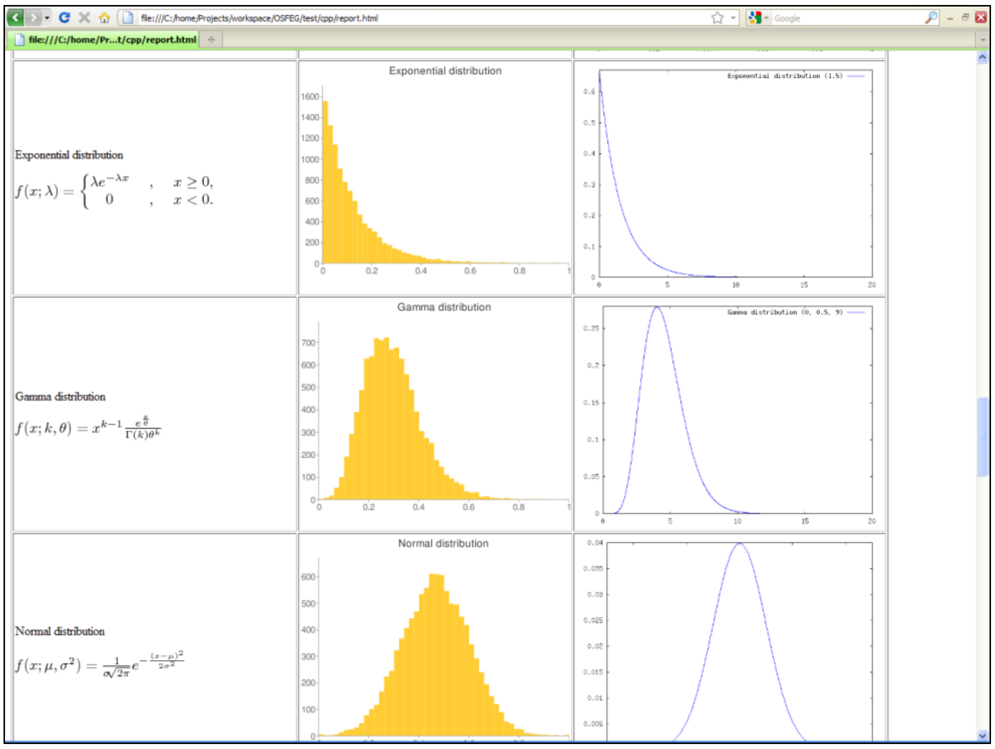

file:///C:/home/Projects/workspace/OSFEG/test/cpp/report.html
file:///C:/home/Pr...t/cpp/report.html
Exponential distribution
$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} & , \quad x \geq 0, \\ 0 & , \quad x < 0. \end{cases}$
Exponential distribution (1.5)
Gamma distribution
$f(x;k,\theta) = x^{k-1} \frac{e^{-\frac{x}{\theta}}}{\Gamma(k)\theta^k}$
Gamma distribution (0, 0.5, 9)
Normal distribution
$f(x;\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

Figure 4-32: Mozilla Firefox as openSF Product Tool

Example: Figure 4-32 shows the use of an Internet Browser (Mozilla Firefox) for graphing the results of testing a random number generator. It uses Octave graphing capabilities and Google Charts API. For further information visit http://code.google.com/intl/en/apis/charttools/

- Mozilla Firefox – Multi-platform
- Google Chrome – Multi-platform
- Opera – Multi-platform

4.5.4.5.5. Other Tools

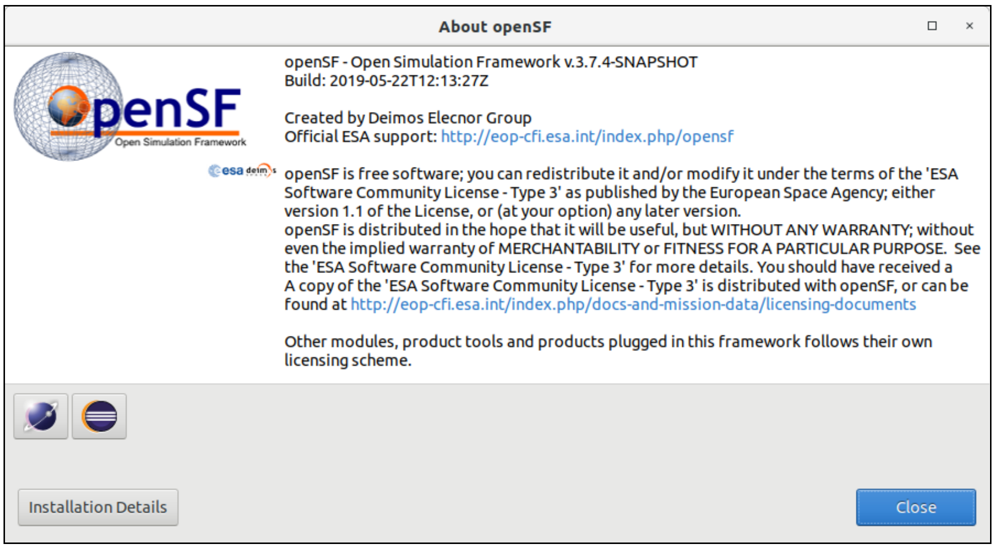
- GNU Octave - If GIMP is the free photoshop-like choice, this is the equivalent for MATLAB - Multi-platform
- GNU Plot - GNU software that gives plotting capabilities through a command line interface - Multi-platform

4.5.5. Check for updates

openSF performs automatic check for new versions by connecting to a remote server. In case a new version is identified the user is given the choice of downloading the software. Afterwards the user can upgrade the software version following the standard installation procedure for openSF software.

4.5.6. About openSF

Accessing this functionality from the menu, the system will show a dialog with the copyright and license scheme for the openSF platform.


About openSF

openSF - Open Simulation Framework v.3.7.4-SNAPSHOT
Build: 2019-05-22T12:13:27Z

Created by Deimos Elecnor Group
Official ESA support: http://eop-cfi.esa.int/index.php/opensf

openSF is free software; you can redistribute it and/or modify it under the terms of the 'ESA Software Community License - Type 3' as published by the European Space Agency; either version 1.1 of the License, or (at your option) any later version.
openSF is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the 'ESA Software Community License - Type 3' for more details. You should have received a A copy of the 'ESA Software Community License - Type 3' is distributed with openSF, or can be found at http://eop-cfi.esa.int/index.php/docs-and-mission-data/licensing-documents

Other modules, product tools and products plugged in this framework follows their own licensing scheme.

Figure 4-33: openSF About View

4.5.7. Help View

The Help View is accessible from the openSF main menu bar, system tab.

Additionally, users are able to view these documents launching the proper software application. The applications for viewing the supported file types are the ones defined as default applications for current Desktop. Depending on the desktop environment these default applications can be different even running under the same operating system.

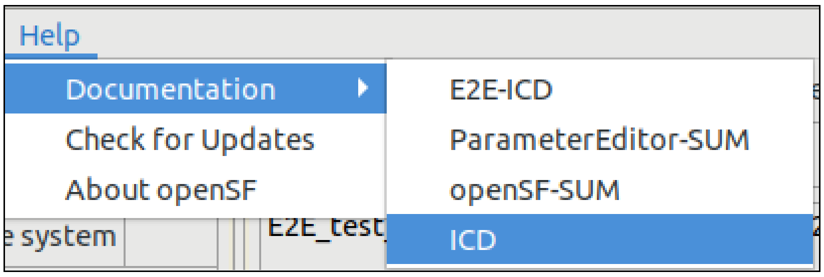


Figure 4-34: Help documents tree view

The supported file types and the typical viewers are the following:

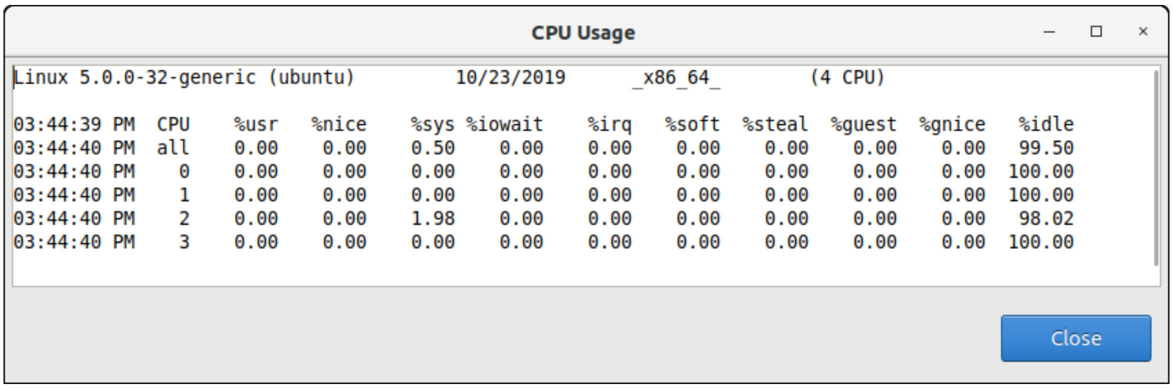
- ❑ PDF: “Portable Data Format”; Acrobat Reader
- ❑ HTML: “Hypertext Markup Language”; Web Browser
- ❑ Plain Text: Notepad, Vim, Emacs ...

4.5.8. CPU Usage

This dialog is helpful in analysing the CPU core use when parallelising module execution in openSF platform.

4.5.8.1. Linux

Accessing this functionality, the system will show a custom dialog with occupation of CPU cores by machine processes.



```
Linux 5.0.0-32-generic (ubuntu)         10/23/2019      _x86_64_        (4 CPU)

03:44:39 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
03:44:40 PM  all    0.00    0.00    0.50    0.00    0.00    0.00    0.00    0.00    0.00   99.50
03:44:40 PM    0    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
03:44:40 PM    1    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
03:44:40 PM    2    0.00    0.00    1.98    0.00    0.00    0.00    0.00    0.00    0.00   98.02
03:44:40 PM    3    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
```

Figure 4-35: CPU Core Usage view

4.5.8.2. macOS

The macOS Operating System already provides a default application to convey information about CPU usage, the ‘Activity Monitor’. Accessing the CPU Usage functionality in openSF shall therefore launch the ‘Activity Monitor’ external application.

4.5.8.3. Windows

In Windows this CPU usage tool is not available and hence the user shall use the Windows task Manager instead.

4.5.9. Exit the system

Upon the selection of this function, openSF will inform the user whenever a simulation is executing. Upon user confirmation, openSF will stop every internal process (including ongoing external modules) and will end its execution. This is the recommended way of ending the application.

4.6. Repository

The openSF system can be divided in two logical parts: repository and executions.

© DEIMOS Space S.L.U., 2019

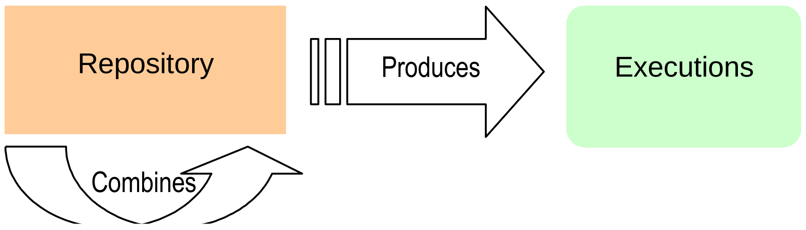


Figure 4-36: openSF logical flow

The first of these parts, repository, is involved in defining the “static” view of the system, the definition of the openSF tool. This is the place to access and define all the elements that, later on, will serve as bricks for the simulation executions:

- **Modules** – executable entities that will perform the scientific or engineer calculations;
- **Descriptor** – entities which define the modules’ set of input and output files and which are used by openSF to connect the modules and define its execution sequence in the simulation runs;
- **Simulations** – complex sequences of modules, linked by their descriptors, with added input/output/configuration files and product tools.

Into the repository tab of the side bar users can find a tree-like structure containing the definitions for all known modules, descriptors and simulations.

This tree-like structure can be collapsed or expanded by clicking in the “arrow” icon.

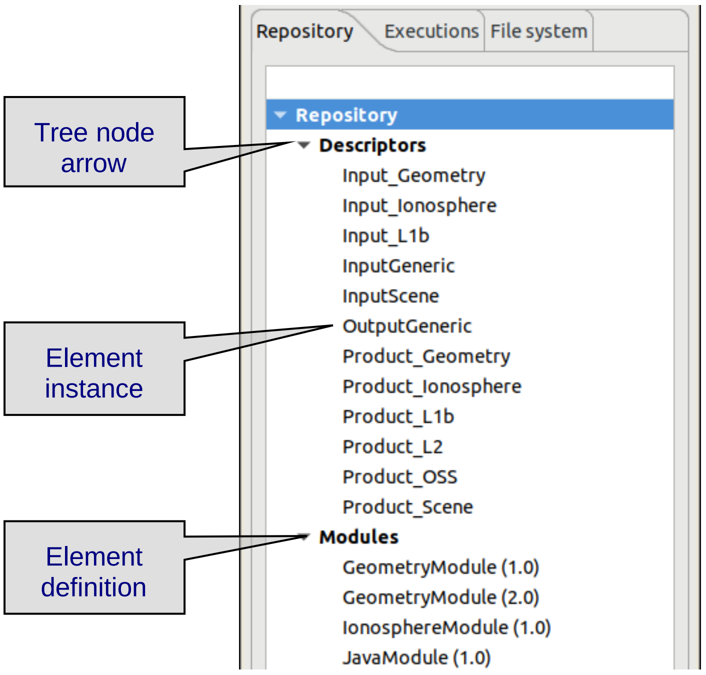


Figure 4-37: Repository view

Every row marked with an arrow icon represents and element definition of the repository. Every row without icon represents an element instance. Right-clicking over both of them, a menu pops up containing some associated commands. These menus are context-sensitive, meaning that different types of elements have their own associated commands. These commands are going to be explained in detail in each element’s section.

A left double click over the elements will activate the first associated command of the menu (typically, editing).

4.7. Descriptors

openSF has the possibility to define the set of input and output files (called descriptors) used to connect different modules in simulation runs.

Users can access the list of nominal descriptors (those provided in the default distribution) inside the repository view of the side bar, as seen in Figure 4-38.

Accessing the corresponding menu of the main menu bar or the context-menu of the side bar, users can activate the following functionalities:

- ❑ *List* – presents the list of existing descriptors;
- ❑ *Creation* – creates a new descriptor into the system;
- ❑ *View* – displays the contents of an existing descriptor
- ❑ *Modification* – edits an existing descriptor to enter changes;
- ❑ *Deletion* – deletes a descriptor from the system;
- ❑ *Copy* – creates a copy of an existing descriptor.

The options available at each time will depend on the active user role as explained in the section 4.13

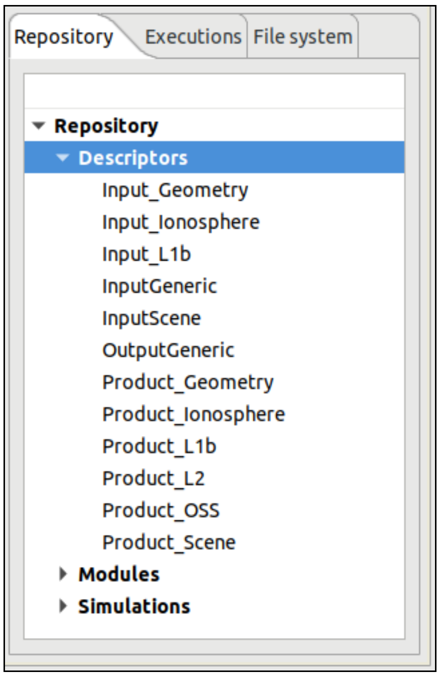
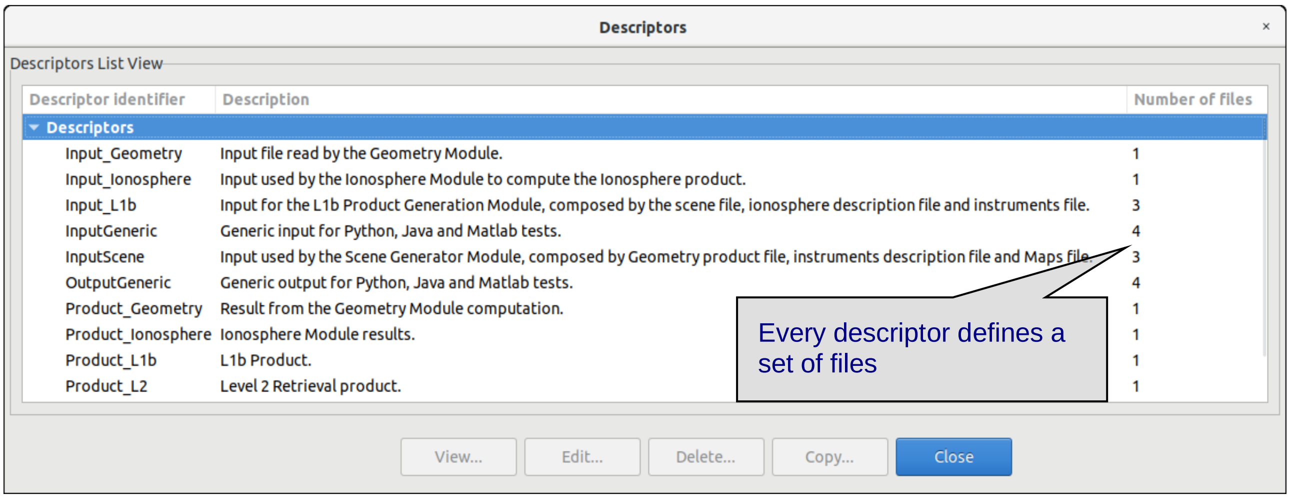


Figure 4-38: Descriptor. Side bar

4.7.1. Descriptor list

Users can access to a window that provides a tree-like structure with the list of descriptors known by the system, just as in the side bar but with the additional information of its identifier and its description.



Descriptor identifier	Description	Number of files
Descriptors		
Input_Geometry	Input file read by the Geometry Module.	1
Input_Ionosphere	Input used by the Ionosphere Module to compute the Ionosphere product.	1
Input_L1b	Input for the L1b Product Generation Module, composed by the scene file, ionosphere description file and instruments file.	3
InputGeneric	Generic input for Python, Java and Matlab tests.	4
InputScene	Input used by the Scene Generator Module, composed by Geometry product file, instruments description file and Maps file.	3
OutputGeneric	Generic output for Python, Java and Matlab tests.	4
Product_Geometry	Result from the Geometry Module computation.	1
Product_Ionosphere	Ionosphere Module results.	1
Product_L1b	L1b Product.	1
Product_L2	Level 2 Retrieval product.	1

Figure 4-39: Descriptor list view

4.7.2. Descriptor creation

Users can define new descriptors in case they want to accommodate third-party modules that cannot make use of any of the nominal descriptors.

The frame shown in Figure 4-40 is responsible to define the descriptor’s characteristics:

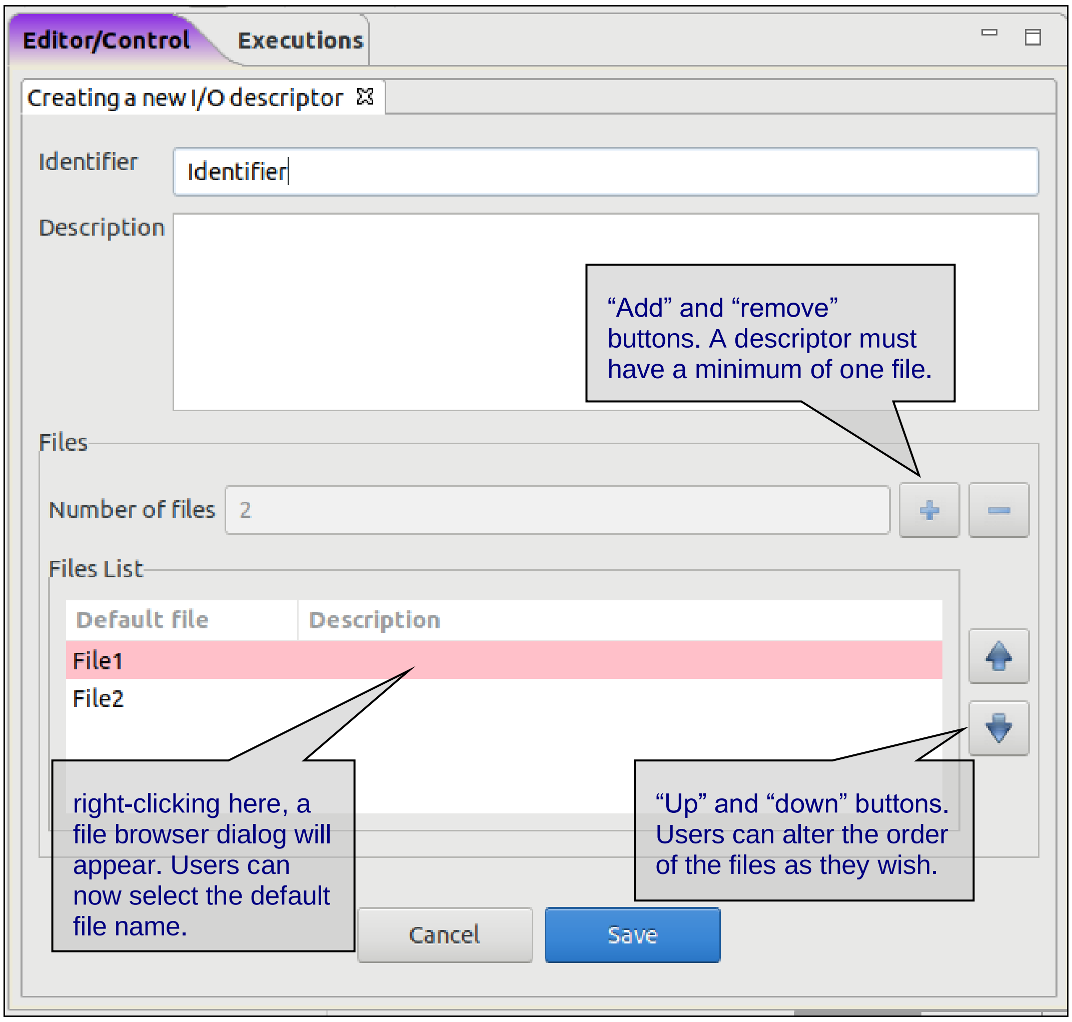

Editor/Control
Executions
Creating a new I/O descriptor
Identifier
Identifier
Description
"Add" and "remove" buttons. A descriptor must have a minimum of one file.
Files
Number of files
2
Files List
Default file
Description
File1
File2
right-clicking here, a file browser dialog will appear. Users can now select the default file name.
"Up" and "down" buttons. Users can alter the order of the files as they wish.
Cancel
Save

Figure 4-40: Descriptors. New descriptor

The attributes that identify the descriptor and that shall be set by the user are the following:

Attribute name	**Format**	**Purpose**	**Sample**
Identifier	Medium string	Descriptor's unique name.	"LIDAR In"
Description	Long string	A brief description of its composition or the purpose of the set of files.	"Orbit information and radiative transfer information"

It is possible to alter the set of files that integrates the descriptor. Users can edit, add or remove files. An individual file must be described by these two parameters:

Attribute name	Format	Purpose	Sample
Default file	Medium string	The default location and name of the file. This is the file that is going to be suggested during the simulation definition (see section 4.9).	"orbit.xml"
Description	Long string	Brief description of the file's composition, its purpose or its type (XML, TIFF, NETCDF, etc.).	"XML file with Orbital information"

It is important to note that **the default file name is the way to know if two modules are compatible and to connect them in the simulation definition**.

The order that those files occupy in the list is important. This order must fulfil the directives of the command line specification of the third-party module because how the files are sorted will define the order of the input and output files in the command line of the module execution (see [AD ICD]). This order can be altered with the "up" and "down" buttons that move the selected file through the list.

Upon creating a descriptor its default file field is considered as a template name, i.e., this definition may be used as-is during simulation execution or it can be changed in the simulation edition window. When in changing the actual file descriptor in simulation edition the file location can be set to any path, either an absolute one (in the local machine) or a relative one. By default, the descriptor is considered as a relative path. It may be relative to (a) the openSF installation folder (as per $OPENSF_HOME), or (b) the simulation execution folder:

(a) It will be relative to the openSF folder when the descriptor is used for input only (e.g. the input descriptor of the first module of a processing chain).

(b) It will be relative to the simulation folder when the descriptor is used both as input descriptor and output descriptor (e.g. a folder generated during processing and later on used as input to a next module).

4.7.3. Descriptor modification

Users can select a certain descriptor and choose the option to edit it. Note that any modifications performed on the descriptor will also affect all the modules and simulations that use it.

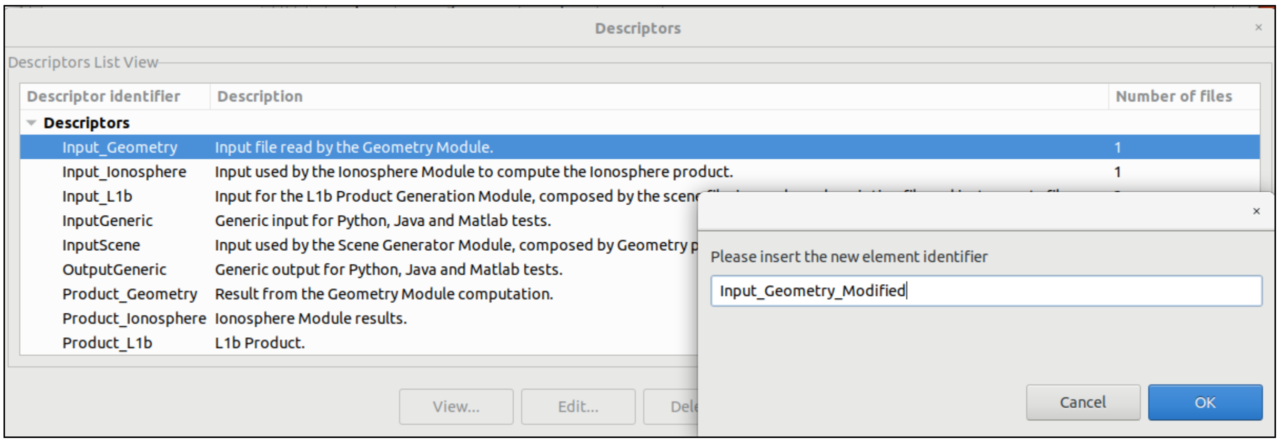
4.7.4. Descriptor deletion

Users can also select a descriptor to delete it. Once users confirm the operation the descriptor is erased from the repository. **Note that for consistency purposes, every module or simulation (and its results) that make use of this descriptor will also be deleted from the system**.

4.7.5. Descriptor copy

Users can select a certain descriptor and choose the option to copy it. The user needs to specify a new name for the descriptor, unique with respect to the existing descriptors.

© DEIMOS Space S.L.U., 2019



Descriptors

Descriptors List View

Descriptor identifier	Description	Number of files
Descriptors		
Input_Geometry	Input file read by the Geometry Module.	1
Input_Ionosphere	Input used by the Ionosphere Module to compute the Ionosphere product.	1
Input_L1b	Input for the L1b Product Generation Module, composed by the scen	
InputGeneric	Generic input for Python, Java and Matlab tests.	
InputScene	Input used by the Scene Generator Module, composed by Geometry p	
OutputGeneric	Generic output for Python, Java and Matlab tests.	
Product_Geometry	Result from the Geometry Module computation.	
Product_Ionosphere	Ionosphere Module results.	
Product_L1b	L1b Product.	

Please insert the new element identifier

Input_Geometry_Modified

Figure 4-41: Descriptors. Copy Descriptor

4.8. Modules

According to the definition given in section 1.3, a module is an executable entity that can take part in a simulation. Users are able to manage all modules that can take part in openSF simulations. The operations upon modules, which vary depending on the active user role (see section 4.13), are:

- List – present the list of existing modules;
- Creation – capability to create a new module into the system;
- View – displays the contents of an existing descriptor
- New version – create a new version of an existing module;
- Modification – edit an existing module to enter changes;
- Deletion – delete a module from the system;
- Copy – creates a copy of an existing module.

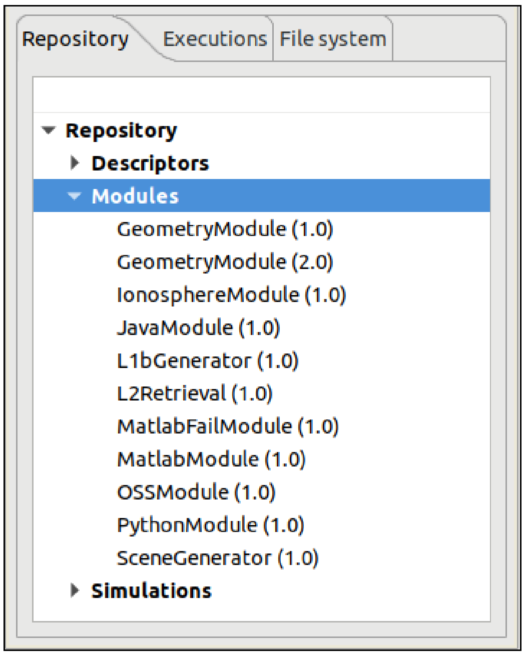


Figure 4-42: Repository view: modules

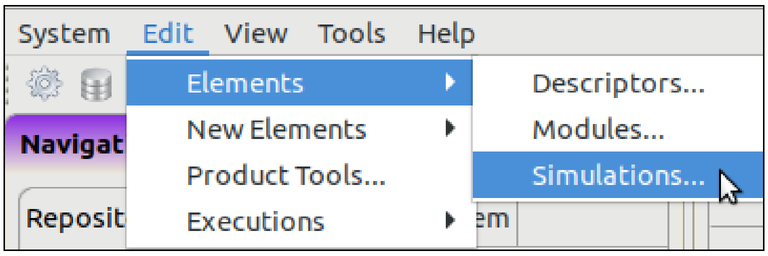


Figure 4-43: Repository menu

Users can access to some of these operations at the modules' menu in the menu bar of the main window or in the correspondent context-menu of the repository view.

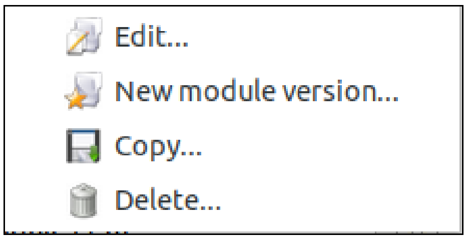


Figure 4-44: Module. Pop-up menu

4.8.1. Module developer's guideline

openSF can integrate as a module almost every executable piece of code that follows the requirements described in [AD-ICD] and compliant modules can be integrated and executed into the system.

Nevertheless, module developers must have in mind the following points:

- ❑ Memory handling is responsibility of the module. openSF does not manage memory assignments and does not destroy any data structure created by the module;
- ❑ A module can create child processes, but their management is still on the module developer's side;
- ❑ openSF does not detect when a module execution is "halted" or in an infinite loop. Please send some logging information to the openSF every two seconds (at most) to let the user know there is no problem;
- ❑ Execution performance of the module could be slightly slowed because of the messaging interception;
- ❑ Module developer is responsible of the error and exception handling as explained in [AD-ICD]. (Error Handling section).

4.8.2. Module list

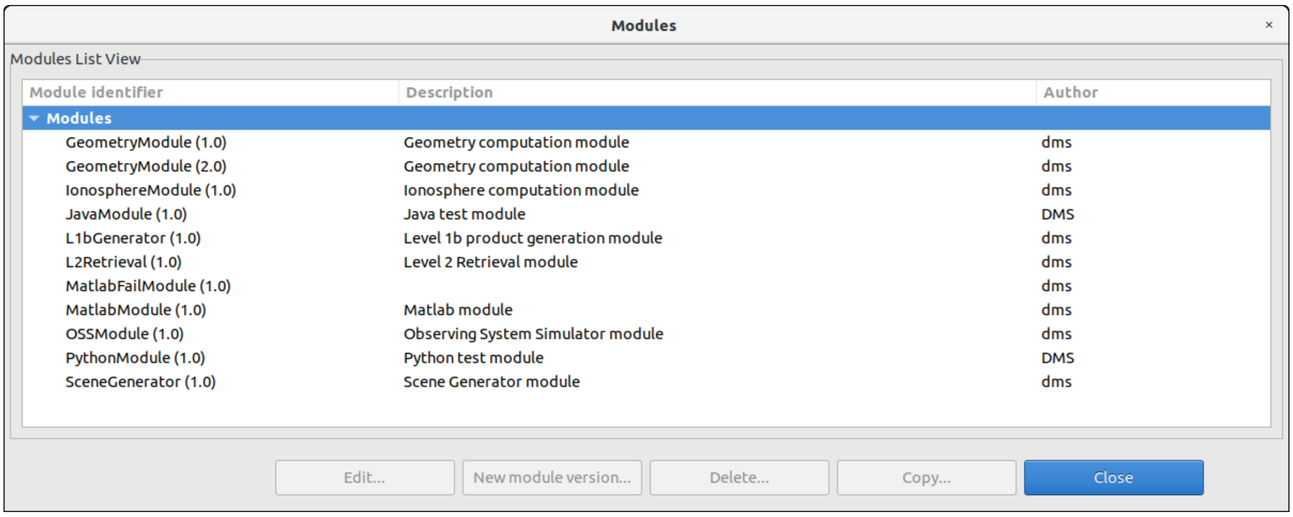
Accessing to this functionality from the main menu or from the repository, the system will show a list of modules known by the system. Figure 4-45 shows an example of the window that appears upon its selection.

As modules and their versions follow a hierarchical structure, i.e. a single module belongs to a family of modules, the information is organized as a tree-table, which put hierarchical data in columns, like a table, but uses an indented outline structure in the first column to illustrate the tree structure.

Users can select a certain version of a module and perform the operations shown in the toolbar.

Data attributes shown in this tree-table are module ID, version number, type, description and the name of the author.

At the "module identifier" column, it is presented the indented outline structure with folder icons as nodes and document icons as leaves. Double-click over each folder will expand or collapse its content.



Module identifier	Description	Author
▾ Modules		
GeometryModule (1.0)	Geometry computation module	dms
GeometryModule (2.0)	Geometry computation module	dms
IonosphereModule (1.0)	Ionosphere computation module	dms
JavaModule (1.0)	Java test module	DMS
L1bGenerator (1.0)	Level 1b product generation module	dms
L2Retrieval (1.0)	Level 2 Retrieval module	dms
MatlabFailModule (1.0)		dms
MatlabModule (1.0)	Matlab module	dms
OSSModule (1.0)	Observing System Simulator module	dms
PythonModule (1.0)	Python test module	DMS
SceneGenerator (1.0)	Scene Generator module	dms

Figure 4-45: Module list view

4.8.3. Adding modules

Users can add a new module accessing this functionality from the main menu or alternatively clicking over the “New” button in the module manager.

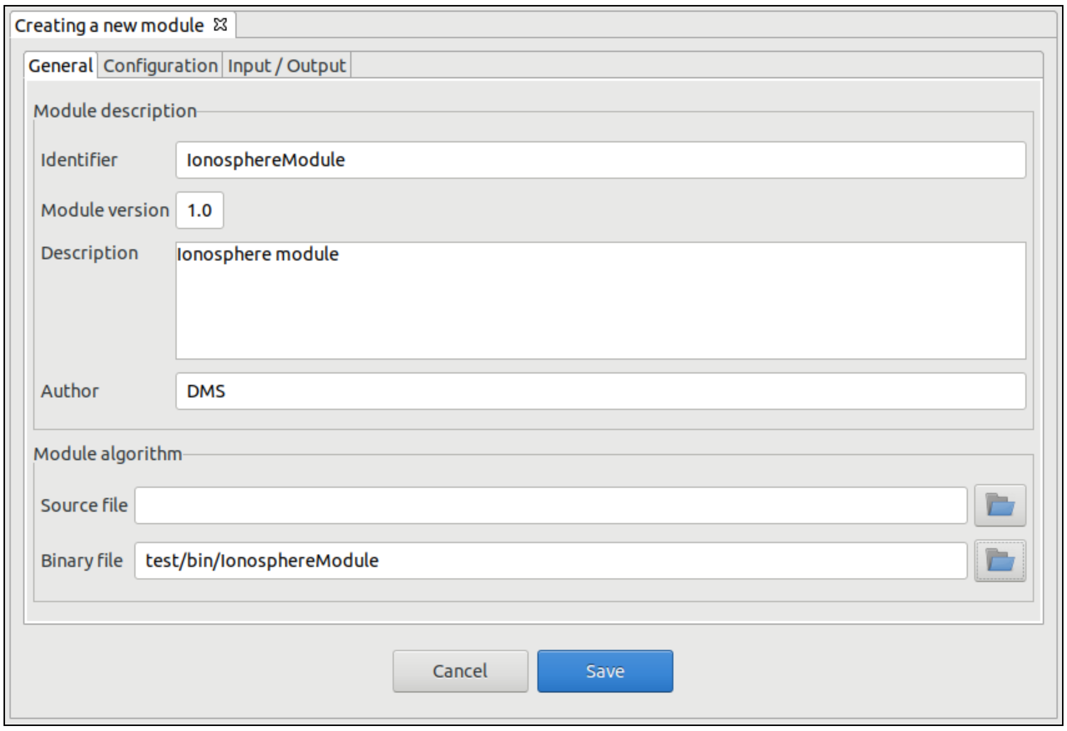
This frame contains the components needed to introduce all data to define a new module in the system. These data (module attributes) are grouped with the following structure:

- General
- Configuration
- Input / Output

Each category is analysed in the following sub-sections.

4.8.3.1. General data

In this group (Figure 4-46) users must define general information about the module to create.



Identifier: IonosphereModule
Module version: 1.0
Description: Ionosphere module
Author: DMS
Source file:
Binary file: test/bin/IonosphereModule

Figure 4-46: Module. General data

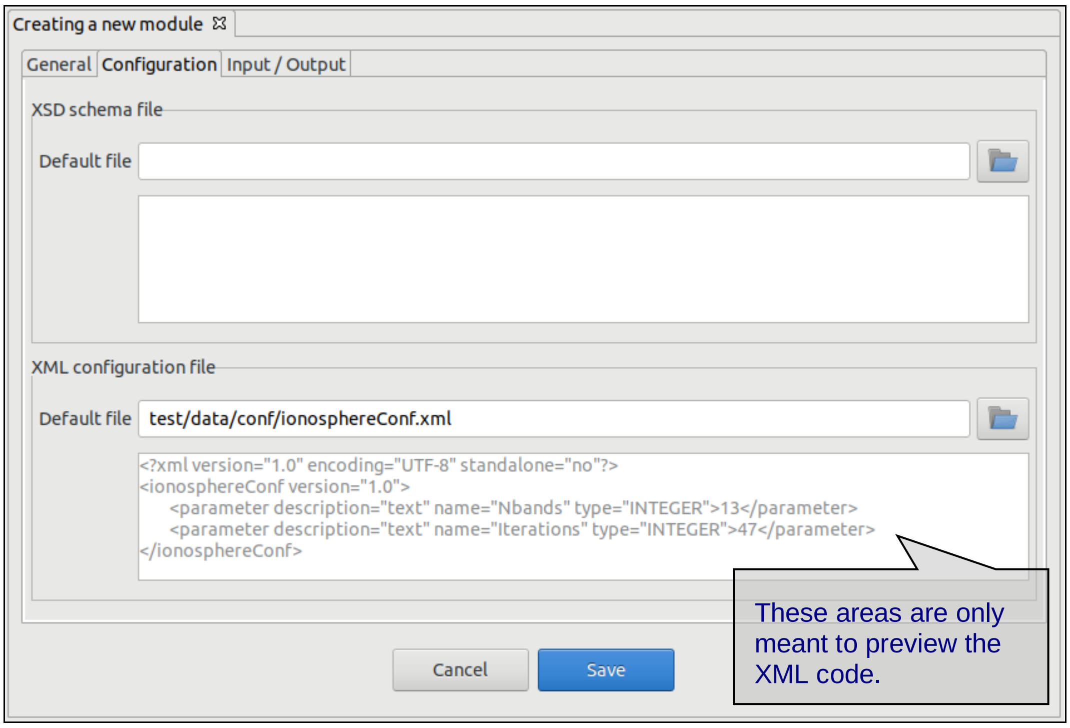
The fields to define are as follows:

Attribute Name	Format	Purpose	Sample
Module Identifier	Medium string	Unique module identification.	“LIDAR”
Module version	Float	In a new module this field will be filled with a default value. Users cannot edit this unless they use the “new version” functionality.	1.0

Attribute Name	Format	Purpose	Sample
Description	Long string	Free writing area where to briefly describe the module.	"State-of-the-art" LIDAR instrument module
Author	Medium string	Text field where to write the author's name.	"DMS"
Source file	File	There is a text area to write the file name[7] and a button that will show a dialog to locate and choose the intended source code file.	"modules/LIDAR/src/lidar.f90"
Executable file	File	The executable[8] file (either a binary executable or script) to be launched when performing a simulation.	"module/LIDAR/bin/lidar"

4.8.3.2. Configuration

Selecting the "Configuration" tab (Figure 4-47), users can select the XML configuration file and its correspondent XSD schema file using file-browser dialogues. Text areas are also provided to preview the XML code.



Default file: test/data/conf/ionosphereConf.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ionosphereConf version="1.0">
    <parameter description="text" name="Nbands" type="INTEGER">13</parameter>
    <parameter description="text" name="Iterations" type="INTEGER">47</parameter>
</ionosphereConf>
```

Figure 4-47: Module. Configuration

[7] Codes in general will have several routines. However, there will be always a "main". This function is interesting to pinpoint, as it is a sort of manager for the rest of routines.

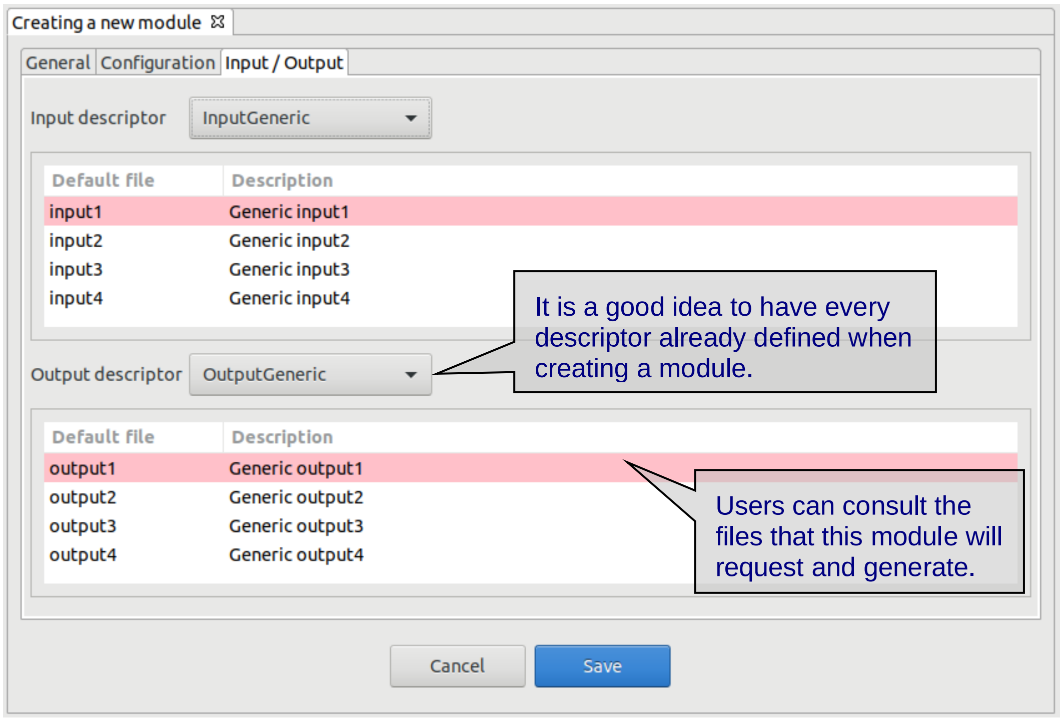
[8] The compilation of the modules is a process outside the scope of openSF.

The XSD schema file will validate the respective configuration file and report any found inconsistencies. By default, as indicated in section 4.5.1, openSF is set to skip said validation. However, the user can select within the Preferences window to allow the validation and generate either warnings or errors.

4.8.3.3. IO descriptors

The "Input/Output" tab (Figure 4-48) from the Module properties window enable users to specify, respectively, the contents of the input files expected for the module, and the output files the module produces as output. Thus, within this tab users can select the input and output descriptors for this module. The default file identifiers of the IO descriptors allow the module connection when defining simulations.

Each IO descriptor has an identifier that uniquely identifies the descriptor among the system. It may happen that an IO descriptor for a new module may exist already in the system, that is, this module uses the same type of files and file contents as another module. Therefore, a combo-box component is presented with the list of known IO descriptors in case the user desires to select an existing one.



Input descriptor: InputGeneric

Default file	Description
input1	Generic input1
input2	Generic input2
input3	Generic input3
input4	Generic input4

Output descriptor: OutputGeneric

Default file	Description
output1	Generic output1
output2	Generic output2
output3	Generic output3
output4	Generic output4

It is a good idea to have every descriptor already defined when creating a module.

Users can consult the files that this module will request and generate.

Figure 4-48: Module. Input/Output specification

4.8.4. Module upgrade - New version

A new version of a module represents an upgrade of the implementation of a given module. This means that users can define a new module by altering any of the elements defined in the model creation, like for example the executable file of the module, the configuration file or the input and output files.

Users can create a new module version selecting the correspondent action from the context-menu of the repository view or alternatively clicking over the "new version" button in the module list.

The system will automatically perform a "minor version upgrade" (for example, from 1.0 to 1.1), but this numbering can be manually modified by the user.

This way, users can have two versions of the same module with modifications between them.

4.8.5. Module modification

Editing a module from the repository view or from the modules list, openSF will present the same window as in the previous chapter with all known data already filled. The window shall present in write-mode only the data fields that are susceptible to be modified. If users want to change more attributes of a certain module, other operations must be used ("module creation" or "new module version").

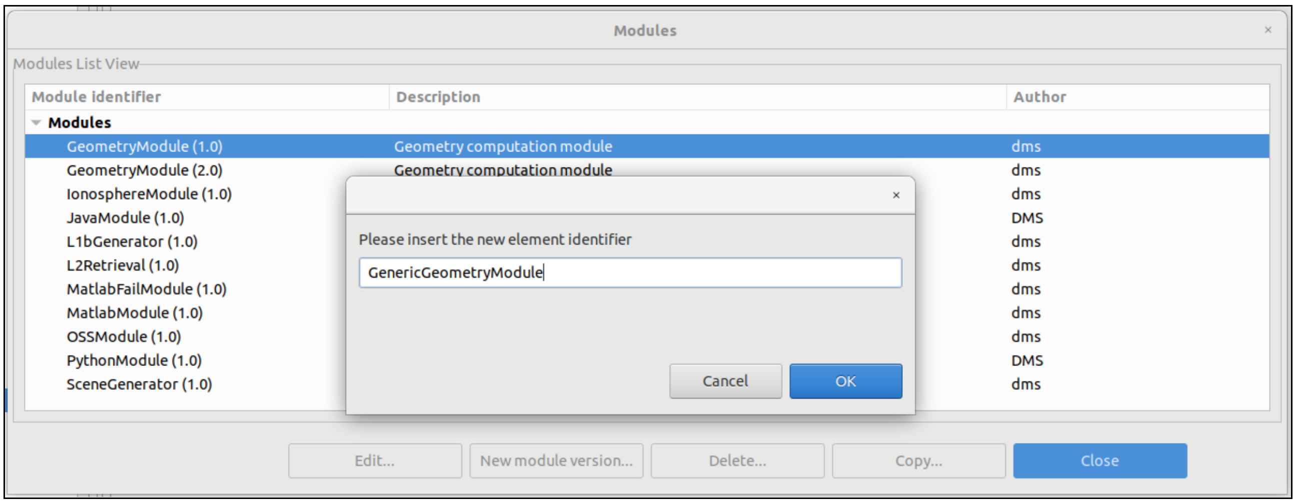
This frame is intended to let users modify data of a certain module. Once they have finished with the editing, they can accept or cancel the changes made with the buttons at the bottom-side toolbar.

4.8.6. Module deletion

Users can select a certain module and choose the option to delete it. Once users confirm the operation the module is erased from the repository and the file system. Note that also every simulation (and its results) that uses this module will also be erased from the system for consistency purposes.

4.8.7. Module copy

Users can select a certain module and choose the option to copy it. The user needs to specify a new name for the module, unique with respect to the existing modules. All module definitions shall be copied.



Module identifier	Description	Author
Modules		
GeometryModule (1.0)	Geometry computation module	dms
GeometryModule (2.0)	Geometry computation module	dms
IonosphereModule (1.0)		dms
JavaModule (1.0)		DMS
L1bGenerator (1.0)		dms
L2Retrieval (1.0)		dms
MatlabFailModule (1.0)		dms
MatlabModule (1.0)		dms
OSSModule (1.0)		dms
PythonModule (1.0)		DMS
SceneGenerator (1.0)		dms

Please insert the new element identifier

GenericGeometryModule

Figure 4-49: Modules. Copy Module

4.9. Simulations

According to section 1.3, a simulation is defined as an execution of either an execution of a set of modules (possibly obtained from the ordered set of modules of a simulation) or an iterative execution of a set of modules with different parameter values.

Users can access to the list of simulations existing in the system in the repository view of the side bar or via the “Simulations” menu (Figure 4-50) from the main menu.

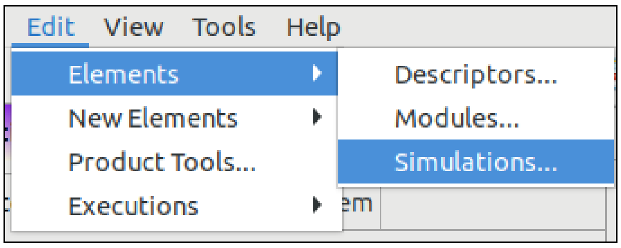


Figure 4-50: Simulation menu

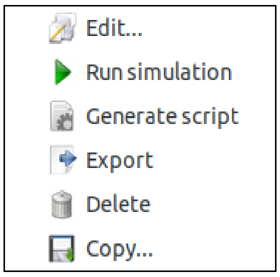


Figure 4-51: Simulation pop-up menu

Operations involving simulations include the following:

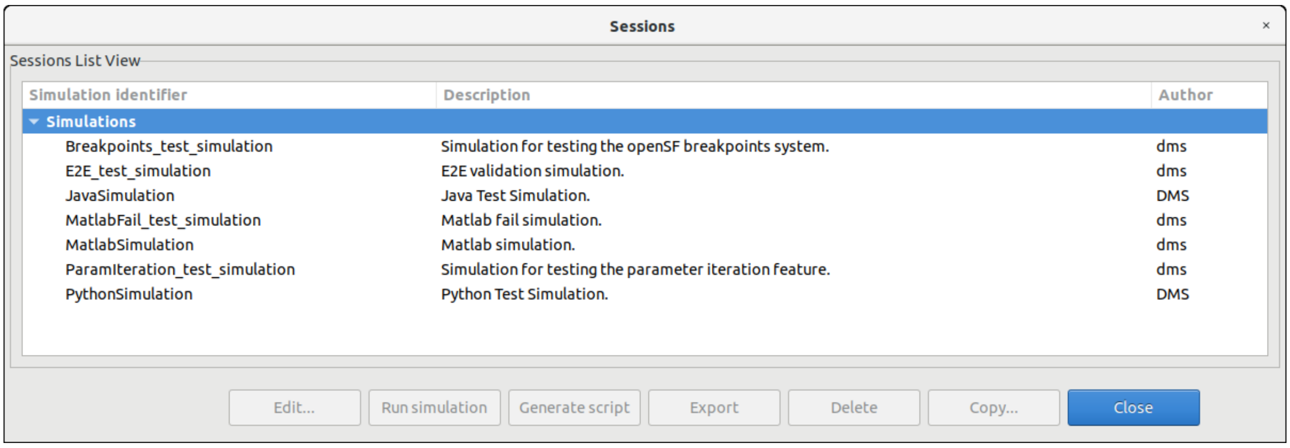
- *List* – present the list of existing simulations;
- *Creation* – capability to create a new simulation into the system;
- *Modification* – edit an existing simulation to enter changes;
- *Deletion* – delete a simulation from the system;
- *Run* – Starts a new simulation execution;
- *Script generation* – creates and stores a script describing the simulation;
- *Exportation* – Exports the entire simulation definition;
- *Copy* – Creates a copy of an existing simulation.

4.9.1. Simulation list

Users can access to the simulation list selecting via the menu bar “Edit”→“Elements”→“Simulations”.

Figure 4-52 shows an example of the simulation list window that is presented upon selection. Below the table including the simulations existing in the system, there is a tool-bar with buttons to access to the different functions listed previously. Users can thus select a certain simulation and perform the operations shown in the toolbar.

Data attributes shown in the simulation list table are simulation ID, description and the name of the author that created the simulation.



Sessions

Sessions List View

Simulation identifier	Description	Author
Simulations		
Breakpoints_test_simulation	Simulation for testing the openSF breakpoints system.	dms
E2E_test_simulation	E2E validation simulation.	dms
JavaSimulation	Java Test Simulation.	DMS
MatlabFail_test_simulation	Matlab fail simulation.	dms
MatlabSimulation	Matlab simulation.	dms
ParamIteration_test_simulation	Simulation for testing the parameter iteration feature.	dms
PythonSimulation	Python Test Simulation.	DMS

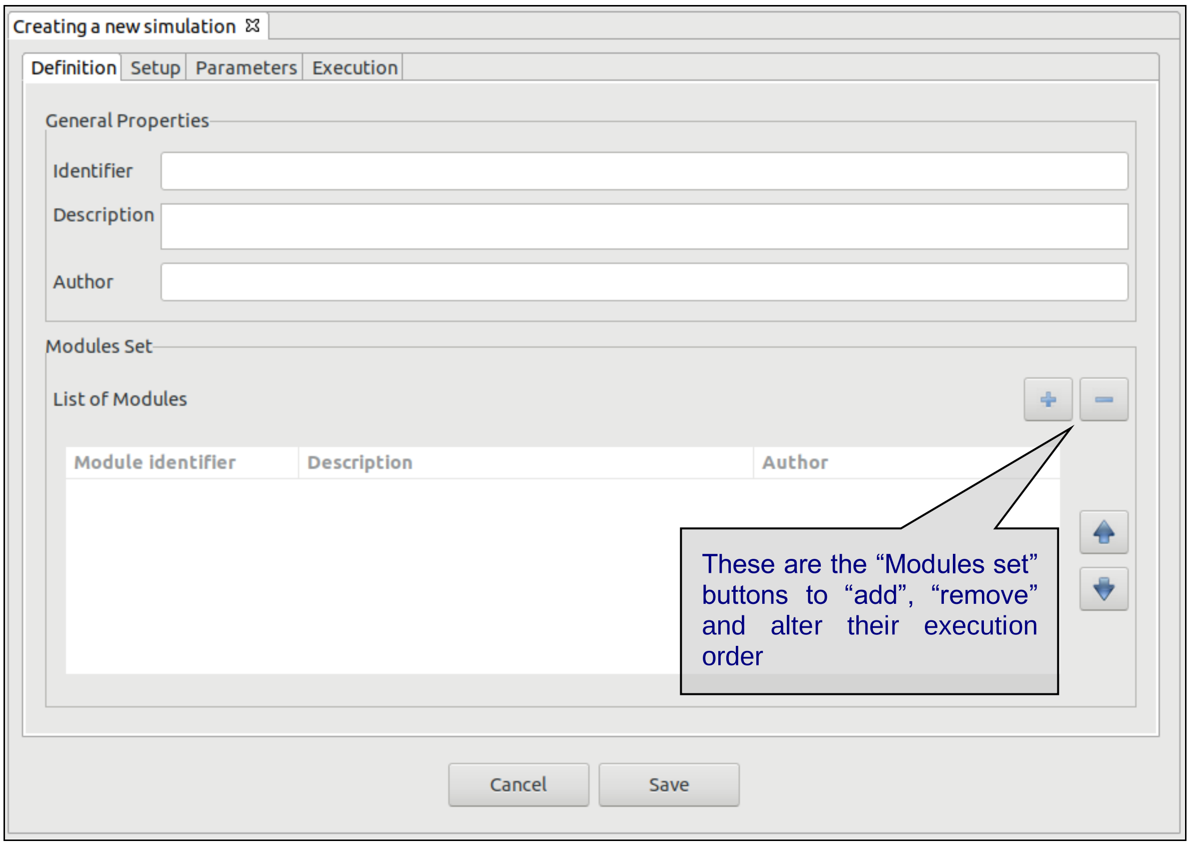
Figure 4-52: Simulation list view

4.9.2. Simulation creation

Users can create a new simulation definition by clicking on the corresponding action from the Simulation menu.

4.9.2.1. General properties

Figure 4-53 shows a blank simulation creation window.



Creating a new simulation

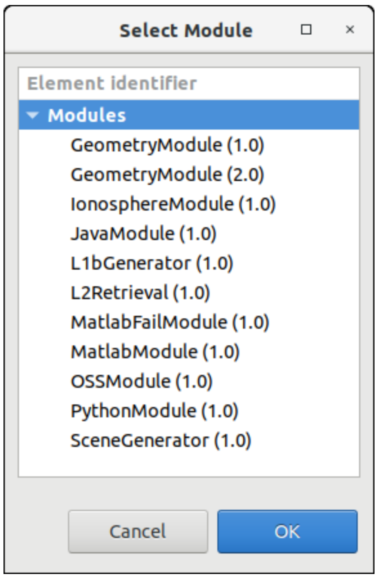
Figure 4-53: Simulation creation - General properties

First thing, the user must fill shall be the following general properties:

Attribute name	Format	Purpose	Sample
Identifier	Medium string. No blank spaces allowed.	Uniquely identifies this simulation definition into the system.	“E2E”
Description	Long string	Brief remarks about the goals and characteristics of the simulation	“This is a full end-to-end simulation for the EarthCARE mission”
Author	Medium string	Name of person or group responsible of the simulation definition	“DMS”

4.9.2.2. Adding/removing modules

First, modules to be executed need to be included by selecting them from a list of available modules. Users just need to click on the desired row at the simulation list (Figure 4-54) and press the “Ok” button.



GeometryModule (1.0)
GeometryModule (2.0)
IonosphereModule (1.0)
JavaModule (1.0)
L1bGenerator (1.0)
L2Retrieval (1.0)
MatlabFailModule (1.0)
MatlabModule (1.0)
OSSModule (1.0)
PythonModule (1.0)
SceneGenerator (1.0)

Figure 4-54: Simulation - Module specification

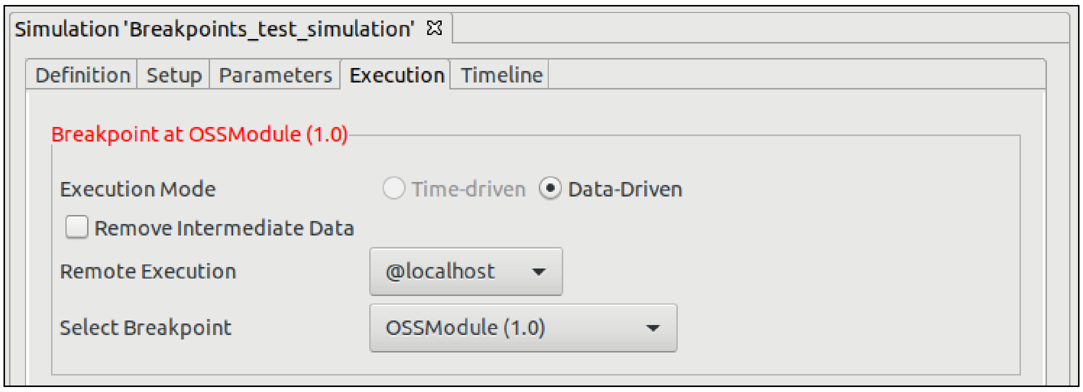
Users can also remove a desired module from the simulation’s modules set by selecting it and pressing the “remove” button.

It is also possible to modify the modules execution order. A selected module can be moved up or down in the order by using the “arrow up” and “arrow down” buttons.

4.9.2.3. Breakpoint Scheduling

Users are able to schedule breakpoints during the simulation execution. A breakpoint is a point where the simulation execution shall stop in a controlled manner, due to system architecture constraints only is possible to interrupt the execution when a determined module has finished the computation and has written the corresponding output.

The user interface for breakpoints addition can be found in the "Execution" tab.


Simulation 'Breakpoints_test_simulation'

Definition | Setup | Parameters | Execution | Timeline

Breakpoint at OSSModule (1.0)

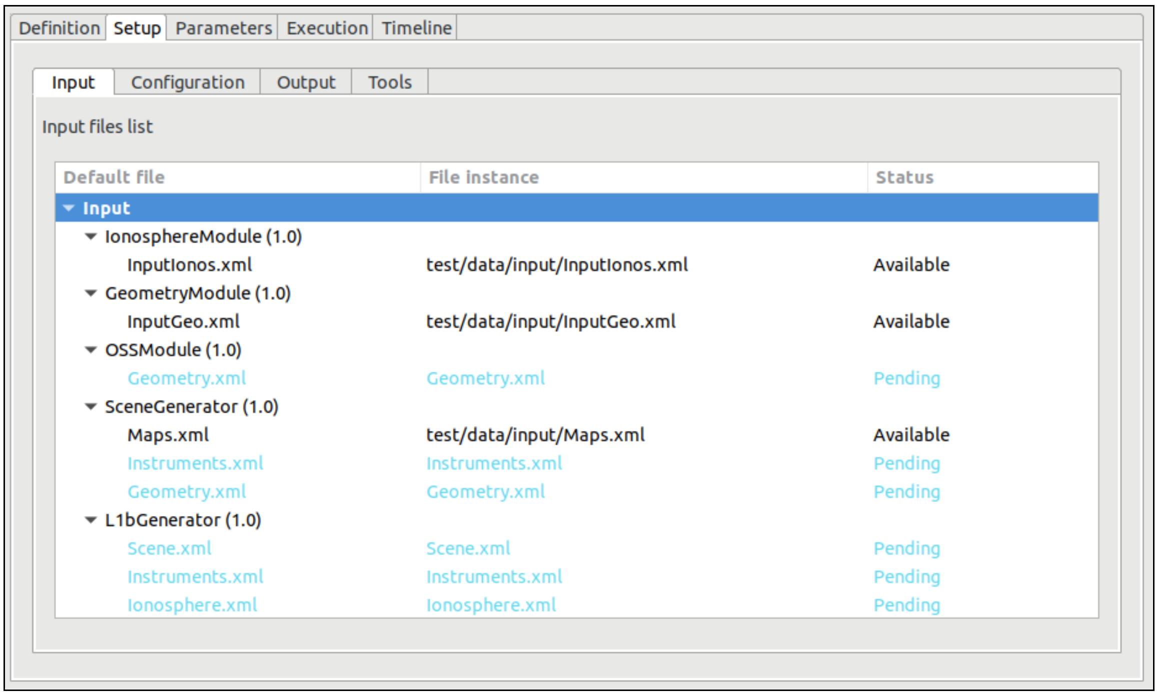
Execution Mode	Time-driven [✓ Data-Driven]
Remove Intermediate Data	☐
Remote Execution	@localhost
Select Breakpoint	OSSModule (1.0)

Figure 4-55: Breakpoint Scheduling Interface

In order to schedule a breakpoint in a module, users shall select its identifier in the "Select Breakpoint" drop-down list box. To remove previously defined breakpoints the user shall select the option "-Remove Breakpoint-" from the same widget.

4.9.2.4. Provision of input data

Selecting the first tab of the simulation setup, (Figure 4-56) the system will ask for the location of the input file list needed to start the simulation.


Definition | Setup | Parameters | Execution | Timeline

Input | Configuration | Output | Tools

Input files list

Default file	File instance	Status
Input		
IonosphereModule (1.0)		
InputIonos.xml	test/data/input/InputIonos.xml	Available
GeometryModule (1.0)		
InputGeo.xml	test/data/input/InputGeo.xml	Available
OSSModule (1.0)		
Geometry.xml	Geometry.xml	Pending
SceneGenerator (1.0)		
Maps.xml	test/data/input/Maps.xml	Available
Instruments.xml	Instruments.xml	Pending
Geometry.xml	Geometry.xml	Pending
L1bGenerator (1.0)		
Scene.xml	Scene.xml	Pending
Instruments.xml	Instruments.xml	Pending
Ionosphere.xml	Ionosphere.xml	Pending

Figure 4-56: Simulation - Inputs definition

Double-clicking on the "file instance" column gives you the possibility of writing the file path. Note that at the time of defining the modules (see chapter 4.8.3), input files are defined from a general point of view (e.g., a file with ground altitude). It is in this step where a user can physically locate each file with the required data.

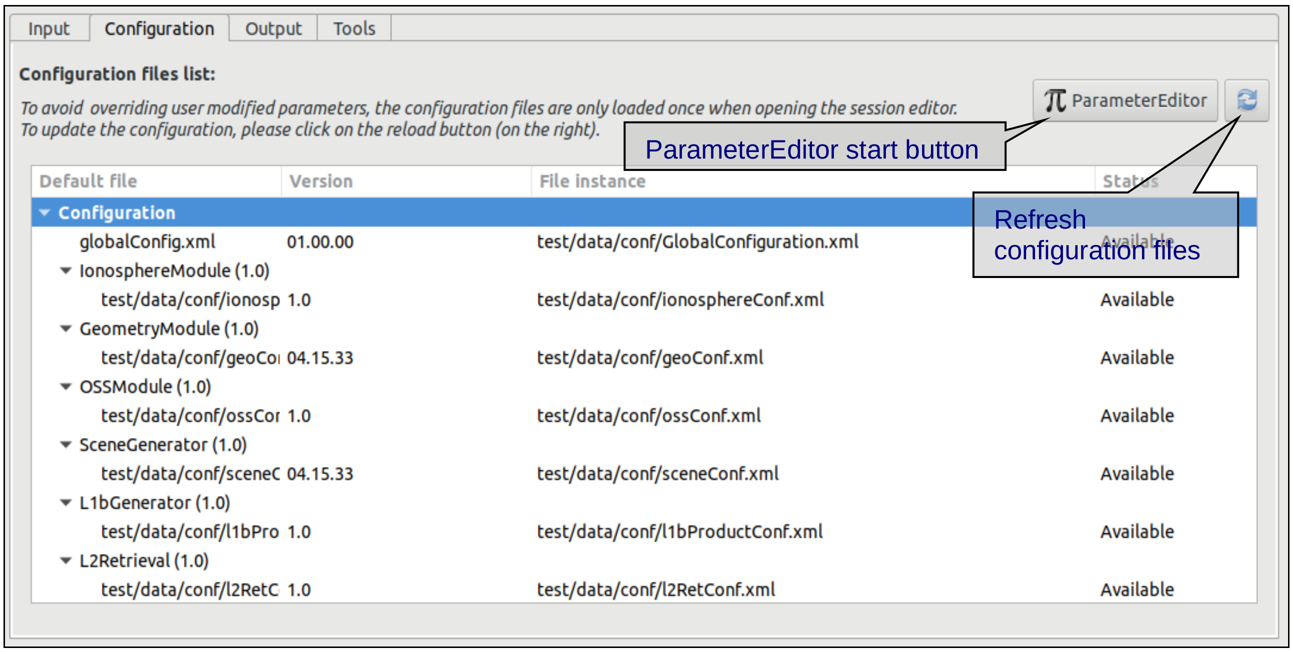
The "Status" column will show one of three different options for each file:

- ❑ Available (green) – the file instance is present, so the file is ready to be used for the module executions;
- ❑ Pending (blue) – the file is not present but it will be generated for the module executions before needed;
- ❑ Missing (red) – the file is not present and is not scheduled to be generated before needed. Edit the file instance and change it so that it appears as an existing file.

File instance locations can be specified using absolute paths or $OPENSF_HOME relative paths.

4.9.2.5. Provision of configuration files

As it is done at the "input data" step, the simulation needs to be provided with the location of configuration files needed by all the modules involved in the simulation (Figure 4-57).



Configuration files list:

To avoid overriding user modified parameters, the configuration files are only loaded once when opening the session editor.
To update the configuration, please click on the reload button (on the right).

Default file	Version	File instance	Status
Configuration			
globalConfig.xml	01.00.00	test/data/conf/GlobalConfiguration.xml	Available
IonosphereModule (1.0)			
test/data/conf/ionosp	1.0	test/data/conf/ionosphereConf.xml	Available
GeometryModule (1.0)			
test/data/conf/geoCo	04.15.33	test/data/conf/geoConf.xml	Available
OSSModule (1.0)			
test/data/conf/ossCor	1.0	test/data/conf/ossConf.xml	Available
SceneGenerator (1.0)			
test/data/conf/sceneC	04.15.33	test/data/conf/sceneConf.xml	Available
L1bGenerator (1.0)			
test/data/conf/l1bPro	1.0	test/data/conf/l1bProductConf.xml	Available
L2Retrieval (1.0)			
test/data/conf/l2RetC	1.0	test/data/conf/l2RetConf.xml	Available

Figure 4-57: Simulation - Configuration files definition

This window will present the list of modules present in the simulation and will ask for the location of each needed configuration file. At this stage, the modules cannot be changed nor deleted.

Double-clicking on a file row, the system will show a file browser to locate a specific configuration file. Providing an existing file will update the status to "Available".

The configuration file panel has two buttons providing the following capabilities:

- ❑ *ParameterEditor button* - located at the top of the configuration files list launches the ParameterEditor application with the selected configuration files already loaded. See [RD-PE].
- ❑ *Refresh button* - reloads all configuration files, reading them from the file system and updating the simulation parameters. It is recommended to do this after editing configuration files with another tool. **Caution: All changes in parameter values made from openSF interface will be lost.**

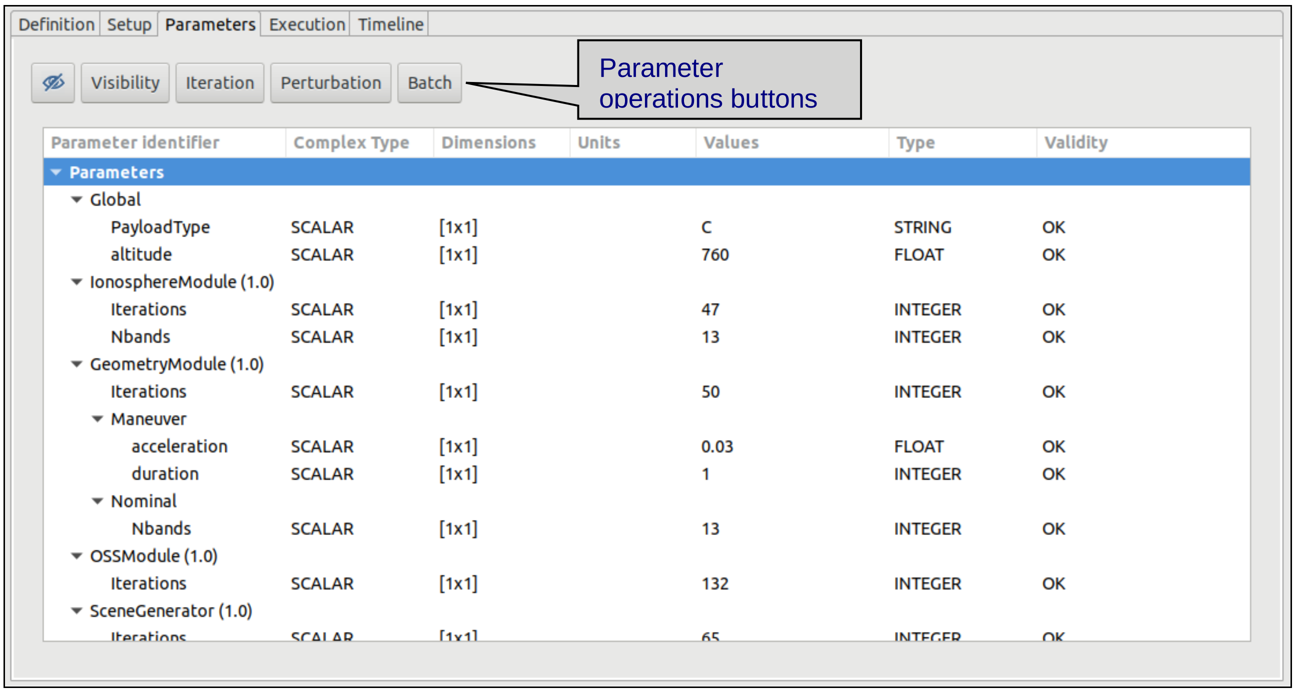
Note that the configuration files for all active modules, plus the global configuration file for the simulation, must be available in order for a simulation to run.

4.9.2.6. Provision of global configuration file

As it has been seen in the previous section, is possible to provide the location of configuration files. The same action is possible to specify where the global configuration file is located. Global configuration file location can be edited every time the user creates a new simulation and can be found in the "Configuration" tab of the "Simulation Setup" panel.

4.9.2.7. Parameters configuration

In this tab, the user is able to alter the contents of the configuration files to change the behaviour of the module. Assuming that the module has been correctly integrated into the system and a valid configuration file is reported, this tab shown in Figure 4-58 will present the list of module parameter (and values) grouped by modules (sorted in execution order).



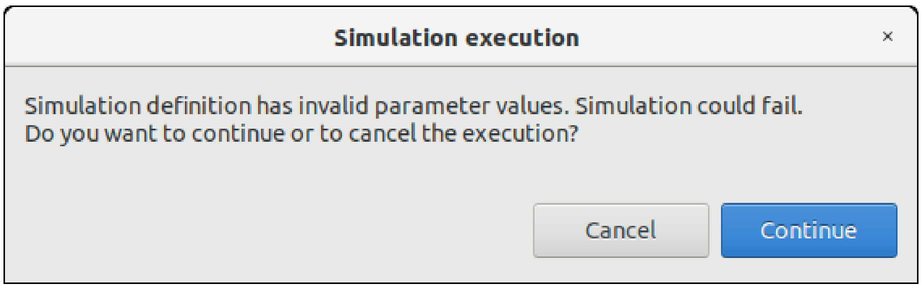
Parameter identifier	Complex Type	Dimensions	Units	Values	Type	Validity
Parameters						
Global						
PayloadType	SCALAR	[1x1]		C	STRING	OK
altitude	SCALAR	[1x1]		760	FLOAT	OK
IonosphereModule (1.0)						
Iterations	SCALAR	[1x1]		47	INTEGER	OK
Nbands	SCALAR	[1x1]		13	INTEGER	OK
GeometryModule (1.0)						
Iterations	SCALAR	[1x1]		50	INTEGER	OK
Maneuver						
acceleration	SCALAR	[1x1]		0.03	FLOAT	OK
duration	SCALAR	[1x1]		1	INTEGER	OK
Nominal						
Nbands	SCALAR	[1x1]		13	INTEGER	OK
OSSModule (1.0)						
Iterations	SCALAR	[1x1]		132	INTEGER	OK
SceneGenerator (1.0)						
Iterations	SCALAR	[1x1]		65	INTEGER	OK

Figure 4-58: Simulation - Parameters definition

Users can consult the following list of attributes:

Attribute name	Format	Purpose	Sample
Parameter identifier	Long string	Complete name of the parameter. A parameter identifier is formed by its path into the file structure (dot separated) and its parameter name.	"parameters. execution_mode"
Description	Long string	Brief description of the parameter purposes and values	"USER of CFI orbit"
Type	Options: INTEGER, FLOAT, STRING, BOOLEAN, FILE, TIME and FOLDER	Parameter values type. Used to present different editor when editing the parameter value. See AD-E2E.	"STRING"
Values	(Different types)	Parameter value. A unique value for each parameter is needed	"CFI"
Complex Type	Options: SCALAR, ARRAY, MATRIX	Parameter structured type. Used to indicate the multidimensional type of the parameter. See AD-E2E.	ARRAY
Dimensions	[*cols* x *rows* x *layers*]	Size of the dimensions. Layers are only displayed if applicable.	[3x2x3]
Units	String	Physical units of measurements if applicable.	"m/s"
Validity	Options: OK, Unknown, TypeMismatch, DimsMismatch, OutOfRange.	Status of the parameter value integrity. openSF checks the parameter validity anytime its value is updated.	"OK"

Only the "values" column is editable to the user, the others just present useful information describing each parameter. openSF checks the parameter validity anytime a run simulation is initiated, if any parameter is not valid the system shows a warning message.


Simulation execution

Simulation definition has invalid parameter values. Simulation could fail.
Do you want to continue or to cancel the execution?

Figure 4-59: Simulation execution, warning message

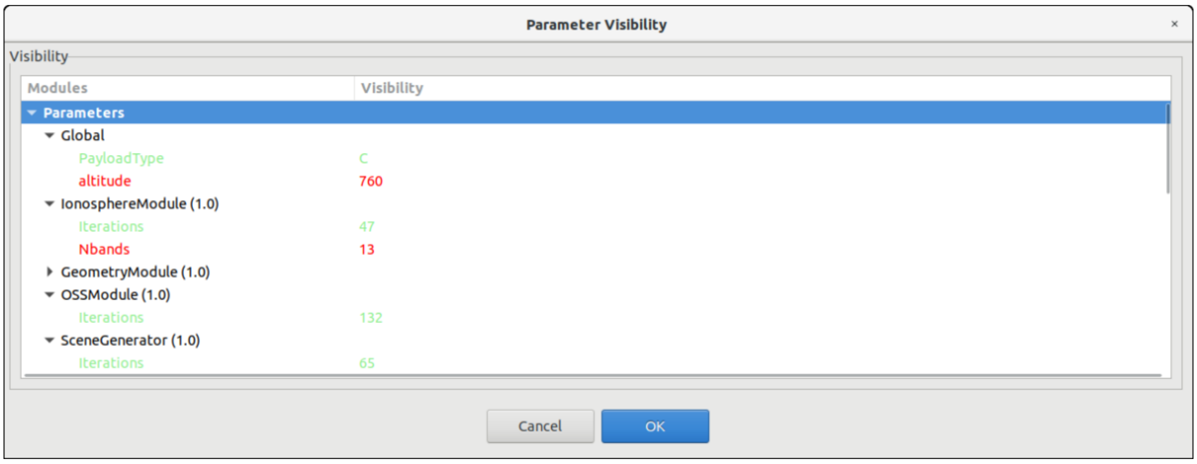
Note that changing the value of these variables will not affect to the “template” configuration files specified in the “Configuration” tab. The variables involved in a simulation definition are stored in the database with the chosen values, meaning that the simulation will use them during the execution.

The specific format used for the single-line representation is the same as in ParameterEditor. The current representation (for non-scalars) is designed to be copied into Python. The same operation can occur in the other direction, from Python to the application, as long as the following conditions hold:

- All elements are literals. For example, `[1,2,3]` works but `[1,sum(range(1,3))]` does not
- Elements are homogeneous. Thus, `["a",2,True]` is not valid input for a parameter.

Note that this representation may change in future versions, and interoperability with Python (or any specific version or library) is not guaranteed in general.

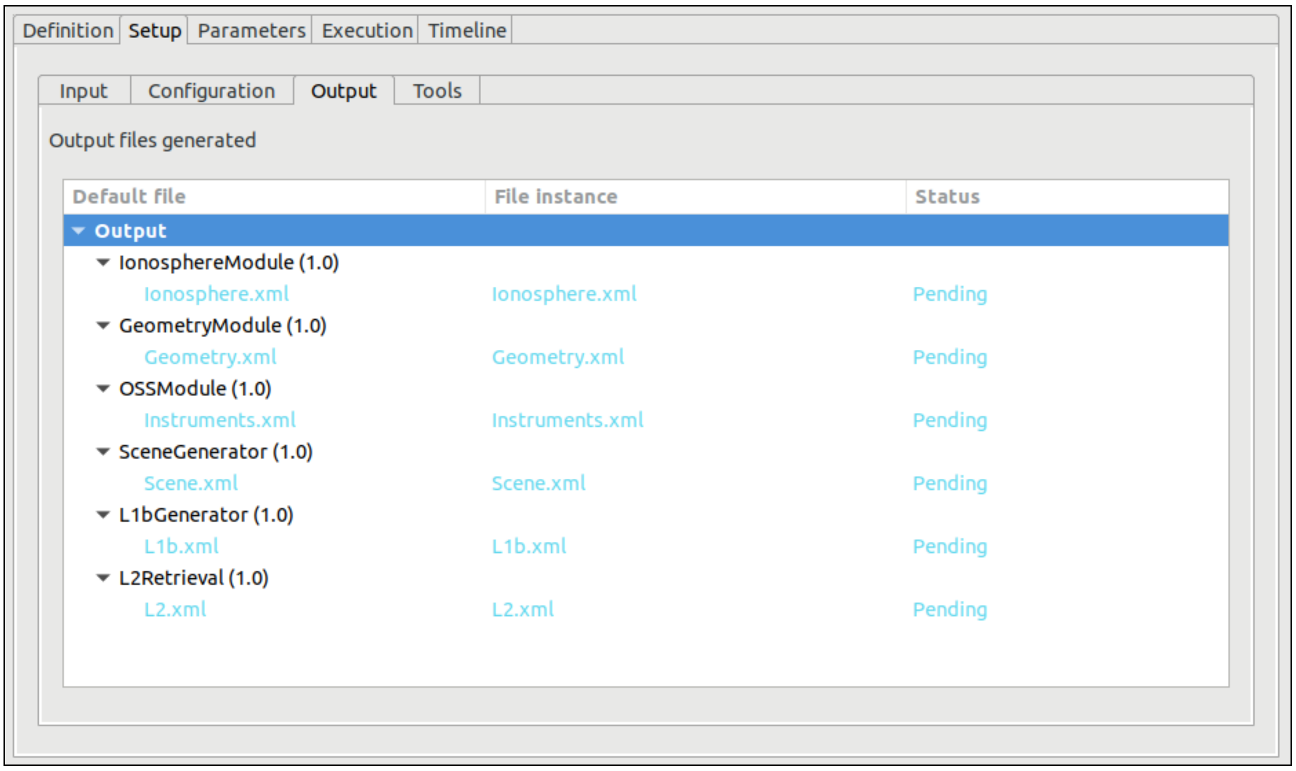
The user can filter the parameters that are displayed in the HMI and which can be altered and/or monitored in simulation execution. The *‘Parameter Visibility’* allows marking each parameter as visible or not. This setting can be done at parameter level, at module level or at the whole simulation level. To do so, the user can open the Visibility view through the dedicated button and toggle the visibility of each parameter with a double left-click over it or by selecting the desired option from the context menu that can be opened with right-click.



Modules	Visibility
▾ Parameters	
▾ Global	
PayloadType	C
altitude	760
▾ IonosphereModule (1.0)	
Iterations	47
Nbands	13
▸ GeometryModule (1.0)	
▾ OSSModule (1.0)	
Iterations	132
▾ SceneGenerator (1.0)	
Iterations	65

Figure 4-60: Visibility view

Once the parameter visibility has been set it can be switched on/off thru the ‘eye’ icon in the simulation setup parameter configuration tab.



Output files generated

Default file	File instance	Status
Output		
IonosphereModule (1.0)		
Ionosphere.xml	Ionosphere.xml	Pending
GeometryModule (1.0)		
Geometry.xml	Geometry.xml	Pending
OSSModule (1.0)		
Instruments.xml	Instruments.xml	Pending
SceneGenerator (1.0)		
Scene.xml	Scene.xml	Pending
L1bGenerator (1.0)		
L1b.xml	L1b.xml	Pending
L2Retrieval (1.0)		
L2.xml	L2.xml	Pending

Figure 4-61: Simulation - Output definition

4.9.2.8. Specification of output files

Users can change the name and location of the output files that will be generated by execution of modules. Selecting the "output" tab, the system will show a list of output files grouped by modules, and following the execution order.

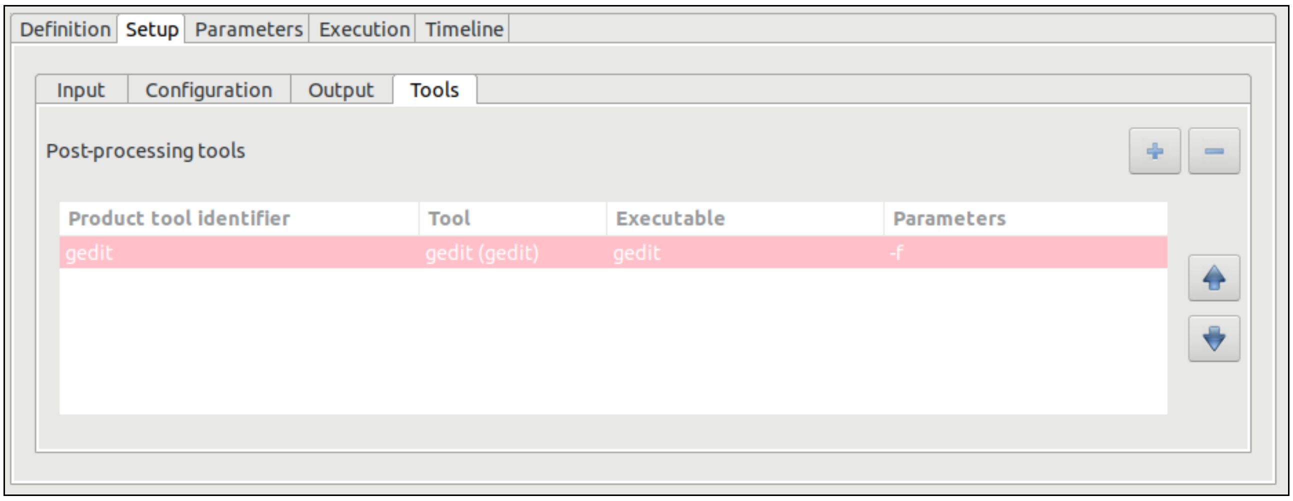
By default, these files will have a "Pending" status, meaning that they will be produced by the action of the module's execution. Once a simulation has been executed and output files generated, file instance column will show the absolute path of the generated file and the status will be "Available".

4.9.2.8.1. Removal of intermediate output files

As shown in Figure 4-55 the user now was an option for removing intermediate output files. By activating this option, a simulation executing will remove from the simulations directory any output files not generated by the last module of a simulation execution and that correspond to intermediate data of a step in the module chain.

4.9.2.9. Specification of final product tools

It is possible to add a list of product tools as post-processing operations, that is, a series of executables to be called upon the execution completion (Figure 4-62).



Product tool identifier	Tool	Executable	Parameters
gedit	gedit (gedit)	gedit	-f

Figure 4-62: Simulation - Product tools specification

There are two ways to add tools to this list:

- ❑ Selecting a file from the input, configuration or output files list. Users can right-click on a file marked as “available” or “pending” and a pop-up menu will appear. This menu will show a list of tools that can be applied to that certain file[9]. These tools can be executed instantly (if the file is already “available” or be scheduled to end of the execution process). Users can change the default parameters for the tool execution. Figure 4-63 shows the contextual menu that pops up when the user right-clicks on a file from the Simulation Creation/Edition view.

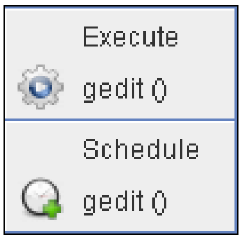


Figure 4-63: File contextual menu.

When scheduling actions to certain files, openSF uses, instead of the actual file name and location, a reference to the file’s foreseen location as an environment variable. These variables are named starting with the dollar symbol, then “IO”, the simulation number and its identifier with no blanks, underscores or dots. For example, $IO0orbitxml denotes the orbital file to be generated in the proper folder by the execution process and $IO1radaroutputnc, the NetCDF file generated by the radar module in the second simulation of a given simulation.

In case the selected tool is an “external” tool (as described in section 4.5.4) the HMI will prefix the $OPENSF_HOME variable to form the absolute path of the file. Following the previous example, if users want to view the contents of the orbit XML file with an external viewer, HMI will present $OPENSF_HOME/$IO0orbitxml. If users want to plot the radar output (using an internal tool) HMI will present only $IO1radaroutputnc.

[9] Tools defined in the system have an action associated to a file extension.

It is also possible to use two other simulation-related variables: $SIMULATION_FOLDER to point the foreseen location of the simulation execution and $SIM_FOLDER_# (with "#" denoting the simulation number) pointing the foreseen location of a certain simulation in current simulation.

In the same way, users can include references to the rest of openSF environment variables like $OPENSF_HOME.

❑ Clicking on the "add tool to simulation" button. Upon a selection of this action, users can choose one tool from the appearing list of defined tools. Users can change the default parameters for the tool execution.

Users can also select a certain tool and remove it from the list and, alternatively, change the order of execution of the tools with the arrow buttons besides. In any case product tools are always executed at the end of the executed simulation.

Use Example:

A simple simulation involving only one module execution. The scenario is composed by the following elements:

❑ Simulation name: *simulationTest*

❑ Module name: *moduleTest*

❑ Input for the module: *inputTest.txt* located in OPENSF_HOME folder

❑ Output generated: *outputTest.txt*

❑ Module configuration: *globalConfig.xml* and *localConfig.xml*

❑ OPENSF_HOME variable points to */home/tester/openSF/*

❑ The simulation folder is */home/tester/openSF/simulations/*

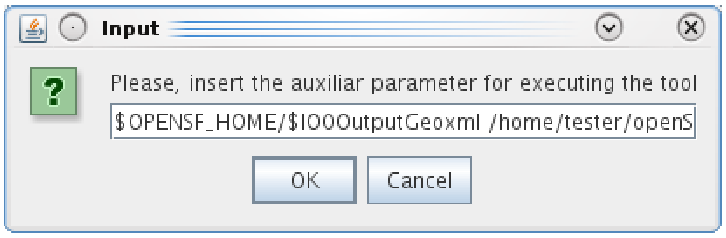
❑ Tool defined associated to txt extension: *meld*

It is desired to compare the input and output files with a visual *diff* like application named meld (http://meld.sourceforge.net/). In this case the syntax for the tool would be:

❑ For input file user can use the original location or the foreseen location where openSF copies that file: */home/tester/openSF/inputTest.txt* or *$IO0inputTesttxt*

❑ Foreseen location for output file: *$IO0outputTesttxt*

❑ As explained before there are different mechanism to schedule the execution of the tool. The one recommended in this case is to right click on the output file whose status is pending and click on the "*meld*" tool under the *schedule* title. A pop-up window shall appear with a text field presenting the variable for the output file location *$IO0outputTesttxt*. User shall complete the syntax for the tool appending the location of the input file (*/home/tester/openSF/inputTest.txt* or *$IO0inputTesttxt).*



Please, insert the auxiliar parameter for executing the tool

$OPENSF_HOME/$IO0OutputGeoxml /home/tester/openS

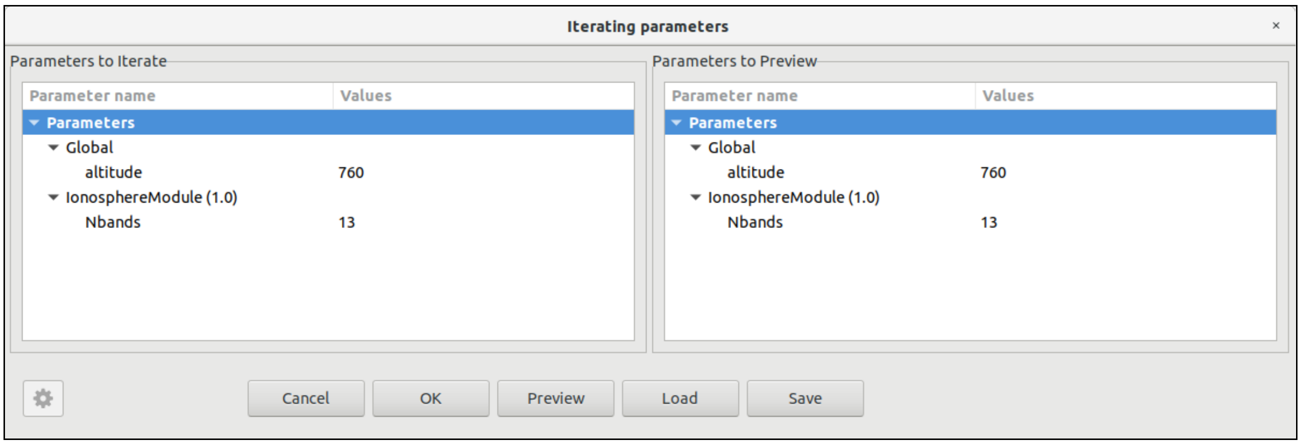
Figure 4-64: Tool parameters specification

- The value of the other simulation related variables would be (DATE represents the date when simulation was executed):
    - $SIMULATION_FOLDER = /home/tester/openSF/simulations/simulationTest$DATE/
    - $SIM_FOLDER_0 = /home/tester/openSF/simulations/simulationTest$DATE/0-simTest/

4.9.2.10. Iterative simulations - iterating input/output files and parameter values

Users can assemble iterative simulations. This is a powerful feature that helps to run a large number of simulations by changing values of the parameters. Users can alter any parameter's value to fine-tune the behaviour of a module for a particular simulation run.

Selecting one or many parameters and pressing on the "Iterate" button, will open the dialog shown in Figure 4-65. In this example we are going to iterate two float parameters from two different modules.


Iterating parameters

Parameters to Iterate

Parameter name	Values
Parameters	
Global	
altitude	760
IonosphereModule (1.0)	
Nbands	13

Parameters to Preview

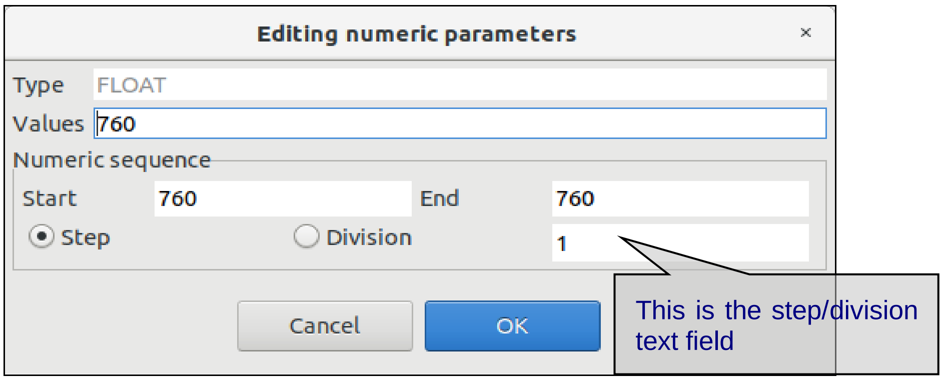
Parameter name	Values
Parameters	
Global	
altitude	760
IonosphereModule (1.0)	
Nbands	13

Figure 4-65: Simulation creation - Iterating parameters

This figure shows the initial state of the dialog. The list of selected parameters is in the left table and a preview of the modules upon the combination of all the different parameter values on the right table.

Accessing to the "values" column of the left table, users can input a list of valid values separated only by blank spaces or commas, but not both.

By selecting a parameter and clicking the cogwheel icon, or by double-clicking (left or right button, depending on the OS) on a parameter, the user can open the following "numeric sequence generator" dialog for an advanced customization of the iteration values:


Editing numeric parameters

Type: FLOAT
Values: 760
Numeric sequence
Start: 760 End: 760
[✓ Step] Division: 1

Figure 4-66: Simulation - Creation - Editing numeric sequences

This dialog lets the user define a numerical sequence of values (of the selected type: FLOAT or INTEGER) in three different ways:

- User input. Users can introduce their own values using the “Values” text field.
- Numeric sequence by step. Once defined the starting (x_1) and ending (x_n) values of the sequence, users can input the value of the step (s) in the step/division text field. Pressing the “generate” (marked with the ellipsis symbol) will create an arithmetical sequence following this rule:

  $$\{x_1, x_1 + s, x_1 + 2s, \ldots\}$$

  Numeric values will never be greater than the upper limit. For example, a numeric sequence starting from 1 to 10 with a step of 5 will generate a series of (1, 6).
- Numeric sequence by division. Once defined the starting (x_1) and ending (x_n) values of the sequence, users can input the number of divisions (d) in the step/division text field. Pressing the “generate” (marked with the ellipsis symbol) will create an arithmetical sequence following this rule:

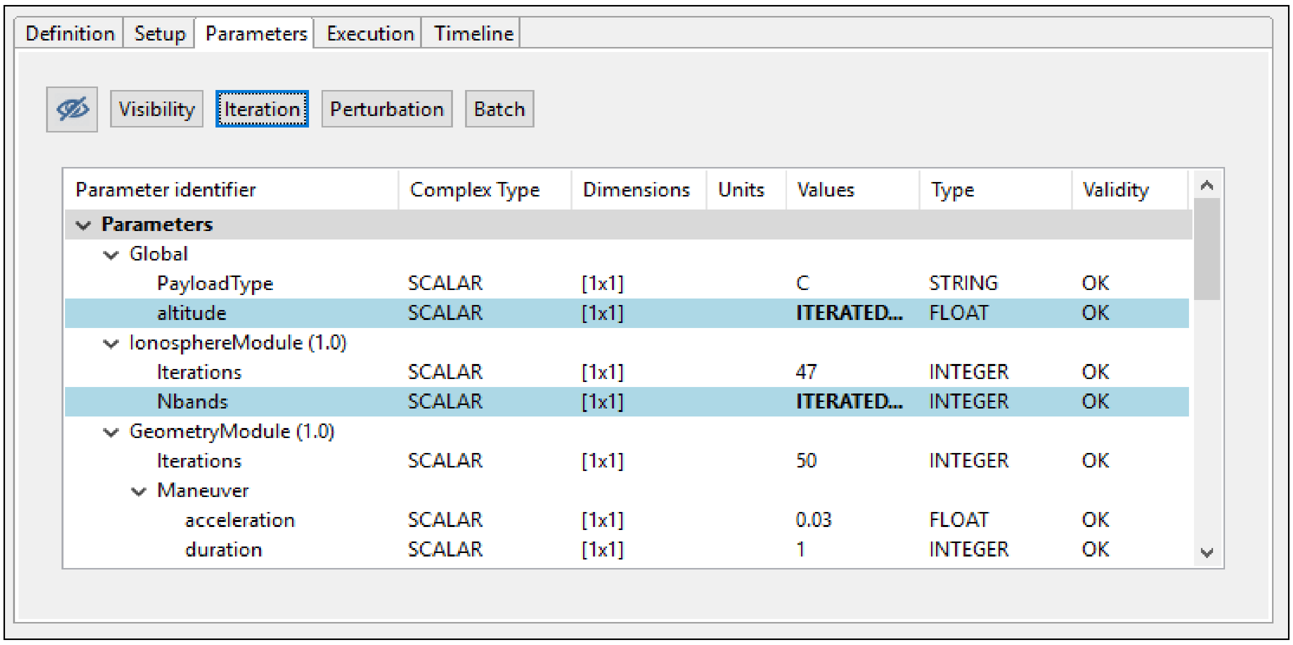
  $$\{x_1, x_1 + s, x_1 + 2s, \ldots\}, s = \frac{(x_n - x_1)}{d}$$

  Numeric values will never be equal or greater than the upper limit. For example, a numeric sequence starting from 0 to 10 with five divisions will generate a series of (0, 2, 4, 6 and 8).

Users can now accept or cancel the numerical sequence.

Once a valid set of values is input in both parameters, users can press the “preview” button to consult the configuration of the iterated parameters. Note that parameters not involved in the iteration will remain fixed to a value but they can be manually changed as seen in chapter 4.9.2.5.

The iterated parameters will be highlighted in the simulation parameters’ tab as shown in the Figure 4-67 and iterate view can be opened again for more customization with double-click on an iterated parameter.



Parameter identifier	Complex Type	Dimensions	Units	Values	Type	Validity
Parameters						
Global						
PayloadType	SCALAR	[1x1]		C	STRING	OK
altitude	SCALAR	[1x1]		ITERATED...	FLOAT	OK
IonosphereModule (1.0)						
Iterations	SCALAR	[1x1]		47	INTEGER	OK
Nbands	SCALAR	[1x1]		ITERATED...	INTEGER	OK
GeometryModule (1.0)						
Iterations	SCALAR	[1x1]		50	INTEGER	OK
Maneuver						
acceleration	SCALAR	[1x1]		0.03	FLOAT	OK
duration	SCALAR	[1x1]		1	INTEGER	OK

Figure 4-67: Simulation with iterated parameters

Readers must know that the openSF system can filter redundant modules out of an execution process. There is more information in chapter 4.9.5.

4.9.2.10.1. Parameter iteration definitions files

Upon defining an iterative configuration, the user can save this definition in a parameter iteration definition file. This is accomplished by using the "Save" button in the dialog shown in Figure 4-65. Additionally, the user can load a previously defined iteration configuration from file (using the "Load" button).

The format of the files used to store the iteration configuration definitions is compliant with the configuration parameter file format as specified in [AD-E2E].

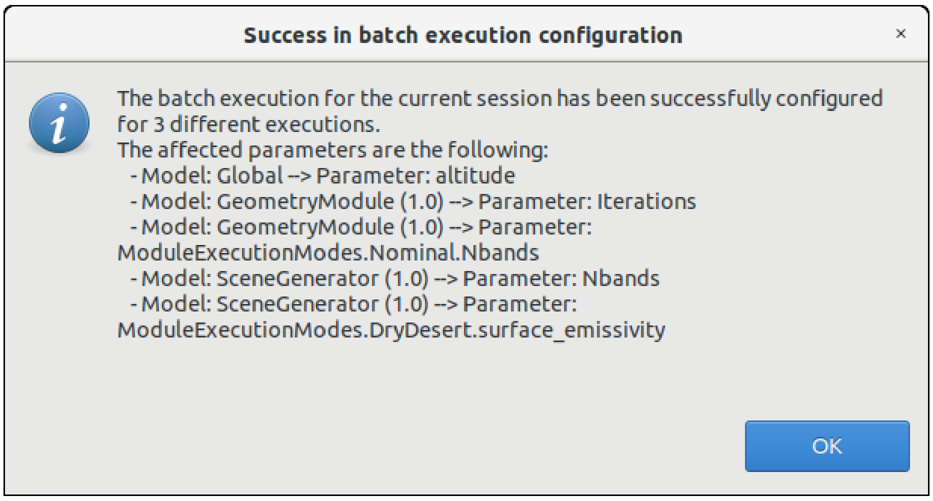
4.9.2.11. Batch simulation

Section 4.9.2.10 explains how the user can configure an iterative simulation, allowing the user to define multiple values for the desired parameters an openSF will automatically perform a permutation between all the possible combinations of values and run all the cases.

In situations where the user needs to run multiple variations of an original simulation with some customization of the parameters' values (but not all the possible combinations of those values), openSF enables the definition of a batch execution. This ability aims to provide fine grained control over the number of executed variations of the original simulation and the parameter values that are customized for each of them.

The configuration of a batch simulation requires the creation of a *"batch simulation configuration file"*. The syntax and format of this type of file is described in [AD-E2E]. This file can be loaded through the *"Batch"* button available in the simulation's parameters tab.

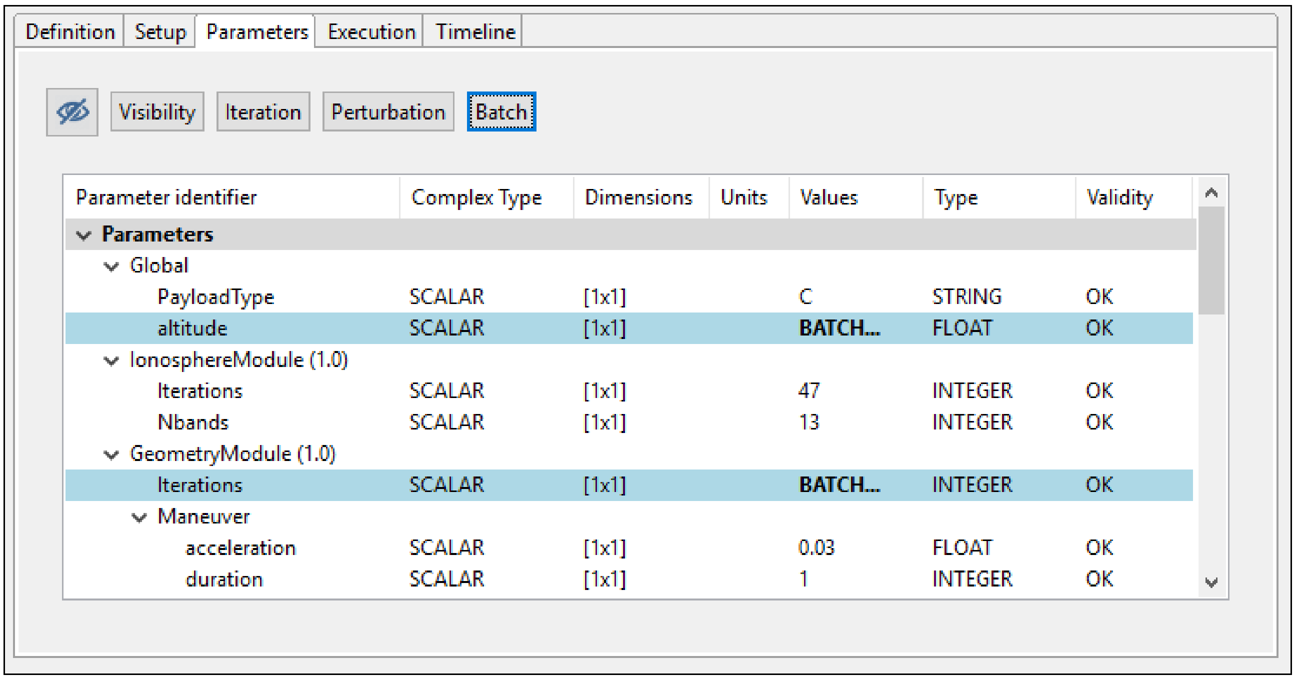
Upon loading the file, OpenSF checks the format and imports the contents. The simulation will be successfully configured and a confirmation message will be displayed with the total number of simulation executions after the simulation template spawning together with the id and module names of the overridden parameters.


Success in batch execution configuration

The batch execution for the current session has been successfully configured for 3 different executions.
The affected parameters are the following:
- Model: Global --> Parameter: altitude
- Model: GeometryModule (1.0) --> Parameter: Iterations
- Model: GeometryModule (1.0) --> Parameter: ModuleExecutionModes.Nominal.Nbands
- Model: SceneGenerator (1.0) --> Parameter: Nbands
- Model: SceneGenerator (1.0) --> Parameter: ModuleExecutionModes.DryDesert.surface_emissivity

Figure 4-68: Successful batch configurated simulation message

As was the case with the overridden parameter values configured through the "Iterate" option, the overridden parameters are highlighted in the simulation's parameters tab as shown in the Figure 4-69.



Parameter identifier	Complex Type	Dimensions	Units	Values	Type	Validity
Parameters						
Global						
PayloadType	SCALAR	[1x1]		C	STRING	OK
altitude	SCALAR	[1x1]		**BATCH...**	FLOAT	OK
IonosphereModule (1.0)						
Iterations	SCALAR	[1x1]		47	INTEGER	OK
Nbands	SCALAR	[1x1]		13	INTEGER	OK
GeometryModule (1.0)						
Iterations	SCALAR	[1x1]		**BATCH...**	INTEGER	OK
Maneuver						
acceleration	SCALAR	[1x1]		0.03	FLOAT	OK
duration	SCALAR	[1x1]		1	INTEGER	OK

Figure 4-69: Simulation with overridden parameters through the batch option

To ease the use of this feature, together with the examples of the validation database, an example of a *"batch simulation configuration file"* for the *"E2E_test_simulation"* is provided and can be found in the following path: *$INSTALL_DIR/test/data/batch/E2E_test_simulation_batch.xml*.

© DEIMOS Space S.L.U., 2019

4.9.2.12. Simulation Perturbation

Figure 4-58 shows a *Parameter Perturbation* option that covers the capability of adding a perturbation function to a module parameter. This functionality is further described in section 4.14.

4.9.3. Simulation Modification

It is possible to edit a given simulation and create a different one altering the information previously stored. Consequently, changes made to the simulation will not alter the previous but will create another.

When editing a simulation, the system will show the same window as shown in the simulation creation. But this time, all the information concerning this simulation will fill every data field.

Once finished, the user can cancel the changes or execute this new simulation pressing the "Run" button.

4.9.4. Simulation deletion

Users can select a certain simulation and choose the option to delete it. Once the operation is confirmed the simulation is deleted from the repository and the file system. **It is to be noticed that for consistency reasons, the simulation deletion causes the removal of all results generated from that simulation from the system**.

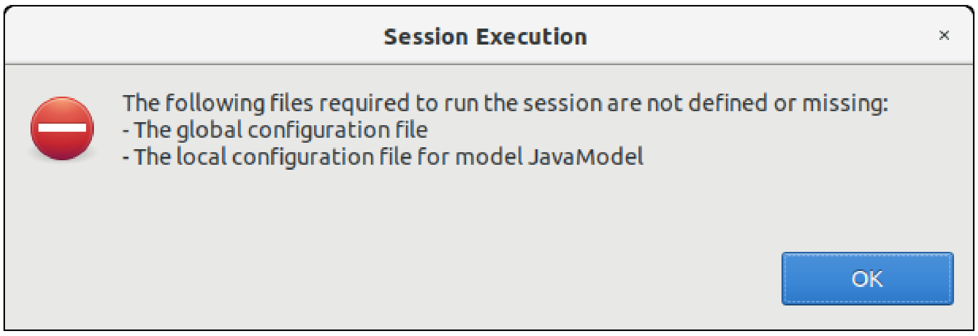
4.9.5. Simulation execution - run

Once a simulation definition is ready, users can execute it.

The effective execution order of the simulation modules is determined by openSF, based on the input/output dependencies – see section 4.9.6.1 for further details.

Upon the activation of the "run" command, the system performs a series of checks to ensure the validity of the simulation:

- ❑ If the global configuration file for the simulation, or the local configuration file of any *active* module in the simulation is missing, the execution will not proceed, and a dialog will be displayed listing the missing files as shown below.

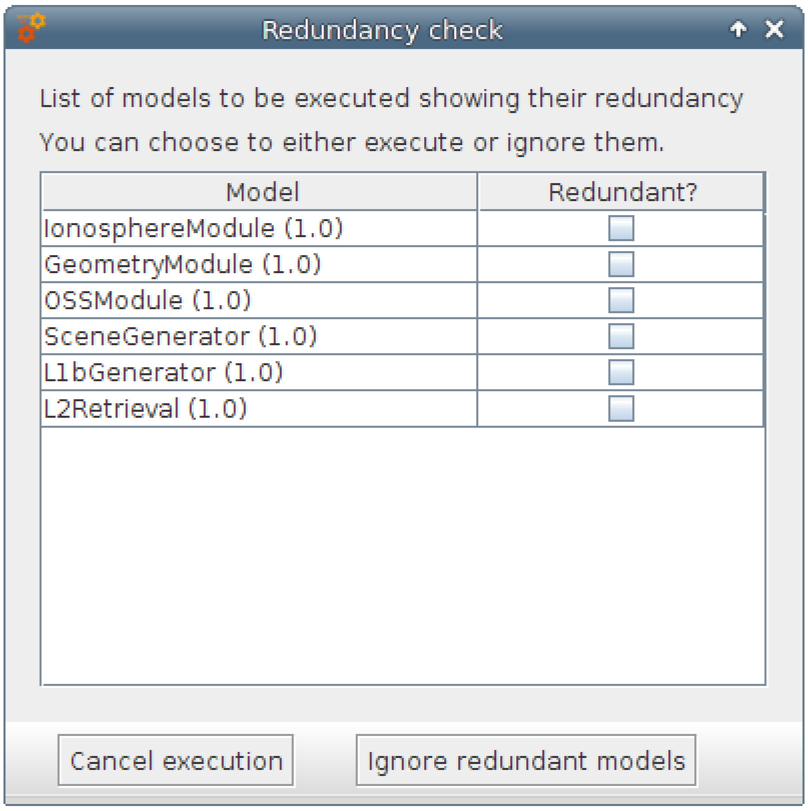


The following files required to run the session are not defined or missing:
- The global configuration file
- The local configuration file for model JavaModel

Figure 4-70: Simulation execution - Execution prevented due to missing configuration files

- ❑ If there is any other file with the "missing" status (that is, the system is unable to find in the given location), openSF **will assume** that this file will be in the right place when needed, so it leaves the responsibility of placing it in the correct place to the user or to process outside the system. A fatal error is likely to be raised by a module if it cannot locate a needed file.

- If there is any parameter not valid, openSF will show a message warning the user. The simulation can be run with parameters in a not valid state, but modules may then raise errors and stop the execution if they cannot parse the value, if a parameter points to a non-existing file, etc.
- In some cases, like executing iterative simulations or sequences of the same modules, some of the modules may be redundant (that is, they will generate the same output because they are run with the same input and configuration).

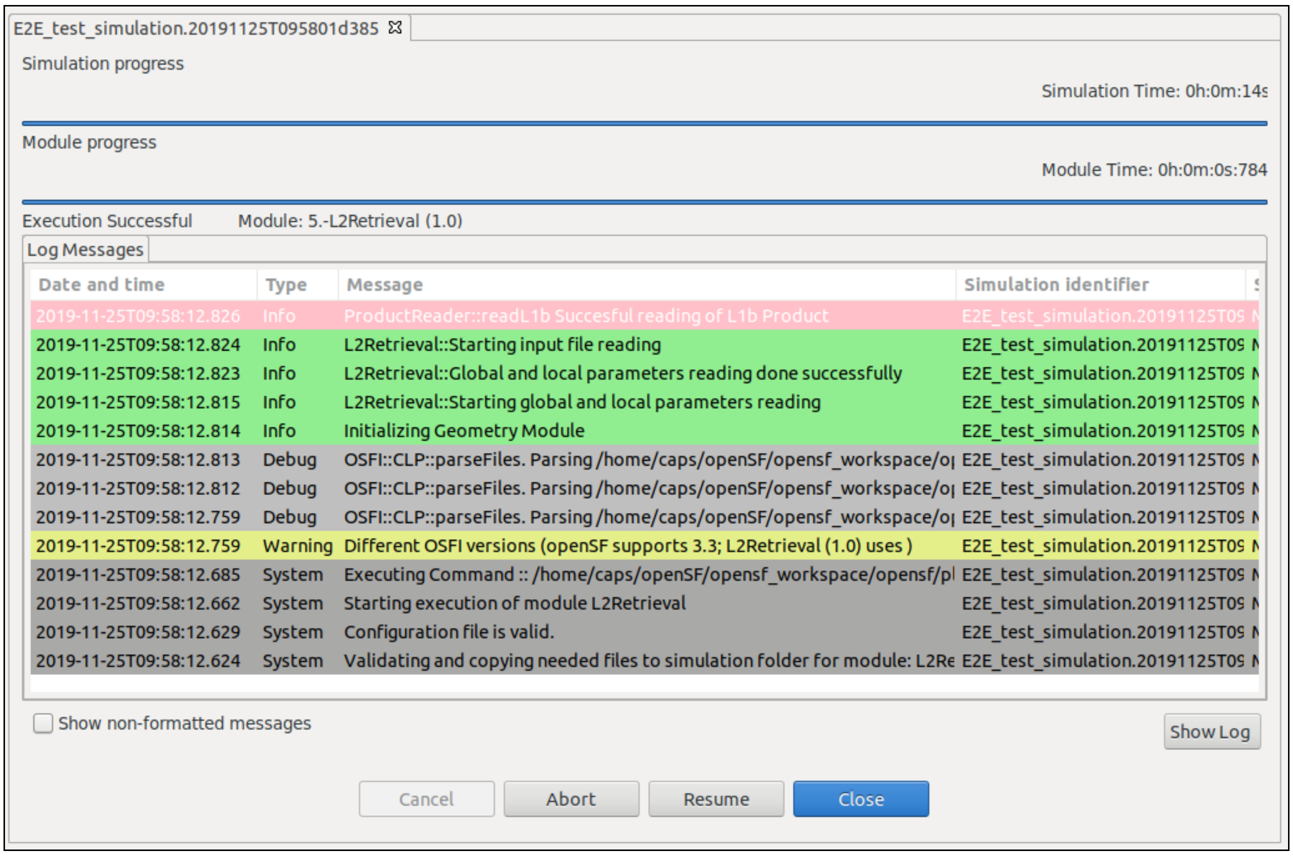

Redundancy check

List of models to be executed showing their redundancy
You can choose to either execute or ignore them.

Model	Redundant?
IonosphereModule (1.0)	☐
GeometryModule (1.0)	☐
OSSModule (1.0)	☐
SceneGenerator (1.0)	☐
L1bGenerator (1.0)	☐
L2Retrieval (1.0)	☐

Figure 4-71: Simulation execution - Redundancy

Figure 4-71 shows the dialog that shall pop-up in case there is any redundancy. Users can choose to either execute or ignore all the modules marked as redundant. Once every validity check is fulfilled, a background process (Figure 4-72) will run the simulation and information concerning it will be given to the user in two different ways. One is the log window and another is the “simulation progress” window. The window presents a progress bar showing the percentage of simulation progress and a button to abort/close the execution.


E2E_test_simulation.20191125T095801d385

Simulation progress

Simulation Time: 0h:0m:14s

Module progress

Module Time: 0h:0m:0s:784

Execution Successful   Module: 5.-L2Retrieval (1.0)

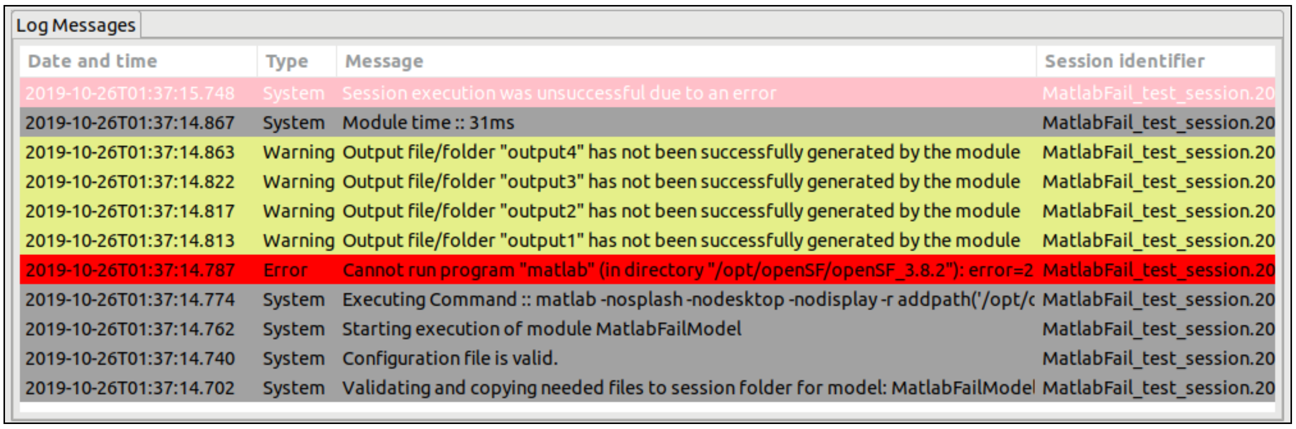
Log Messages

Date and time	Type	Message	Simulation identifier
2019-11-25T09:58:12.826	Info	ProductReader::readL1b Succesful reading of L1b Product	E2E_test_simulation.20191125T09
2019-11-25T09:58:12.824	Info	L2Retrieval::Starting input file reading	E2E_test_simulation.20191125T09
2019-11-25T09:58:12.823	Info	L2Retrieval::Global and local parameters reading done successfully	E2E_test_simulation.20191125T09
2019-11-25T09:58:12.815	Info	L2Retrieval::Starting global and local parameters reading	E2E_test_simulation.20191125T09
2019-11-25T09:58:12.814	Info	Initializing Geometry Module	E2E_test_simulation.20191125T09
2019-11-25T09:58:12.813	Debug	OSFI::CLP::parseFiles. Parsing /home/caps/openSF/opensf_workspace/o	E2E_test_simulation.20191125T09
2019-11-25T09:58:12.812	Debug	OSFI::CLP::parseFiles. Parsing /home/caps/openSF/opensf_workspace/o	E2E_test_simulation.20191125T09
2019-11-25T09:58:12.759	Debug	OSFI::CLP::parseFiles. Parsing /home/caps/openSF/opensf_workspace/o	E2E_test_simulation.20191125T09
2019-11-25T09:58:12.759	Warning	Different OSFI versions (openSF supports 3.3; L2Retrieval (1.0) uses)	E2E_test_simulation.20191125T09
2019-11-25T09:58:12.685	System	Executing Command :: /home/caps/openSF/opensf_workspace/opensf/p	E2E_test_simulation.20191125T09
2019-11-25T09:58:12.662	System	Starting execution of module L2Retrieval	E2E_test_simulation.20191125T09
2019-11-25T09:58:12.629	System	Configuration file is valid.	E2E_test_simulation.20191125T09
2019-11-25T09:58:12.624	System	Validating and copying needed files to simulation folder for module: L2Re	E2E_test_simulation.20191125T09

Figure 4-72: Simulation - Execution progress

During the simulation execution, all events raised by the modules is collected and displayed in the Log Messages table. By default these messages are processed and displayed/coloured based on their type, but OpenSF allows to inspect the original messages by checking the Show non-formatted messages. All event messages are collected and stored in a log file, that can be opened for further inspection using the Show Log button. The collected events can be of one of the following types:

- *System information* – An event with some information to the user is generated by the platform. This is a harmless event; thus, the execution continues with no interruption. Coloured in dark green;
- *Information* – Some module raises an event. Its message is intercepted and stored by the platform. This is a harmless event; thus, the execution continues with no interruption. Coloured in green;
- *Warning* – A module has detected a non-fatal error or situation that may cause a fatal error. This is a harmless event; thus, the execution continues with no interruption. Coloured in yellow;
- *Debug* – These events are raised when executing the simulation in "debug mode". Some modules optionally use this environment variable to show debugging information. Coloured in grey;
- *Error* – A fatal error has happened in the module execution and the module itself informs the platform about it, so the module has time to "graciously" close the execution. Another event that causes an error is that a module execution unexpectedly crashes, so this time the platform intercepts this error, informs the user and stops the execution. Coloured in red.
- *Exception* – This log type shows the error output stream of a module when the execution crashes. Typically, this kind of messages are produced when an un-controlled exception has occurred (Ex: error in bash script syntax). Coloured in orange. See figure below.



Log Messages

Date and time	Type	Message	Session identifier
2019-10-26T01:37:15.748	System	Session execution was unsuccessful due to an error	MatlabFail_test_session.20
2019-10-26T01:37:14.867	System	Module time :: 31ms	MatlabFail_test_session.20
2019-10-26T01:37:14.863	Warning	Output file/folder "output4" has not been successfully generated by the module	MatlabFail_test_session.20
2019-10-26T01:37:14.822	Warning	Output file/folder "output3" has not been successfully generated by the module	MatlabFail_test_session.20
2019-10-26T01:37:14.817	Warning	Output file/folder "output2" has not been successfully generated by the module	MatlabFail_test_session.20
2019-10-26T01:37:14.813	Warning	Output file/folder "output1" has not been successfully generated by the module	MatlabFail_test_session.20
2019-10-26T01:37:14.787	Error	Cannot run program "matlab" (in directory "/opt/openSF/openSF_3.8.2"): error=2	MatlabFail_test_session.20
2019-10-26T01:37:14.774	System	Executing Command :: matlab -nosplash -nodesktop -nodisplay -r addpath('/opt/c	MatlabFail_test_session.20
2019-10-26T01:37:14.762	System	Starting execution of module MatlabFailModel	MatlabFail_test_session.20
2019-10-26T01:37:14.740	System	Configuration file is valid.	MatlabFail_test_session.20
2019-10-26T01:37:14.702	System	Validating and copying needed files to session folder for model: MatlabFailModel	MatlabFail_test_session.20

Figure 4-73 Execution Log showing Exception message

In the previous screenshot it is also shown how openSF warns the user about output files that have not been created by the module: "*Output file/folder "…" has not been successfully generated by the module*", there is a log message for each output item not generated.

Pressing the "abort" button will make the system ask for confirmation. Once granted, the execution will be interrupted with an error event generated by the system. Later on, this simulation execution can be restarted or recovered from the last valid module executed.

4.9.5.1. Switch module version

In order to make more flexible the definition of a given simulation, this functionality allows selecting a specific version of a module for a simulation execution (as depicted in Figure 4-74 below).

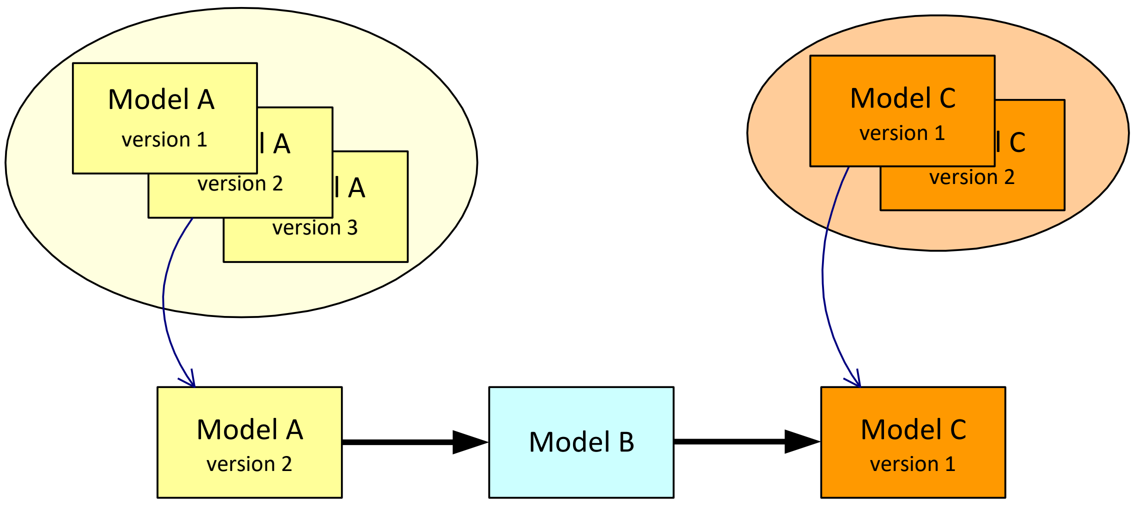


Figure 4-74 Module chain with different module versions

From the openSF HMI, the switch module version operation can be invoked from the simulation execution window by navigating down to the module that the user wishes to alter the version. Next, right-clicking over it whenever the module has more than one version available the "Switch module version" option appears for selection. This is illustrated in Figure 4-75 below.

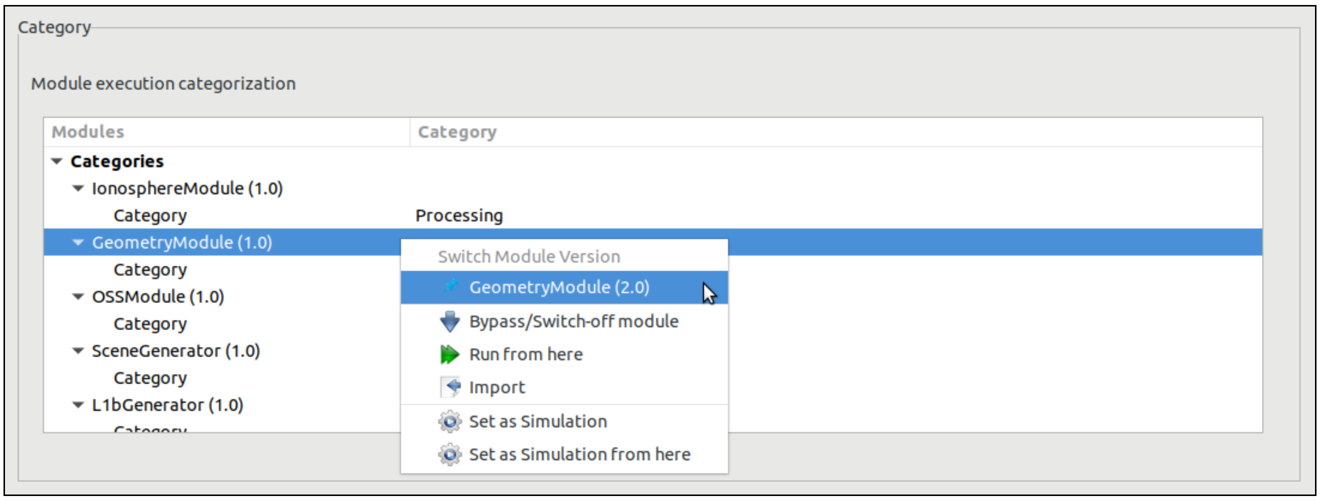


Figure 4-75: Switch module version

4.9.5.2. Bypass/Switch-off module execution

This functionality enables users to switch off certain modules when running simulations.

From the openSF HMI, the bypass/switch-off module operation can be invoked from the simulation execution window by navigating down to the module that the user wishes to bypass. Next, right-clicking over it the "Bypass/Switch-off module" option appears for selection, as illustrated in Figure 4-76 below. This choice is persistent, if the simulation is saved to database.

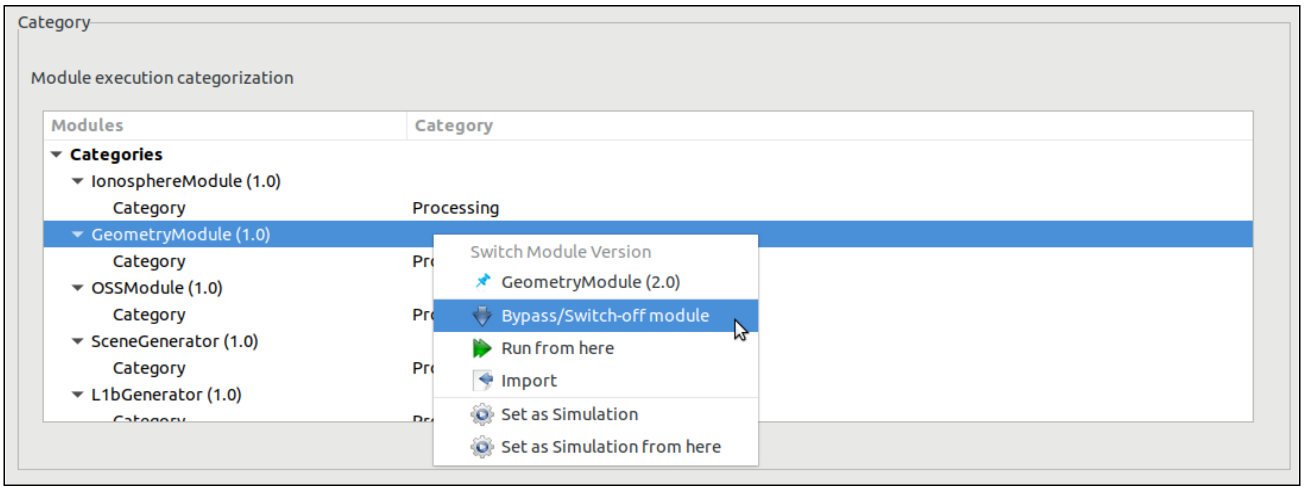
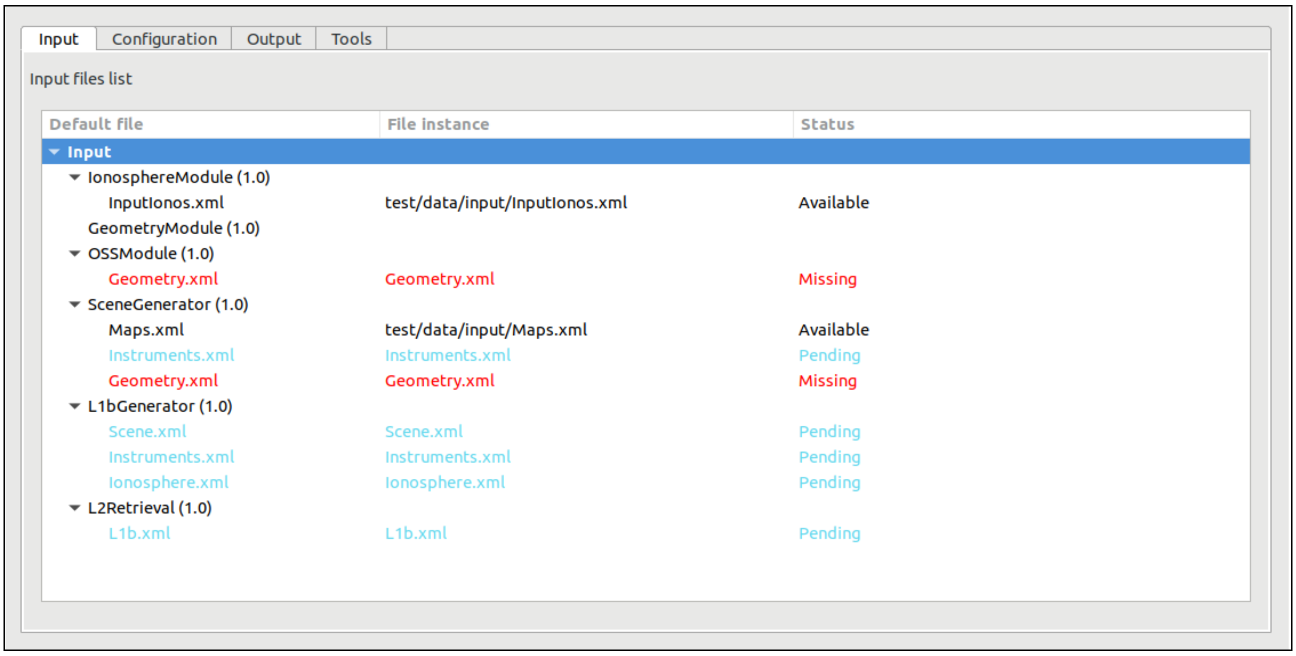


Figure 4-76: Bypass/Switch-off module

As a result, openSF will inform the user of the change in the list of data files needed to be provided due to the omission of modules and their corresponding outputs. Some inputs will no longer be needed, but other files may become "missing" instead of "pending" if a module that was going to generate them as its output becomes inactive.

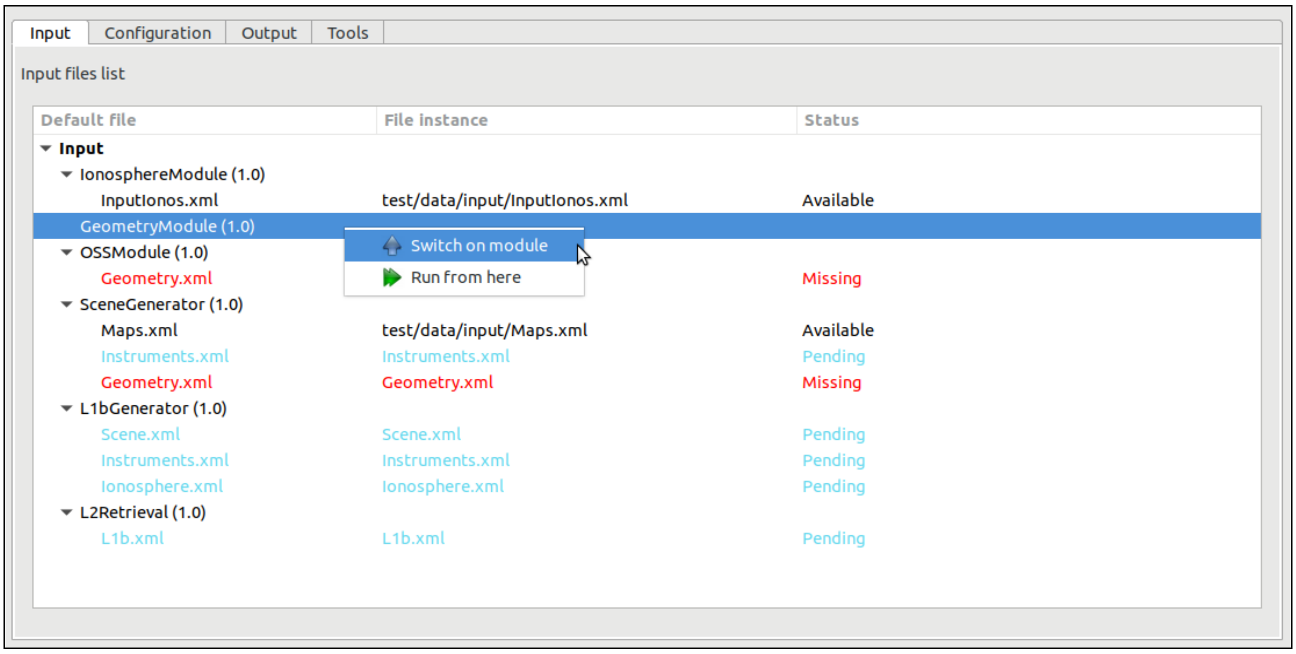


Input files list

Default file	File instance	Status
Input		
IonosphereModule (1.0)		
InputIonos.xml	test/data/input/InputIonos.xml	Available
GeometryModule (1.0)		
OSSModule (1.0)		
Geometry.xml	Geometry.xml	Missing
SceneGenerator (1.0)		
Maps.xml	test/data/input/Maps.xml	Available
Instruments.xml	Instruments.xml	Pending
Geometry.xml	Geometry.xml	Missing
L1bGenerator (1.0)		
Scene.xml	Scene.xml	Pending
Instruments.xml	Instruments.xml	Pending
Ionosphere.xml	Ionosphere.xml	Pending
L2Retrieval (1.0)		
L1b.xml	L1b.xml	Pending

Figure 4-77: Bypass/Switch-off module missing files

The user can revert the bypassed module and switch it back on by navigating down to the module that the user wishes to re-activate. Next, right-clicking over it the “Switch-on module” option appears for selection.



Input files list

Default file	File instance	Status
Input		
IonosphereModule (1.0)		
InputIonos.xml	test/data/input/InputIonos.xml	Available
GeometryModule (1.0)		
OSSModule (1.0)		
Geometry.xml		Missing
SceneGenerator (1.0)		
Maps.xml	test/data/input/Maps.xml	Available
Instruments.xml	Instruments.xml	Pending
Geometry.xml	Geometry.xml	Missing
L1bGenerator (1.0)		
Scene.xml	Scene.xml	Pending
Instruments.xml	Instruments.xml	Pending
Ionosphere.xml	Ionosphere.xml	Pending
L2Retrieval (1.0)		
L1b.xml	L1b.xml	Pending

Figure 4-78: Switch-on module

4.9.5.3. Run from a given point in the module chain

The idea is to allow users to skip modules at the beginning of the simulations, and therefore start simulations from a certain point. However, the data from non-executed modules is needed for the re-run. Before executing the simulation, the user needs to define the data files needed for the run.

The figure below shows a simple example. Modules A, B and C constitute the simulation. If we want to rerun it but starting from B, we need to provide the output of Module A, from a previous run.

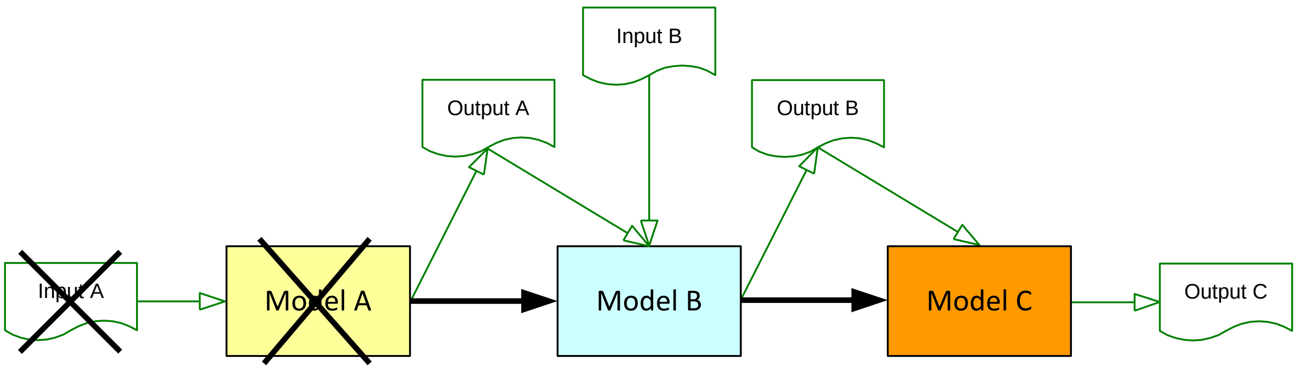


Figure 4-79 Run simulation from Module B

From the openSF HMI, the run from a given point capability can be invoked from the simulation execution window of a completed simulation by navigating down, either in the "Setup/Input" tab or in the "Execution/Category" one, to the module that the user wishes to start the execution from. Next, right-clicking over it the "Run from here" option appears for selection. This is illustrated in Figure 4-76 above.

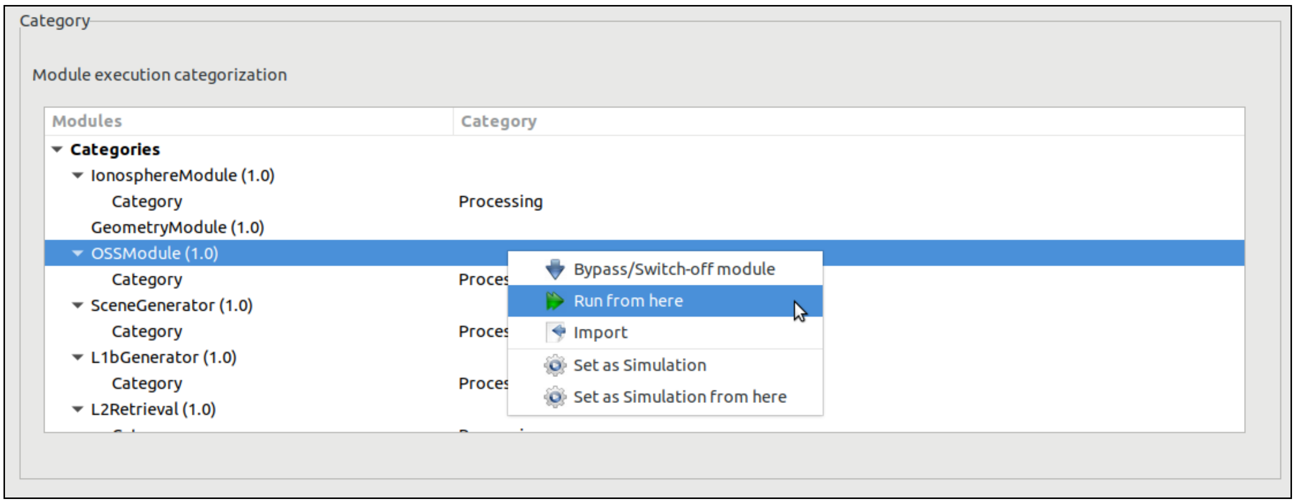


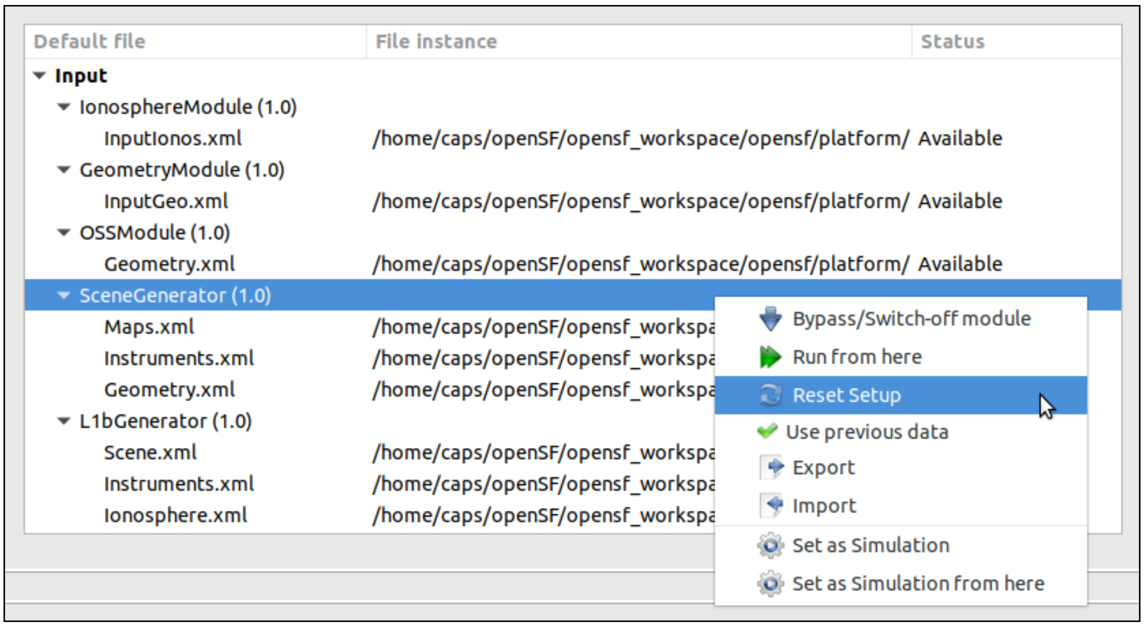
Figure 4-80: Run simulation from a given module

The result of this action is similar to doing option bypass/switch-off modules to each module previous to the point where the execution should start.

4.9.5.3.1. Run from a given module using previous data

The option to run a simulation from a given module is available both for a new simulation definition as well as for re-running a simulation execution. In this latter case the user can choose to use the input data previously used for the original simulation execution or use the input data produced in the simulation execution.

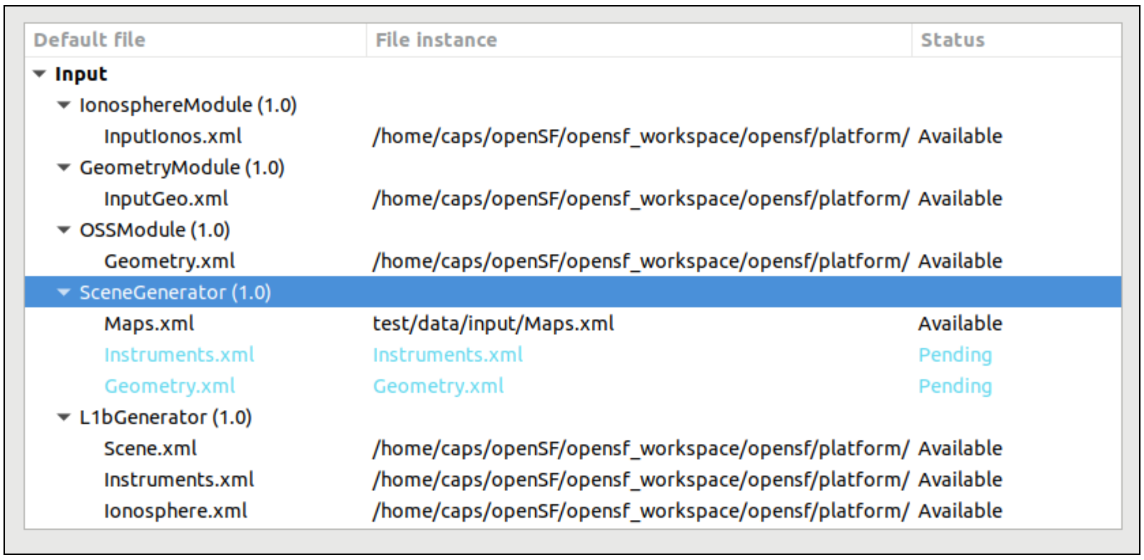
The user can revert the input and configuration data to the one originally used in the simulation definition by navigating down, either in the "Setup/Input" tab or in the "Execution/Category" one, to the module that the user wishes to reset the IO descriptors. Next, after right-clicking over it the "Reset setup" option appears in a context menu for selection.



Default file	File instance	Status
Input		
IonosphereModule (1.0)		
InputIonos.xml	/home/caps/openSF/opensf_workspace/opensf/platform/	Available
GeometryModule (1.0)		
InputGeo.xml	/home/caps/openSF/opensf_workspace/opensf/platform/	Available
OSSModule (1.0)		
Geometry.xml	/home/caps/openSF/opensf_workspace/opensf/platform/	Available
SceneGenerator (1.0)		
Maps.xml	/home/caps/openSF/opensf_workspa	
Instruments.xml	/home/caps/openSF/opensf_workspa	
Geometry.xml	/home/caps/openSF/opensf_workspa	
L1bGenerator (1.0)		
Scene.xml	/home/caps/openSF/opensf_workspa	
Instruments.xml	/home/caps/openSF/opensf_workspa	
Ionosphere.xml	/home/caps/openSF/opensf_workspa	

Figure 4-81: Reset IO descriptor option

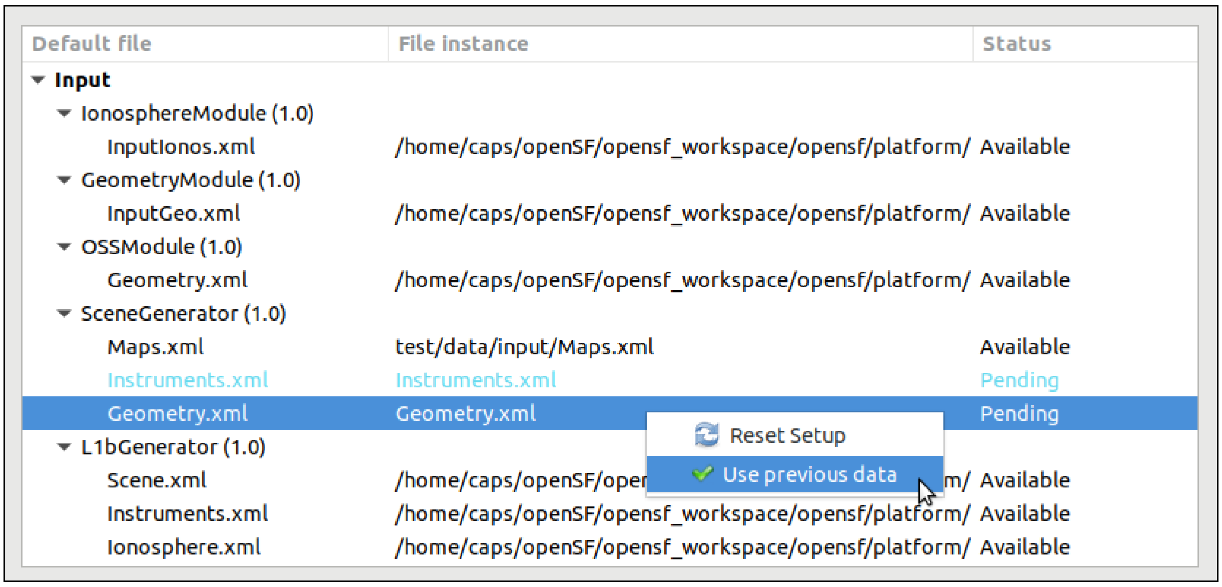
As a result, the simulation's original definitions are now re-established.



Default file	File instance	Status
Input		
IonosphereModule (1.0)		
InputIonos.xml	/home/caps/openSF/opensf_workspace/opensf/platform/	Available
GeometryModule (1.0)		
InputGeo.xml	/home/caps/openSF/opensf_workspace/opensf/platform/	Available
OSSModule (1.0)		
Geometry.xml	/home/caps/openSF/opensf_workspace/opensf/platform/	Available
SceneGenerator (1.0)		
Maps.xml	test/data/input/Maps.xml	Available
Instruments.xml	Instruments.xml	Pending
Geometry.xml	Geometry.xml	Pending
L1bGenerator (1.0)		
Scene.xml	/home/caps/openSF/opensf_workspace/opensf/platform/	Available
Instruments.xml	/home/caps/openSF/opensf_workspace/opensf/platform/	Available
Ionosphere.xml	/home/caps/openSF/opensf_workspace/opensf/platform/	Available

Figure 4-82: Reset IO descriptor setup

Furthermore, the user can revert to the data produced in the simulation execution for an individual IO descriptor by navigating down to the descriptor that the user wishes to restore the IO descriptors. Next, right-clicking over it the "Use previous data" option appears for selection.



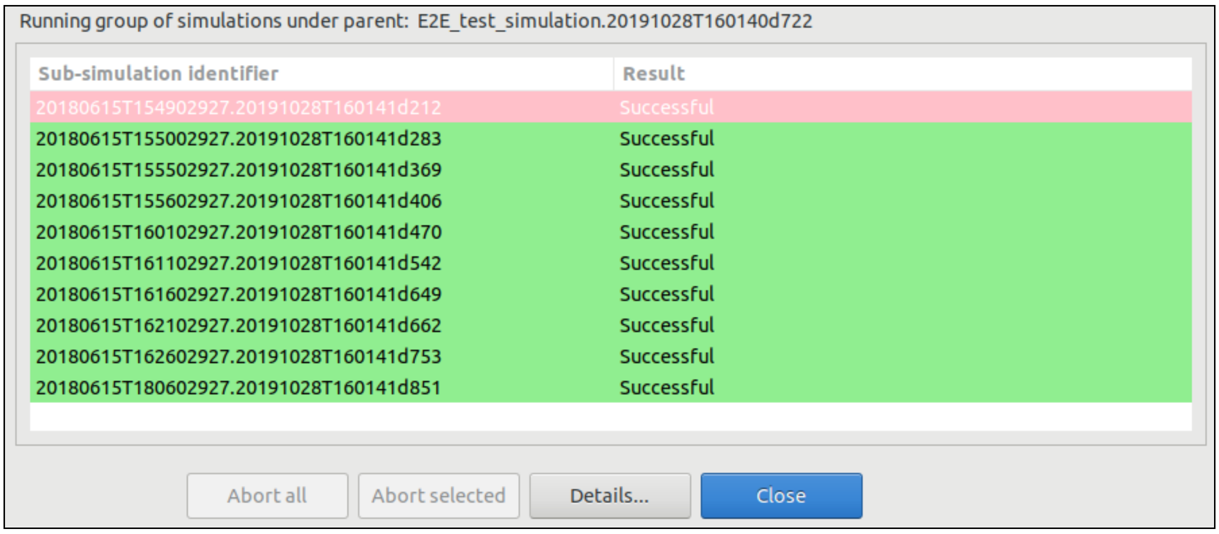
Default file	File instance	Status
Input		
IonosphereModule (1.0)		
InputIonos.xml	/home/caps/openSF/opensf_workspace/opensf/platform/	Available
GeometryModule (1.0)		
InputGeo.xml	/home/caps/openSF/opensf_workspace/opensf/platform/	Available
OSSModule (1.0)		
Geometry.xml	/home/caps/openSF/opensf_workspace/opensf/platform/	Available
SceneGenerator (1.0)		
Maps.xml	test/data/input/Maps.xml	Available
Instruments.xml	Instruments.xml	Pending
Geometry.xml	Geometry.xml	Pending
L1bGenerator (1.0)		
Scene.xml	/home/caps/openSF/oper…m/	Available
Instruments.xml	/home/caps/openSF/opensf_workspace/opensf/platform/	Available
Ionosphere.xml	/home/caps/openSF/opensf_workspace/opensf/platform/	Available

Figure 4-83: Use previous setup IO descriptor options

4.9.5.4. Grouping simulations - Time-Driven or Iteration/Perturbation

When simulations run in Time-driven or Iteration/Perturbation mode, a considerable number of children simulations can be opened at the same time. To tackle this issue, openSF groups all the child sub-simulations into a parent window, as well as in a parent output folder (section 4.11) to ease the user management of the overall simulation results.

Thus, if any simulation is executed following any of these modes the user will be prompted with a screen comprising all the sub-simulations and each of the execution results – depicted in Figure 4-84 for the Timeline execution and in Figure 4-85 for an execution with iterated or perturbed parameters.

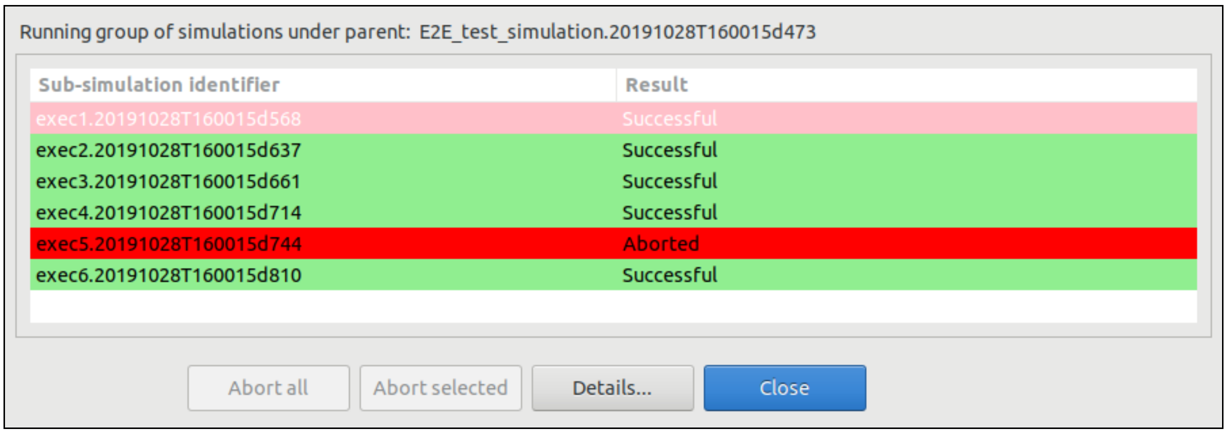


Running group of simulations under parent: E2E_test_simulation.20191028T160140d722

Sub-simulation identifier	Result
20180615T154902927.20191028T160141d212	Successful
20180615T155002927.20191028T160141d283	Successful
20180615T155502927.20191028T160141d369	Successful
20180615T155602927.20191028T160141d406	Successful
20180615T160102927.20191028T160141d470	Successful
20180615T161102927.20191028T160141d542	Successful
20180615T161602927.20191028T160141d649	Successful
20180615T162102927.20191028T160141d662	Successful
20180615T162602927.20191028T160141d753	Successful
20180615T180602927.20191028T160141d851	Successful

Figure 4-84 Grouping of simulations for the Time-Driven execution

While the simulations are executing, the user has several available options: “Abort all…” which aborts all the sub-simulations, “Abort selected” which aborts only the selected Sub-simulation or “Details…” if the user wants to open one subs-simulation in particular. The latest option can also be accessed if the user double-clicks on top of any simulation.


Running group of simulations under parent: E2E_test_simulation.20191028T160015d473

Sub-simulation identifier	Result
exec1.20191028T160015d568	Successful
exec2.20191028T160015d637	Successful
exec3.20191028T160015d661	Successful
exec4.20191028T160015d714	Successful
exec5.20191028T160015d744	Aborted
exec6.20191028T160015d810	Successful

Figure 4-85 Grouping of simulations for the Iteration/Perturbation execution

4.9.6. Parallelisation of module execution

Parallel execution of modules in openSF is based on Multicore programming for parallel computing.

A multi-core processor is a single computing component with two or more independent actual processors (called "cores"), which are the units that read and execute program instructions. The improvement in performance gained by the use of a multi-core processor depends very much on the software algorithms used and their implementation. In particular, possible gains are limited by the fraction of the software that can be parallelised to run on multiple cores simultaneously (Amdahl's law).

openSF target system is typically a computer or server. The availability of multicore processors makes it common that almost every target computer nowadays has two or more cores.

Regarding the parallel execution the approach is based on performing Parallelisation at module level: each module acquires a core resource thread and uses it, then releases it when finished. This approach is also generic enough to cover parallelisation at module chain level as well.

Parallelisation is implemented in openSF according to the principle that a module starts its execution as soon as its inputs are available. Therefore, the only modules that can run in parallel are those having external inputs (i.e. not generated by other modules).

4.9.6.1. Parallel execution

The parallel execution of simulation modules is activated and controlled by openSF based on the configured maximum number of distinct processes used for execution (see option Maximum Execution Threads in the preferences).

The process of scheduling the simulation modules execution can be decomposed in two distinct steps:

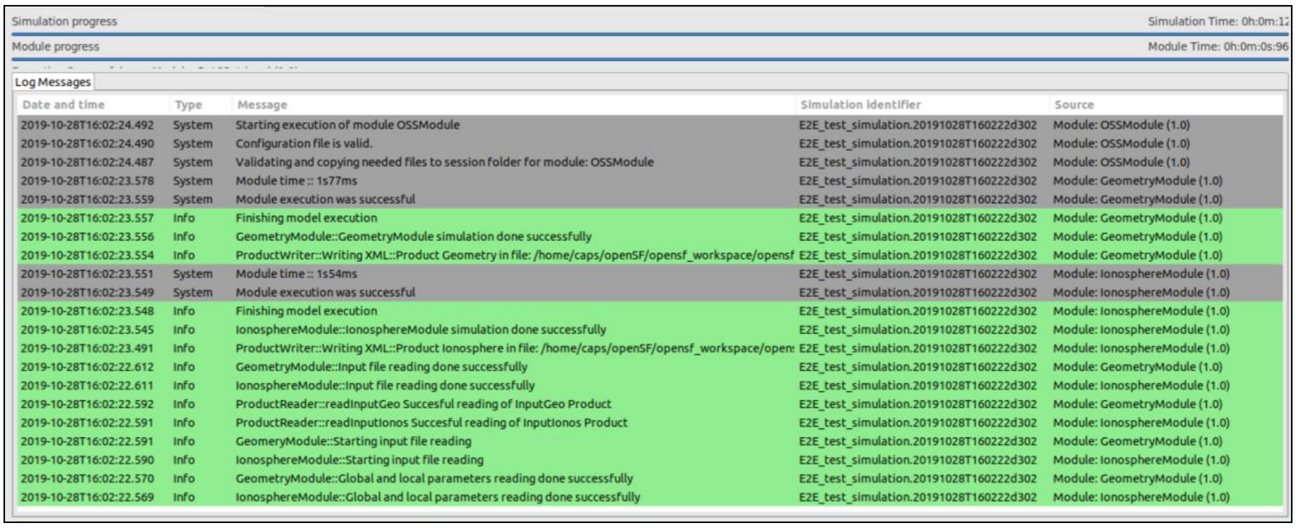
1. Taking into account the input/output dependencies between all simulation modules (as specified via the descriptors), openSF calculates a serializable execution schedule of all modules
2. Then, at the start of the execution and whenever a simulation module terminates, OpenSF launches in parallel as many modules as possible, considering that:

a. a module can only be launched if all modules providing its inputs have already completed their execution

b. the number of running processes must always be less or equal to the maximum number of processes allowed

It is important to notice that currently OpenSF relies only on process termination to determine that a simulation module has completed, without any other additional check – i.e. openSF does not check the return code, nor verifies if the intended outputs have been generated.

The parallelisation of the simulation modules is essentially transparent to the user, i.e., the parallelisation is performed without requiring feedback from the user, both at module chain level and module level. Figure 4-86 show an example of a simulation execution were two modules are executed in parallel as can be seen by log messages originating from the two modules interleaved with each other.



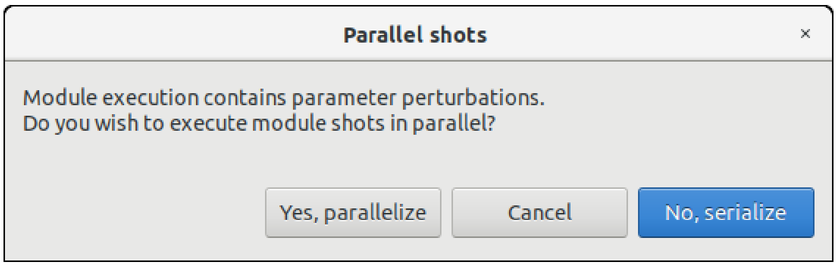
Date and time	Type	Message	Simulation identifier	Source
2019-10-28T16:02:24.492	System	Starting execution of module OSSModule	E2E_test_simulation.20191028T160222d302	Module: OSSModule (1.0)
2019-10-28T16:02:24.490	System	Configuration file is valid.	E2E_test_simulation.20191028T160222d302	Module: OSSModule (1.0)
2019-10-28T16:02:24.487	System	Validating and copying needed files to session folder for module: OSSModule	E2E_test_simulation.20191028T160222d302	Module: OSSModule (1.0)
2019-10-28T16:02:23.578	System	Module time :: 1s77ms	E2E_test_simulation.20191028T160222d302	Module: GeometryModule (1.0)
2019-10-28T16:02:23.559	System	Module execution was successful	E2E_test_simulation.20191028T160222d302	Module: GeometryModule (1.0)
2019-10-28T16:02:23.557	Info	Finishing model execution	E2E_test_simulation.20191028T160222d302	Module: GeometryModule (1.0)
2019-10-28T16:02:23.556	Info	GeometryModule::GeometryModule simulation done successfully	E2E_test_simulation.20191028T160222d302	Module: GeometryModule (1.0)
2019-10-28T16:02:23.554	Info	ProductWriter::Writing XML::Product Geometry in file: /home/caps/openSF/opensf_workspace/opensf	E2E_test_simulation.20191028T160222d302	Module: GeometryModule (1.0)
2019-10-28T16:02:23.551	System	Module time :: 1s54ms	E2E_test_simulation.20191028T160222d302	Module: IonosphereModule (1.0)
2019-10-28T16:02:23.549	System	Module execution was successful	E2E_test_simulation.20191028T160222d302	Module: IonosphereModule (1.0)
2019-10-28T16:02:23.548	Info	Finishing model execution	E2E_test_simulation.20191028T160222d302	Module: IonosphereModule (1.0)
2019-10-28T16:02:23.545	Info	IonosphereModule::IonosphereModule simulation done successfully	E2E_test_simulation.20191028T160222d302	Module: IonosphereModule (1.0)
2019-10-28T16:02:23.491	Info	ProductWriter::Writing XML::Product Ionosphere in file: /home/caps/openSF/opensf_workspace/open:	E2E_test_simulation.20191028T160222d302	Module: IonosphereModule (1.0)
2019-10-28T16:02:22.612	Info	GeometryModule::Input file reading done successfully	E2E_test_simulation.20191028T160222d302	Module: GeometryModule (1.0)
2019-10-28T16:02:22.611	Info	IonosphereModule::Input file reading done successfully	E2E_test_simulation.20191028T160222d302	Module: IonosphereModule (1.0)
2019-10-28T16:02:22.592	Info	ProductReader::readInputGeo Succesful reading of InputGeo Product	E2E_test_simulation.20191028T160222d302	Module: GeometryModule (1.0)
2019-10-28T16:02:22.591	Info	ProductReader::readInputIonos Succesful reading of InputIonos Product	E2E_test_simulation.20191028T160222d302	Module: IonosphereModule (1.0)
2019-10-28T16:02:22.591	Info	GeomeryModule::Starting input file reading	E2E_test_simulation.20191028T160222d302	Module: GeometryModule (1.0)
2019-10-28T16:02:22.590	Info	IonosphereModule::Starting input file reading	E2E_test_simulation.20191028T160222d302	Module: IonosphereModule (1.0)
2019-10-28T16:02:22.570	Info	GeometryModule::Global and local parameters reading done successfully	E2E_test_simulation.20191028T160222d302	Module: GeometryModule (1.0)
2019-10-28T16:02:22.569	Info	IonosphereModule::Global and local parameters reading done successfully	E2E_test_simulation.20191028T160222d302	Module: IonosphereModule (1.0)

Figure 4-86 Simulation execution showing Parallel module execution

In case two modules are executing in parallel the log messages are shown in the order of arrival to openSF. They will appear mixed in the simulation log. By looking at the Source column the user can identify which module produced each message. Nevertheless, the user can access the "*Execution/Logs*" option and select/filter whichever log messages according to given criteria. Notice that the writing accesses to the log file itself is protected for concurrency issues.

The choice on whether to parallelise module execution is based on the simulation module's IO descriptors dependencies. Only modules without such dependencies are considered for parallel execution. This means that during the simulation execution the consistency of the data flow constituents is granted.

In case parallelisation is active for a simulation execution with parameter perturbation a choice is given to the user whether parameter perturbation can be parallelisable or if it should be serialized.


Parallel shots

Module execution contains parameter perturbations.
Do you wish to execute module shots in parallel?

Figure 4-87 Simulation execution showing Parallel module execution

4.9.6.2. Precautions for module developers to ensure safe module parallelisation

The evolution of openSF to allow parallel execution brings also added responsibility to module developers. **It should be clarified that module developers must ensure that the modules/algorithms developed are in fact parallelisable, e.g., that the implementation has the proper precautions regarding access to common resources**. openSF can only go so far in assuring synchronization of module execution and must rely on modules being "well behaved" with respect to parallel execution.

In order to ensure safe module parallelisation module developers should ensure that modules are either:

- Thread safe: implementation is guaranteed to be free of race conditions when accessed by multiple threads simultaneously, or;
- Conditionally safe: different threads can access different objects simultaneously, and access to shared data is protected from race conditions.

The use of software libraries can provide certain thread-safety guarantees. For example, concurrent reads are typically guaranteed to be thread-safe, but concurrent writes might not be. Whether or not a program using such a library is thread-safe depends on whether it uses the library in a manner consistent with those guarantees. Thread safety guarantees imply some design steps to prevent or limit the risk of different forms of deadlocks, as well as optimizations to maximize concurrent performance.

There are several approaches for avoiding race conditions to achieve thread safety. The first class of approaches focuses on avoiding shared state, and includes:

- Re-entrancy: writing code in such a way that it can be partially executed by a thread re-executed by the same thread or simultaneously executed by another thread and still correctly completes the original execution. This requires the saving of state information in variables local to each execution, usually on a stack, instead of in static or global variables or other non-local state. All non-local state must be accessed through atomic operations and the data-structures must also be re-entrant;
- Thread-local storage: variables are localized so that each thread has its own private copy. These variables retain their values across subroutine and other code boundaries, and are thread-safe since they are local to each thread, even though the code which accesses them might be executed simultaneously by another thread.

The second class of approaches are synchronization-related, and are used in situations where shared state cannot be avoided:

- Mutual exclusion: access to shared data is serialized using mechanisms (e.g. semaphores) that ensure only one thread reads or writes to the shared data at any time. Incorporation of mutual exclusion needs to be well thought out, since improper usage can lead to side-effects like deadlocks and resource starvation;

- Atomic operations: shared data are accessed by using atomic operations which cannot be interrupted by other threads. This usually requires using special machine language instructions, which might be available in a runtime library. Since the operations are atomic, the shared data are always kept in a valid state, no matter how other threads access it. Atomic operations form the basis of many thread locking mechanisms, and are used to implement mutual exclusion primitives;
- Immutable objects: the state of an object cannot be changed after construction. This implies that only read-only data is shared and inherent thread safety. Mutable (non-const) operations can then be implemented in such a way that they create new objects instead of modifying existing ones (e.g. this approach is used by the string implementations in Java, C# and python).

Thread safety

Thread safety is a simple concept: is it "safe" to perform operation A on one thread whilst another thread is performing operation B, which may or may not be the same as operation A. This can be extended to cover many threads. In this context, "safe" means:

- No undefined behaviour;
- All invariants of the data structures are guaranteed to be observed by the threads.

The actual operations A and B are important. If two threads both read a plain int variable, then this is fine. However, if any thread may write to that variable, and there is no synchronization to ensure that the read and write cannot happen together, then a data race occurs, which is undefined behaviour, and this is not thread safe.

Unless special precautions are taken, then it is not safe to have one thread read from a structure at the same time as another thread writes to it. If it can be guaranteed that the threads cannot access the data structure at the same time (through some form of synchronization such as a mutex, critical section, semaphore or event) then there should be no problem.

Element like mutexes and critical sections can be used to prevent concurrent access to some data, so that the writing thread is the only thread accessing the data when it is writing, and the reading thread is the only thread accessing the data when it is reading, thus providing the thread safety guarantee. This therefore avoids the undefined behaviour mentioned above.

However, the programmer still needs to ensure that the code is safe in the wider context: if more than one variable needs to be modified then the mutex needs to be held across the whole operation rather than for each individual access, otherwise the invariants of the data structure may not be observed by other threads.

It is also possible that a data structure may be thread safe for some operations but not others. For example, a single-producer single-consumer queue will be fine if one thread is pushing items on the queue and another is popping items off the queue, but will break if two threads are pushing items, or two threads are popping items.

Global variables are implicitly shared between all threads, and therefore all accesses must be protected by some form of synchronization (such as a mutex) if any thread can modify them. On the other hand, if a separate copy of the data is held for each thread, then that thread can modify its copy without worrying about concurrent access from any other thread, and no synchronization is required. Of course, synchronization is always needed if two or more threads are going to operate on the same data.

© DEIMOS Space S.L.U., 2019

4.9.7. Remote execution

When execution a simulation the user can select a remote machine in which the execution is to take place. This configuration can be applied:

(a) to the whole simulation: selecting a remote machine (previously configured) in the 'Remote Machine' drop down list box of the simulation execution window;

(b) on a module by module case: selecting for each module a remote machine from the contextual menu obtained when right-clicking over the module listed in the simulation setup pane of the simulation execution window.

The configuration assumed is the last one performed: selecting a machine for the whole simulation overrides previously configured machines per module; as configuring a remote machine module by module after configuring the whole simulation allows a finer configuration.

4.9.8. Simulation script generation

This functionality is provided to create and save a file script to enable the external execution of the simulation. This script file, called "<*simulation_name*>.sh", will be saved in the simulation folder as every needed input and configuration files. This script shall contain all the environment variables definitions and calls for modules' executions. The script can be executed from command line and it requires no parameters.

Note that the script is always generated when executing a simulation. Hence this functionality of Simulation Script Generation is made available to generate the script without having to execute the simulation for it to be generated.

It is important to recall that while the execution will be mimicked, executing this script outside the openSF system, the error handling and results storage capabilities will be lost. Moreover, the simulation is executed without resorting to the parallelisation capabilities provided by openSF.

4.9.9. Simulation Resuming

Once it is possible to interrupt a simulation execution with the breakpoint scheduling system it is possible to resume the paused simulation and continue with the simulation chain keeping the same settings as the previous run.

Users can also resume aborted or failed simulations taking into account that the simulation chain will start in the last successfully run module.

4.9.10. Import and Export simulations

The import/export capability provides the means to share all the information associated to simulations among different openSF instances. Thus, the data needed to import a simulation consists in two files, obtained through a previously executed export operation:

- ❑ SQL file, containing the SQL operations to perform a replica of the database data into the target openSF instance.
- ❑ ZIP file, containing the data files needed for the execution of the simulation. Furthermore, in case of an executed simulation, the zip file includes also the input files used for that particular run.

4.9.10.1. Export simulation

From the openSF HMI, the export operation can be invoked from two different locations:

1. From the **Repository** menu. In this case, we need to navigate from the Repository menu down to the simulation that the user wishes to export. Next, right-clicking over it; the "Export" option appears for selection. This is illustrated in Figure 4-88 below.

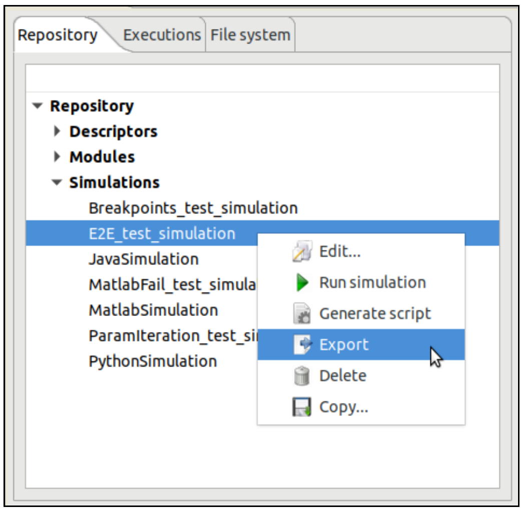


Figure 4-88: Export from the Repository menu

2. From the **Executions** menu. If the user wishes to export a simulation that has been run, he/she can do so by accessing the Executions menu and selecting the Export operation upon the desired simulation, similarly to the previous case. This is shown in the next figure.

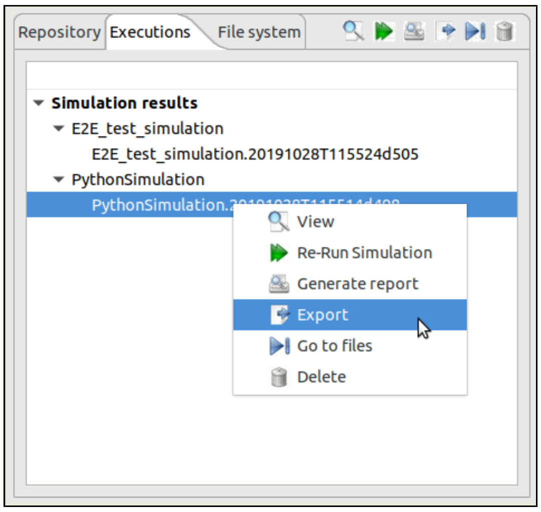
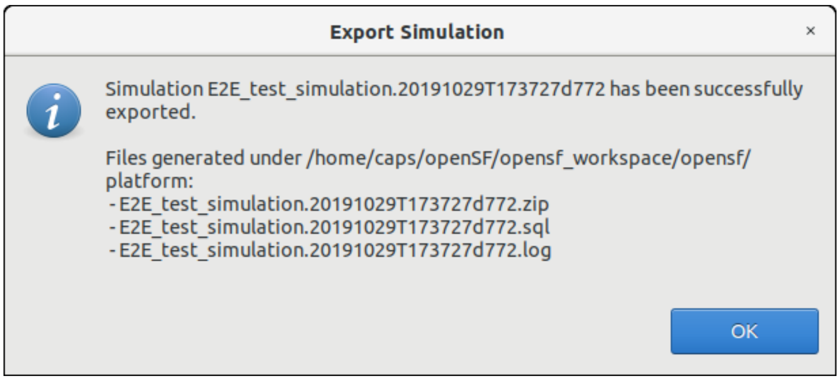


Figure 4-89: Export from the Executions menu

In both cases, the output obtained as a result of the Export operation are three files (sql, log and zip) that are placed in the folder indicated by the value of the OPENSF_HOME environment variable. These files are needed for the import operation, and they ensure the creation in the database of the constituent elements of the simulation (i.e. descriptors, modules, tools, and simulation, as well as the provision of input and configuration files) needed for the simulation's execution. However, it is to be noticed that the executable files corresponding to the modules and tools are not included in the export operation.

The difference between both types of export (from the Repository menu and from the Executions menu) is that the Import of the latter type creates the simulation only in the Executions tab. That is due to its identifier (featuring the execution time stamp) and its state (Successful or Failed), which indicates that it is an executed simulation.

Once the export has been carried out, openSF reports the status in a dialog as the one shown below.



Simulation E2E_test_simulation.20191029T173727d772 has been successfully exported.

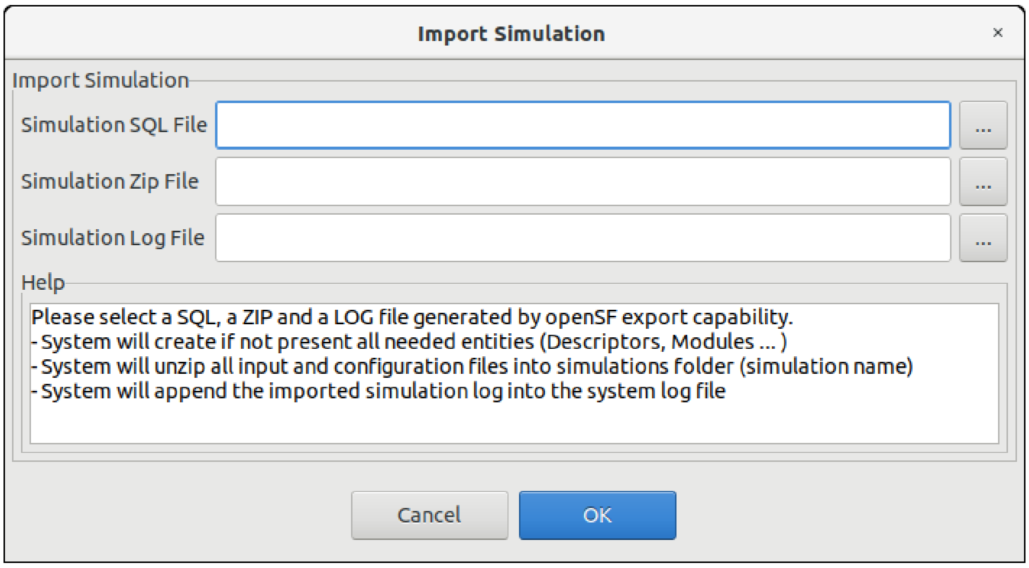
Files generated under /home/caps/openSF/opensf_workspace/opensf/platform:
- E2E_test_simulation.20191029T173727d772.zip
- E2E_test_simulation.20191029T173727d772.sql
- E2E_test_simulation.20191029T173727d772.log

Figure 4-90: Successful execution of the export

4.9.10.2. Import simulation

Users can activate the import operation by accessing the Repository menu and clicking on the "Import simulation …" option, showing the dialog of Figure 4-91. The dialog requests the files needed to complete the operation: the SQL file that includes the statements needed to create the simulation in the target database, the log file and a ZIP archive with the simulation data files (configuration and optionally, inputs) for the simulation's execution.

Note that for simulations exported by openSF v3.7.1 onwards, the ZIP archive includes the simulation logs, so it is not necessary to provide the log file on import. If one is specified, it will be ignored.



Please select a SQL, a ZIP and a LOG file generated by openSF export capability.
- System will create if not present all needed entities (Descriptors, Modules ...)
- System will unzip all input and configuration files into simulations folder (simulation name)
- System will append the imported simulation log into the system log file

Figure 4-91: Inputs requested for the import

The user can navigate through the file system in order to access the files by clicking on the buttons appearing at the right side of each input field. Upon completion openSF shall report on the status of the operation. The figure below shows the dialogue that is presented in case the import was performed successfully.

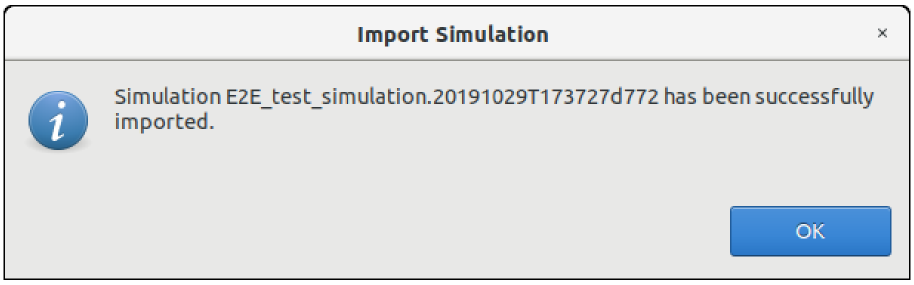

Import Simulation
Simulation E2E_test_simulation.20191029T173727d772 has been successfully imported.
OK

Figure 4-92: Successful execution of the import

4.9.10.3. Export module of a simulation

This capability deals with the possibility of exporting the data associated to a module that has already taken place in a simulation. The export functionality exports the data related to only one module of a given simulation. Thus, the data exported is comprised by the module configuration and input files.

From the openSF HMI, the export module operation can be invoked from the simulation execution window by navigating down to the module that the user wishes to export. Next, right-clicking over it; the "Export" option appears for selection. This is illustrated in Figure 4-93 below.

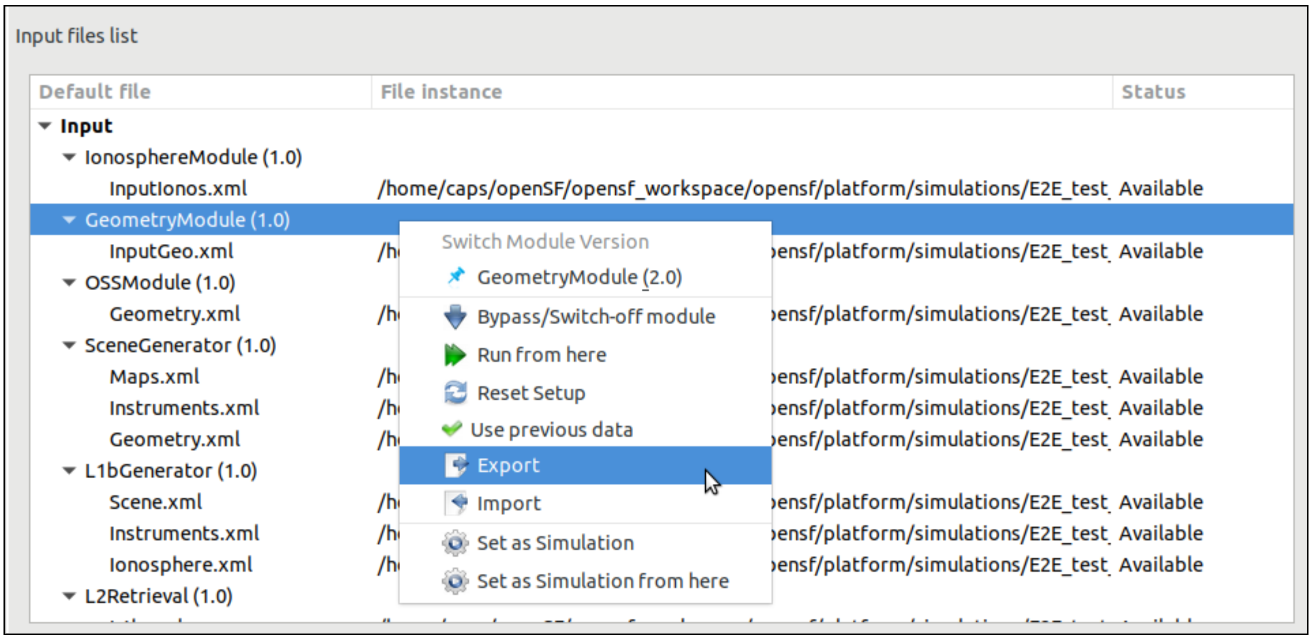
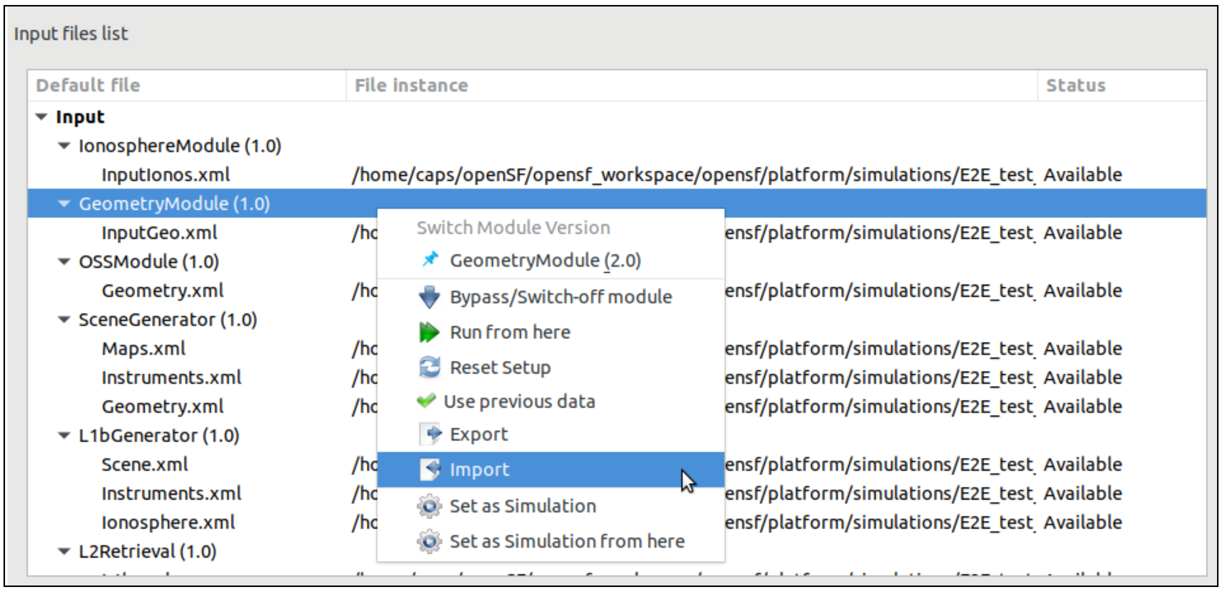


Figure 4-93: Export module from the Simulation Result view

The output obtained as a result of the Export operation is a zip file placed in the folder indicated by the value of the OPENSF_HOME environment variable. This file is needed for the import operation providing the input and configuration files needed for the simulation's execution. However, it is to be noticed that the executable files corresponding to the modules and tools are not included in the export.

4.9.10.4. Import module of a simulation

From the openSF HMI, the import module operation can be invoked from the simulation edition window by navigating down to the module that the user wishes to import to. Next, right-clicking over it; the "Import" option appears for selection. This is illustrated in Figure 4-94 below.



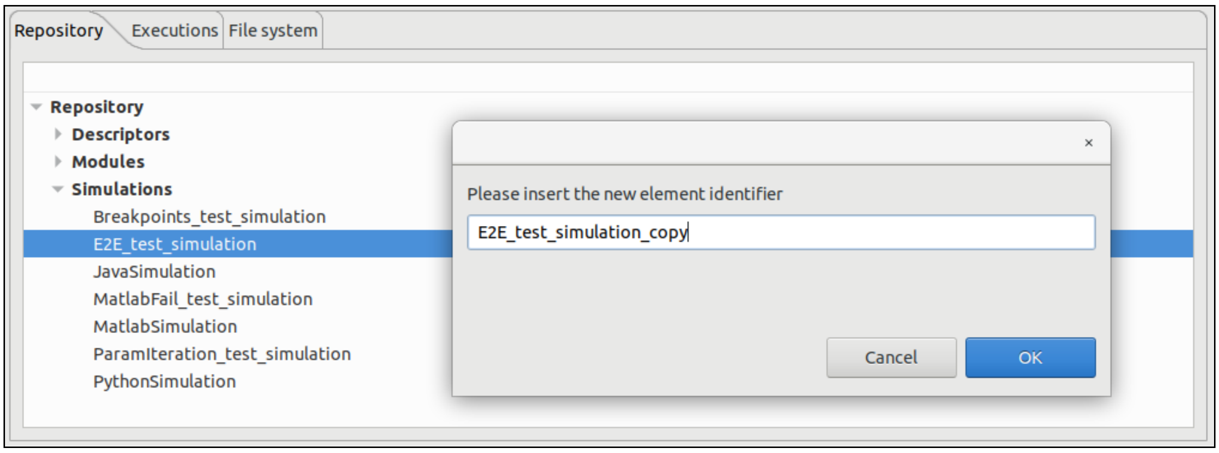
Default file	File instance	Status
Input		
IonosphereModule (1.0)		
InputIonos.xml	/home/caps/openSF/opensf_workspace/opensf/platform/simulations/E2E_test	Available
GeometryModule (1.0)		
InputGeo.xml	/hc ensf/platform/simulations/E2E_test	Available
OSSModule (1.0)		
Geometry.xml	/hc ensf/platform/simulations/E2E_test	Available
SceneGenerator (1.0)		
Maps.xml	/hc ensf/platform/simulations/E2E_test	Available
Instruments.xml	/hc ensf/platform/simulations/E2E_test	Available
Geometry.xml	/hc ensf/platform/simulations/E2E_test	Available
L1bGenerator (1.0)		
Scene.xml	/hc ensf/platform/simulations/E2E_test	Available
Instruments.xml	/hc ensf/platform/simulations/E2E_test	Available
Ionosphere.xml	/hc ensf/platform/simulations/E2E_test	Available
L2Retrieval (1.0)		

Figure 4-94: Import module from the Simulation edition view

A module data can be imported into an openSF instance from the data obtained from the export operation. As the contents of the export relate to data files, it is required that the module exists in the target openSF instance.

4.9.11. Copy simulation

Users can select a certain simulation and choose the option to copy it. The user needs to specify a new name for the simulation, unique with respect to the existing simulations. All simulation definitions shall be copied to a new simulation instance.



Please insert the new element identifier

E2E_test_simulation_copy

Figure 4-95: Simulation. Copy Simulation

4.10. Executions

As mentioned in section 4.6, openSF can be divided in two logical parts: repository and executions.

The third of these parts, named executions, represents the dynamic view of the system. Here the executed simulations are stored with their input and output data. Users can consult their results and log messages generated, as well as re-run simulations as needed.

The list of all the executed simulations stored in the database can be through the executions view of the side bar or via the “Executions / Manage” menu (Figure 4-98) from the main menu.

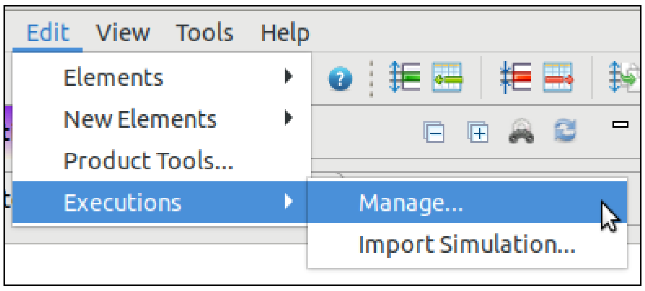


Figure 4-96: Results menu

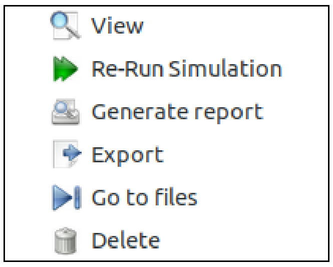
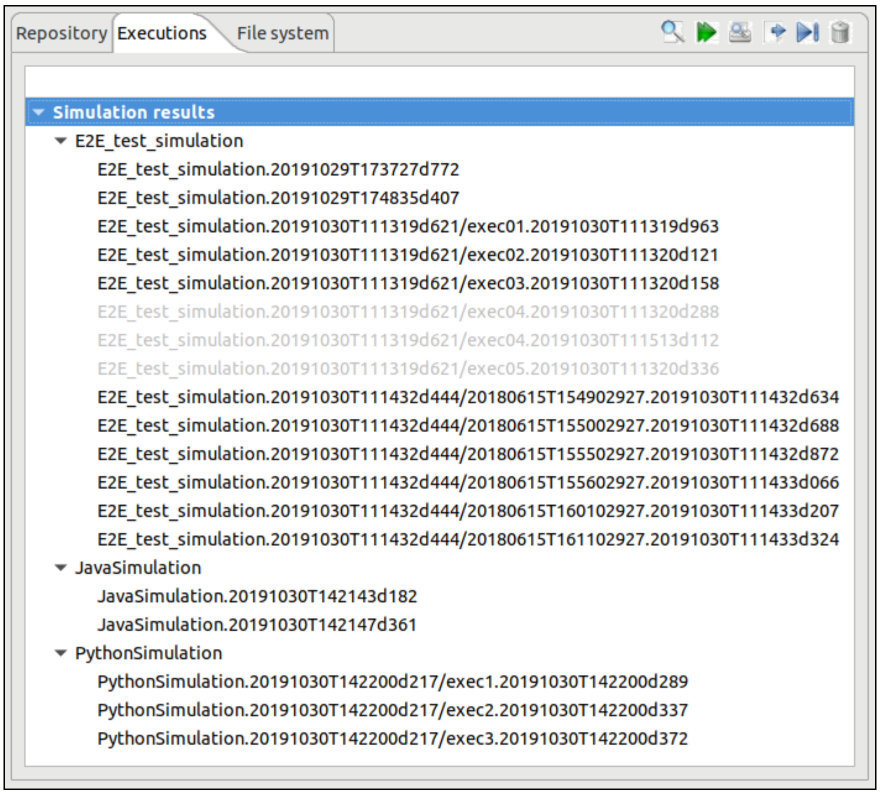


Figure 4-97: Results pop-up menu

The Executions tab of the side view uses a colour code to provide information about the differences between the connected database and the File System. If a result is present in the system, the execution is displayed in black, whereas if the result is present in the database but not in the system the item is displayed in light grey.

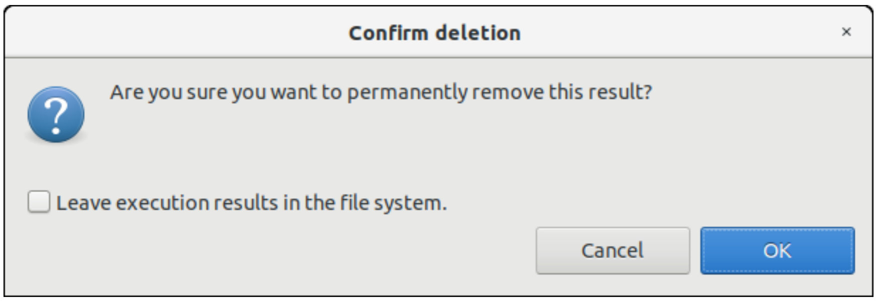


```
Simulation results
  E2E_test_simulation
    E2E_test_simulation.20191029T173727d772
    E2E_test_simulation.20191029T174835d407
    E2E_test_simulation.20191030T111319d621/exec01.20191030T111319d963
    E2E_test_simulation.20191030T111319d621/exec02.20191030T111320d121
    E2E_test_simulation.20191030T111319d621/exec03.20191030T111320d158
    E2E_test_simulation.20191030T111319d621/exec04.20191030T111320d288
    E2E_test_simulation.20191030T111319d621/exec04.20191030T111513d112
    E2E_test_simulation.20191030T111319d621/exec05.20191030T111320d336
    E2E_test_simulation.20191030T111432d444/20180615T154902927.20191030T111432d634
    E2E_test_simulation.20191030T111432d444/20180615T155002927.20191030T111432d688
    E2E_test_simulation.20191030T111432d444/20180615T155502927.20191030T111432d872
    E2E_test_simulation.20191030T111432d444/20180615T155602927.20191030T111433d066
    E2E_test_simulation.20191030T111432d444/20180615T160102927.20191030T111433d207
    E2E_test_simulation.20191030T111432d444/20180615T161102927.20191030T111433d324
  JavaSimulation
    JavaSimulation.20191030T142143d182
    JavaSimulation.20191030T142147d361
  PythonSimulation
    PythonSimulation.20191030T142200d217/exec1.20191030T142200d289
    PythonSimulation.20191030T142200d217/exec2.20191030T142200d337
    PythonSimulation.20191030T142200d217/exec3.20191030T142200d372
```

Figure 4-98: Executions view in the side bar

For each of the results of the Executions tree, a number of operations can be performed. These operations can be performed for a single result or for multiple ones simultaneously, by selecting more than one result with the Ctrl or Shift keys. Operations involving results include the following:

- *List* – present the list of existing execution results;
- *View* – consult the data of an existing execution result;
- *Re-run* – starts a new simulation execution. This new simulation is a replica of the former, but shall create a new simulation folder.
- *Report generation* – shows a text report describing the execution.
- *Exportation* - exports the entire execution definition;
- *Go to files* – show the files of the selected execution in the File System tree of the side bar.
- *Deletion* – delete an execution result from the database. A dialog is prompt to the user to confirm the execution deletion from the database and, if desired, also from the file system.


Confirm deletion

Are you sure you want to permanently remove this result?

☐ Leave execution results in the file system.

Figure 4-99: Confirmation dialog to delete execution(s) from database and file system

4.11. Results

openSF approach for naming simulation execution directories, as well as their relative supporting files, involves the use of names with a timestamp. The use of a timestamp is meant to ensure a unique identification of the simulation folder and files.

In case of a nominal simulation (single execution) the execution result (whether it was successful or not) is stored in a <simulations> folder, named as "<simulation_id>.<starting_time>". Starting time is coded as "YYYYMMDDTHHmmSSdsss"[10] in local time – see the format in Figure 4-100 and Figure 4-102.

In case of timeline-based and iteration/perturbation-based simulations, openSF groups related simulations execution results into separate output folders (since these two types of executions can end up to hundreds of simulations). Also, in these two cases, in order to uniquely identify the executions, a timestamp is appended to the execution identifier, as shown in Figure 4-100.

Note that the prefix "exec#" is used for each iteration folder as a way of facilitating the user's distinction between Timeline and Iteration/Perturbation executions.

Summarising, the naming scheme for simulation result folders is as follows:

- Nominal simulation (single execution)

[10] "sss" denotes milliseconds

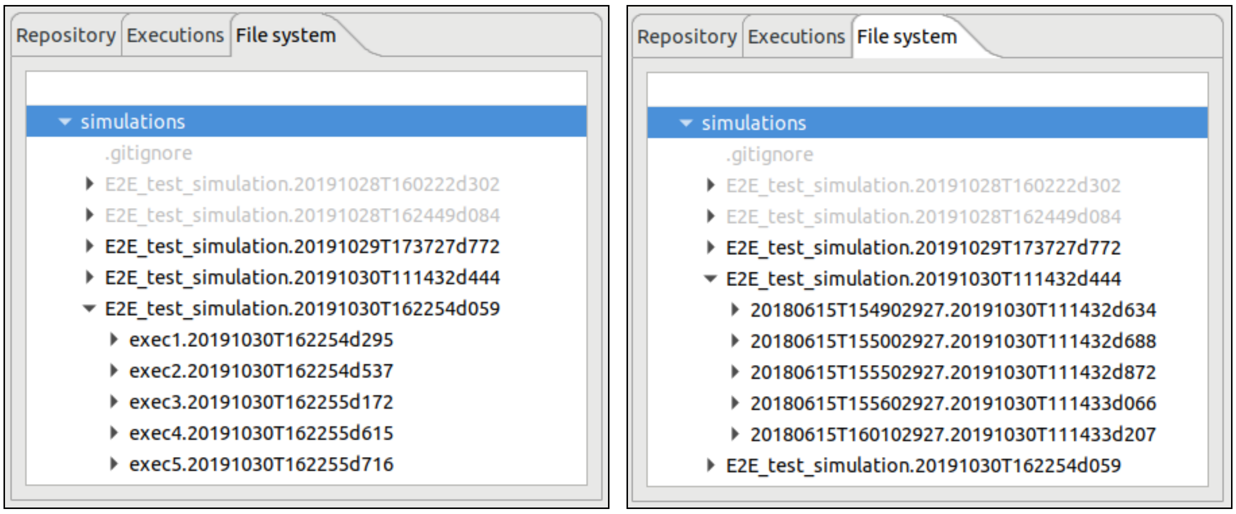
- ○ <*simulation_id*>.<*start_time*>, where
    - ▪ *simulation_id* is the simulation name (as seen in the Repository view)
    - ▪ *start_time* is the simulation execution start time, formatted as YYYYMMDDTHHmmSSdsss

❑ Iteration/perturbation-based simulation (multiple executions, based on the combination of iterations/perturbations)

- ○ <*simulation_id*><*start_time*>/exec#.<*start_time2*>, where
    - ▪ *simulation_id* is the parent simulation name (as seen in the Repository view)
    - ▪ *start_time* is the simulation execution start time, formatted as YYYYMMDDTHHmmSSdsss
    - ▪ *#* is an incremental integer defining the order of execution of the multiple executions
    - ▪ *start_time2* is the iterated execution start time, formatted as YYYYMMDDTHHmmSSdsss

❑ Timeline-based simulation (multiple executions, based on time segments specified)

- ○ <*simulation_id*><*start_time*>/<*time_segment_start*>.<*start_time2*>
    - ▪ *simulation_id* is the parent simulation name (as seen in the Repository view)
    - ▪ *start_time* is the simulation execution start time, formatted as YYYYMMDDTHHmmSSdsss
    - ▪ *time_segment_start* is the time segment start time (as defined by the user)
    - ▪ *start_time2* is the time segment execution start time, formatted as YYYYMMDDTHHmmSSdsss

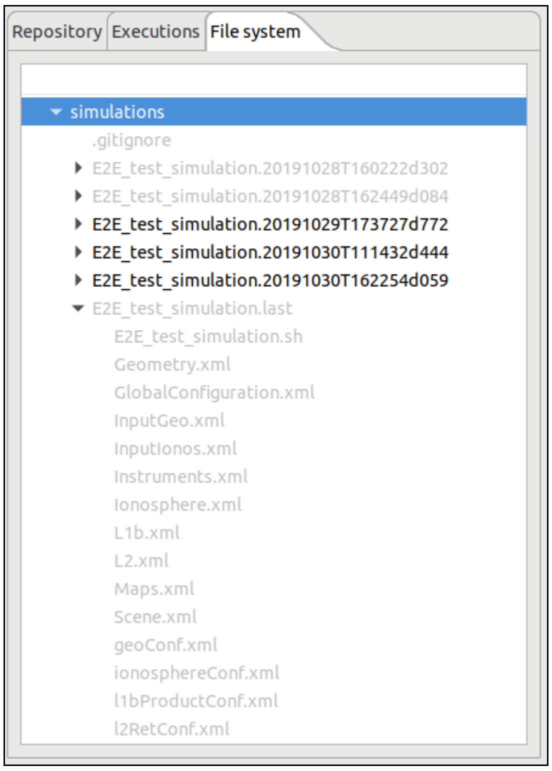


```
▾ simulations
    .gitignore
  ▸ E2E_test_simulation.20191028T160222d302
  ▸ E2E_test_simulation.20191028T162449d084
  ▸ E2E_test_simulation.20191029T173727d772
  ▸ E2E_test_simulation.20191030T111432d444
  ▾ E2E_test_simulation.20191030T162254d059
    ▸ exec1.20191030T162254d295
    ▸ exec2.20191030T162254d537
    ▸ exec3.20191030T162255d172
    ▸ exec4.20191030T162255d615
    ▸ exec5.20191030T162255d716
```

```
▾ simulations
    .gitignore
  ▸ E2E_test_simulation.20191028T160222d302
  ▸ E2E_test_simulation.20191028T162449d084
  ▸ E2E_test_simulation.20191029T173727d772
  ▾ E2E_test_simulation.20191030T111432d444
    ▸ 20180615T154902927.20191030T111432d634
    ▸ 20180615T155002927.20191030T111432d688
    ▸ 20180615T155502927.20191030T111432d872
    ▸ 20180615T155602927.20191030T111433d066
    ▸ 20180615T160102927.20191030T111433d207
  ▸ E2E_test_simulation.20191030T162254d059
```

Figure 4-100: (left) grouping of iteration/perturbation simulations; (right) grouping of timeline simulations.

In order to simplify simulation results directory names, symbolic links are used. Each time a simulation is executed a Linux symbolic link is generated in the file system with the name of the simulation being executed (appended by ".last") and pointing to the corresponding simulation execution directory. Each time a simulation is re-run the symbolic link is re-generated pointing to the latest simulation execution (the one with the latest timestamp). Keep in mind that these symbolic links are not generated in Windows. The contents of openSF's installation directory and the simulation execution ones can be reviewed in the File System tab of the side bar, see section 4.17.



```
▾ simulations
    .gitignore
  ▸ E2E_test_simulation.20191028T160222d302
  ▸ E2E_test_simulation.20191028T162449d084
  ▸ E2E_test_simulation.20191029T173727d772
  ▸ E2E_test_simulation.20191030T111432d444
  ▸ E2E_test_simulation.20191030T162254d059
  ▾ E2E_test_simulation.last
      E2E_test_simulation.sh
      Geometry.xml
      GlobalConfiguration.xml
      InputGeo.xml
      InputIonos.xml
      Instruments.xml
      Ionosphere.xml
      L1b.xml
      L2.xml
      Maps.xml
      Scene.xml
      geoConf.xml
      ionosphereConf.xml
      l1bProductConf.xml
      l2RetConf.xml
```

Figure 4-101: Side bar. File system (including symbolic link to last simulation)

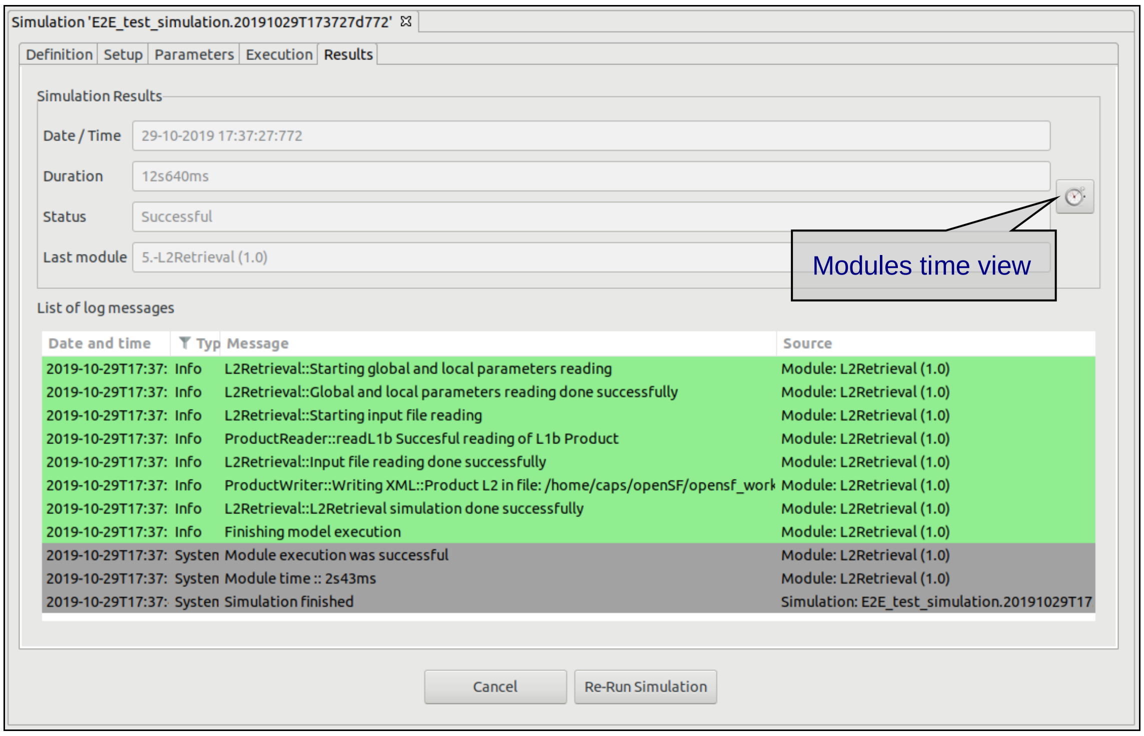
4.11.1. Result view

Users can inspect the results view of any executed simulation by opening it and selecting the "Results" tab. This view is similar to the simulation editor but including more information (see Figure 4-102). Accessing this functionality, users can consult the result of a simulation execution.

First of all, some data is presented in the "general properties" region showing these attributes:

- Date / time – this is the local computer date and time that the execution began. This date and time can also be part of the simulation identifier to distinguish this simulation execution from others;
- Duration – the time (in minutes and seconds) elapsed from the starting time until the execution was finished or interrupted;
- Status – the overall status of the execution. The possible values are "Failed, "Successful" and "Aborted";
- Last module – This is the number and identifier of the last module successfully executed. In case of a successful execution, this module must coincide with the very last module of the simulation. This information is useful for the user to know which module was erroneous.

Second, the Log tab is filled with the log messages generated by the simulation execution. Users can access all these messages to check its performance.



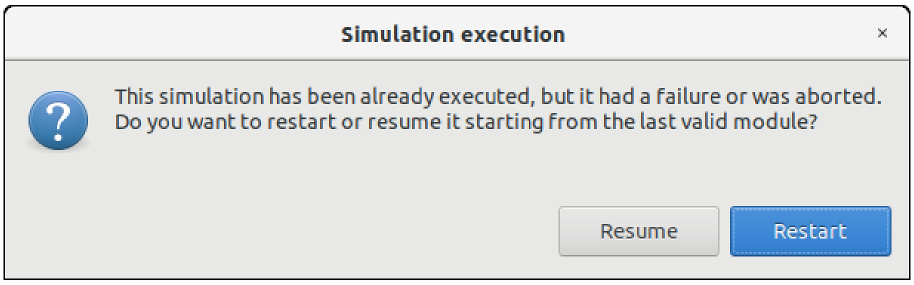
Date / Time: 29-10-2019 17:37:27:772
Duration: 12s640ms
Status: Successful
Last module: 5.-L2Retrieval (1.0)

Date and time	Typ	Message	Source
2019-10-29T17:37:	Info	L2Retrieval::Starting global and local parameters reading	Module: L2Retrieval (1.0)
2019-10-29T17:37:	Info	L2Retrieval::Global and local parameters reading done successfully	Module: L2Retrieval (1.0)
2019-10-29T17:37:	Info	L2Retrieval::Starting input file reading	Module: L2Retrieval (1.0)
2019-10-29T17:37:	Info	ProductReader::readL1b Succesful reading of L1b Product	Module: L2Retrieval (1.0)
2019-10-29T17:37:	Info	L2Retrieval::Input file reading done successfully	Module: L2Retrieval (1.0)
2019-10-29T17:37:	Info	ProductWriter::Writing XML::Product L2 in file: /home/caps/openSF/opensf_work	Module: L2Retrieval (1.0)
2019-10-29T17:37:	Info	L2Retrieval::L2Retrieval simulation done successfully	Module: L2Retrieval (1.0)
2019-10-29T17:37:	Info	Finishing model execution	Module: L2Retrieval (1.0)
2019-10-29T17:37:	Systen	Module execution was successful	Module: L2Retrieval (1.0)
2019-10-29T17:37:	Systen	Module time :: 2s43ms	Module: L2Retrieval (1.0)
2019-10-29T17:37:	Systen	Simulation finished	Simulation: E2E_test_simulation.20191029T17

Figure 4-102: Execution results

4.11.2. Result re-run

Accessing to this functionality, users can repeat the execution of a previously executed simulation. If the simulation execution was successful, the system just creates another execution (changing the starting date and time) but, if the previous execution was aborted or failed, the system will inform the user with the dialog shown below.



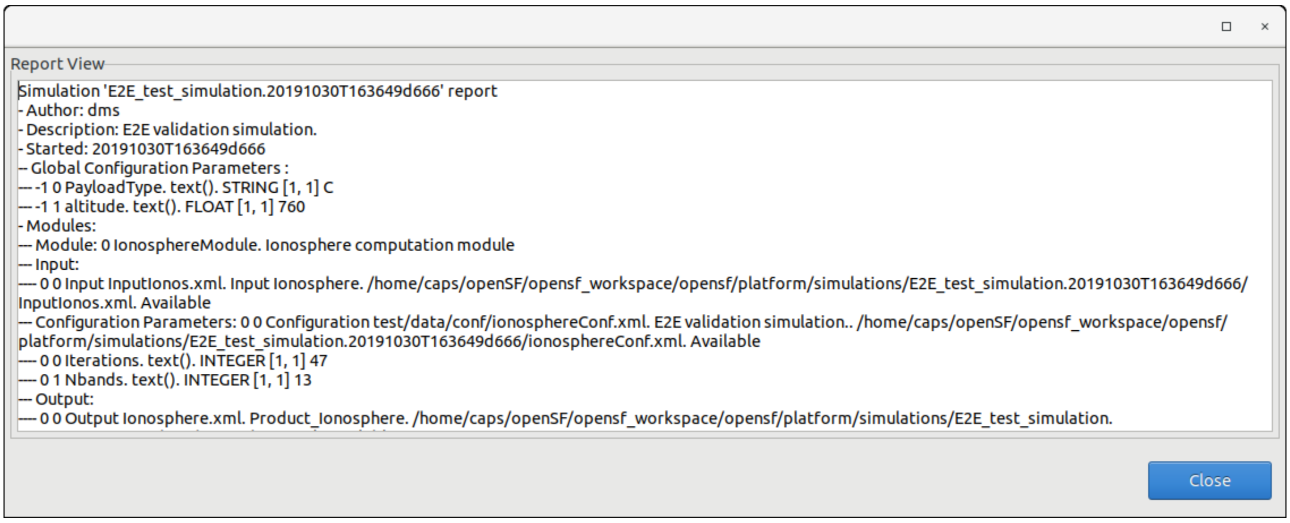
This simulation has been already executed, but it had a failure or was aborted.
Do you want to restart or resume it starting from the last valid module?

Figure 4-103: Result. Re-run

Users now can choose to restart the execution from the beginning or try to resume the execution, that is, to continue the execution from the last valid module. So, the execution will continue provided that the outer conditions that ruined the previous run have been corrected.

4.11.3. Report generation

Clicking on the "Generate report" option from the Results pop-up menu accesses this functionality. A window similar to the one shown in Figure 4-104 is presented to the user.



```
Simulation 'E2E_test_simulation.20191030T163649d666' report
- Author: dms
- Description: E2E validation simulation.
- Started: 20191030T163649d666
-- Global Configuration Parameters :
--- -1 0 PayloadType. text(). STRING [1, 1] C
--- -1 1 altitude. text(). FLOAT [1, 1] 760
- Modules:
-- Module: 0 IonosphereModule. Ionosphere computation module
-- Input:
---- 0 0 Input InputIonos.xml. Input Ionosphere. /home/caps/openSF/opensf_workspace/opensf/platform/simulations/E2E_test_simulation.20191030T163649d666/
InputIonos.xml. Available
-- Configuration Parameters: 0 0 Configuration test/data/conf/ionosphereConf.xml. E2E validation simulation.. /home/caps/openSF/opensf_workspace/opensf/
platform/simulations/E2E_test_simulation.20191030T163649d666/ionosphereConf.xml. Available
---- 0 0 Iterations. text(). INTEGER [1, 1] 47
---- 0 1 Nbands. text(). INTEGER [1, 1] 13
-- Output:
---- 0 0 Output Ionosphere.xml. Product_Ionosphere. /home/caps/openSF/opensf_workspace/opensf/platform/simulations/E2E_test_simulation.
```

Figure 4-104: Execution report

This execution report consists in a textual description of the same data that users can access with the "Result view" functionality. The only difference resides in that this textual information can be copied and pasted into another application outside the openSF system.

4.11.4. Delete result

Users can select a certain execution result and choose the option to delete it. Once users confirm the operation the execution result is erased from the repository and the file system. Log messages associated with this simulation execution result will also be erased.

4.11.5. Modules execution time

In Figure 4-102 the simulation results view is presented. Within this panel there is a clock icon button in the mid-right side that presents the time consumed by each module involved in a simulation run.

Module execution time is presented in a new window with three tabs:

1. *Module Times tab*: presents the module execution time in a bar chart graph*. See Figure 4-105
2. *Time Statistics tab*: shows a pie with the simulation time percentage consumed by each module. In case there is more than one Simulation involving one Module, time is divided by the number of module repetitions. See Figure 4-106
3. *Module Times Table tab*: shows the same information that the bar chart graph but in a table. See Figure 4-107

**Hint: Users can zoom within bar chart graph in order to better visualize module times*

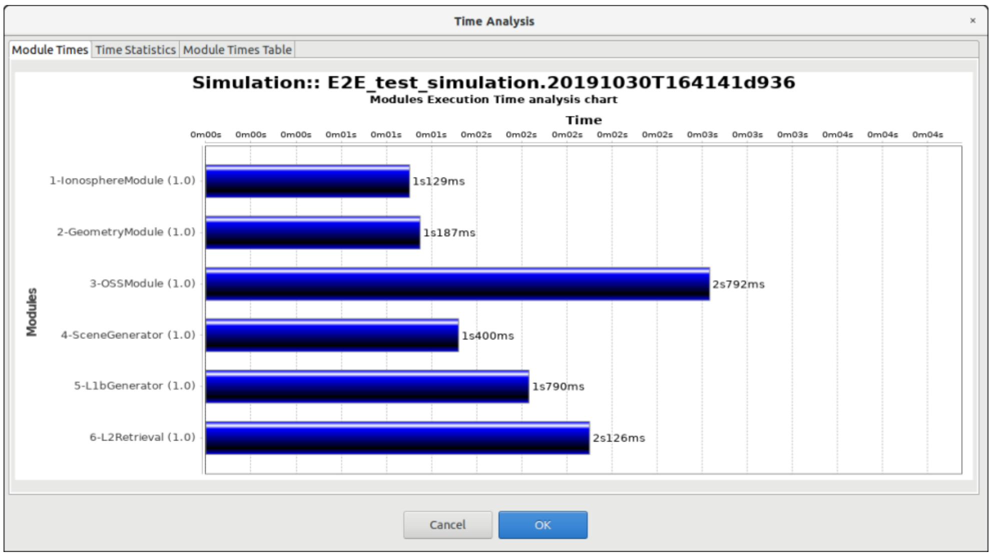

Time Analysis
Module Times
Time Statistics
Module Times Table
Simulation:: E2E_test_simulation.20191030T164141d936
Modules Execution Time analysis chart
Time
0m00s 0m00s 0m00s 0m01s 0m01s 0m01s 0m02s 0m02s 0m02s 0m02s 0m02s 0m03s 0m03s 0m03s 0m04s 0m04s 0m04s
Modules
1-IonosphereModule (1.0) 1s129ms
2-GeometryModule (1.0) 1s187ms
3-OSSModule (1.0) 2s792ms
4-SceneGenerator (1.0) 1s400ms
5-L1bGenerator (1.0) 1s790ms
6-L2Retrieval (1.0) 2s126ms
Cancel
OK

Figure 4-105 Bar graph showing Module times

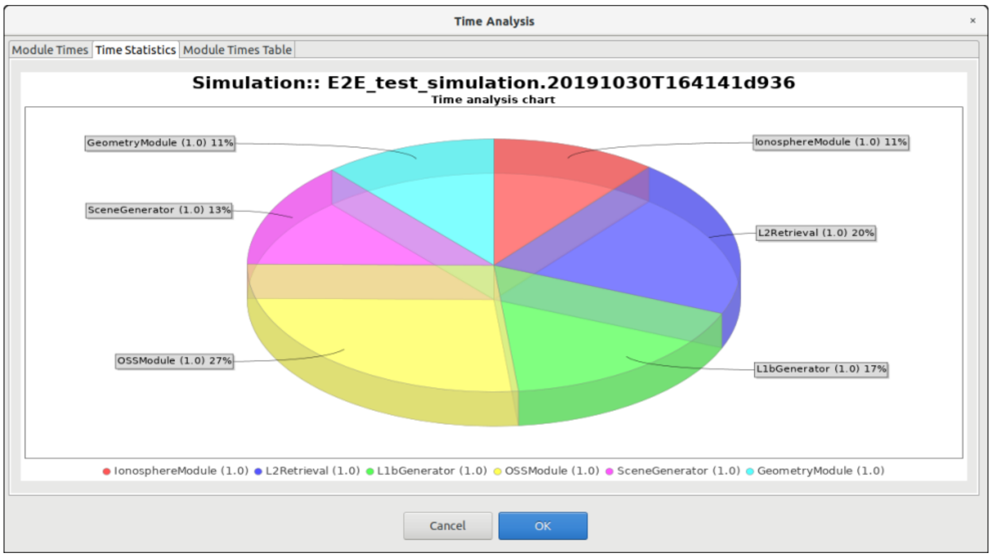
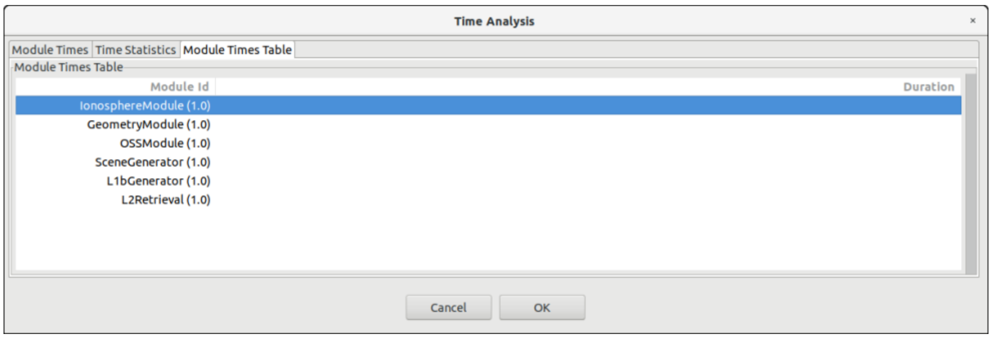

Time Analysis
Module Times
Time Statistics
Module Times Table
Simulation:: E2E_test_simulation.20191030T164141d936
Time analysis chart
GeometryModule (1.0) 11%
IonosphereModule (1.0) 11%
SceneGenerator (1.0) 13%
L2Retrieval (1.0) 20%
OSSModule (1.0) 27%
L1bGenerator (1.0) 17%
● IonosphereModule (1.0) ● L2Retrieval (1.0) ● L1bGenerator (1.0) ● OSSModule (1.0) ● SceneGenerator (1.0) ● GeometryModule (1.0)
Cancel
OK

Figure 4-106 Pie chart showing the percentage of time



Module Times Table

Module Id	Duration
IonosphereModule (1.0)	
GeometryModule (1.0)	
OSSModule (1.0)	
SceneGenerator (1.0)	
L1bGenerator (1.0)	
L2Retrieval (1.0)	

Figure 4-107 Table showing Module times

4.12. Logs

Executing simulations has a secondary consequence: a set of events are produced and stored for the system. These events are described in detail in section 4.9.5.

Log messages are stored in the file `$OPENSF_HOME/simulations/openSF.log` for global events, and in a file named `log/simulation.log` under each simulation folder for events related with a specific simulation. Also, if a certain application setting is enabled (see §4.5.1), log files are also stored for each module in a simulation, in the same folder and named after the related module.

Users can access to the complete set of logs stored by the system in the "Logs" menu (Figure 4-108) from the main menu.

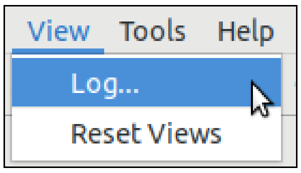
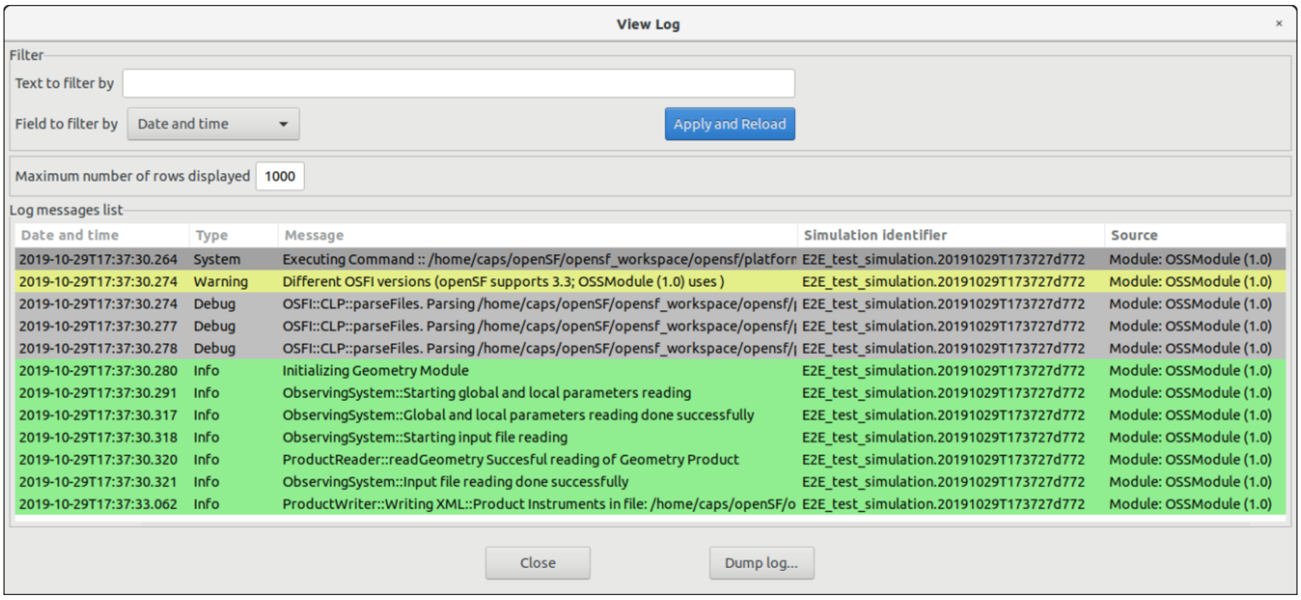


Figure 4-108: Logs menu

4.12.1. Log messages list

The figure below shows a window with a list of log messages stored by the system. As can be seen, the table shows the computer date and time when the platform intercepted the event, the type of the event, a message describing the event, the identifier of the simulation associated to the event and its detailed source (module, simulation, simulation or system).

This list of events is sorted (by default) in increasing time order until filling the "Maximum number of rows displayed" field. Users can change the number of log messages to be displayed. For example, if the "Maximum number of rows displayed" is set to 10, the list displays the last 10 messages



Text to filter by:
Field to filter by: Date and time
Maximum number of rows displayed: 1000

Log messages list

Date and time	Type	Message	Simulation identifier	Source
2019-10-29T17:37:30.264	System	Executing Command :: /home/caps/openSF/opensf_workspace/opensf/platforn	E2E_test_simulation.20191029T173727d772	Module: OSSModule (1.0)
2019-10-29T17:37:30.274	Warning	Different OSFI versions (openSF supports 3.3; OSSModule (1.0) uses)	E2E_test_simulation.20191029T173727d772	Module: OSSModule (1.0)
2019-10-29T17:37:30.274	Debug	OSFI::CLP::parseFiles. Parsing /home/caps/openSF/opensf_workspace/opensf/	E2E_test_simulation.20191029T173727d772	Module: OSSModule (1.0)
2019-10-29T17:37:30.277	Debug	OSFI::CLP::parseFiles. Parsing /home/caps/openSF/opensf_workspace/opensf/	E2E_test_simulation.20191029T173727d772	Module: OSSModule (1.0)
2019-10-29T17:37:30.278	Debug	OSFI::CLP::parseFiles. Parsing /home/caps/openSF/opensf_workspace/opensf/	E2E_test_simulation.20191029T173727d772	Module: OSSModule (1.0)
2019-10-29T17:37:30.280	Info	Initializing Geometry Module	E2E_test_simulation.20191029T173727d772	Module: OSSModule (1.0)
2019-10-29T17:37:30.291	Info	ObservingSystem::Starting global and local parameters reading	E2E_test_simulation.20191029T173727d772	Module: OSSModule (1.0)
2019-10-29T17:37:30.317	Info	ObservingSystem::Global and local parameters reading done successfully	E2E_test_simulation.20191029T173727d772	Module: OSSModule (1.0)
2019-10-29T17:37:30.318	Info	ObservingSystem::Starting input file reading	E2E_test_simulation.20191029T173727d772	Module: OSSModule (1.0)
2019-10-29T17:37:30.320	Info	ProductReader::readGeometry Succesful reading of Geometry Product	E2E_test_simulation.20191029T173727d772	Module: OSSModule (1.0)
2019-10-29T17:37:30.321	Info	ObservingSystem::Input file reading done successfully	E2E_test_simulation.20191029T173727d772	Module: OSSModule (1.0)
2019-10-29T17:37:33.062	Info	ProductWriter::Writing XML::Product Instruments in file: /home/caps/openSF/o	E2E_test_simulation.20191029T173727d772	Module: OSSModule (1.0)

Figure 4-109: Logs list view

Users can also filter this list. Users can select a field, input a string that must contain this field and press the “filter” button. Then another search is performed into the system and the results are shown on screen. Only records fulfilling the filter restriction will be shown. When clearing the filter text, the system shall retrieve again the full set of log messages.

Moreover, users can access the “dump log” functionality at the bottom of the window. Once selected, users can select the name and location of the log destination file. Then, the list of logs shown by the window is stored in the file system.

4.13. User Roles

openSF, as a simulator integration framework, intends to support two different types of users whose goals are clearly distinct. Each such type of use requires a different set of features for their typical use of the tool.

On the one hand, there is the user responsible for the the development of the simulator modules and their integration into openSF to compose the simulator. This user is expected to have a deep understanding of openSF, its capacities and limitations. Typically, the work of this user unfolds during the development phase of the simulator, and it is expected that a significant number of modifications to the openSF simulation elements (i.e. descriptors, modules and simulations) will be needed until a state of maturity is reached and the simulator can be considered production ready. For these users, openSF provides the features enabled by “Developer Mode”.

On the other hand, there is the user of the E2E simulators integrated in openSF. This user is not expected to understand openSF in detail, as his goal is to use the fully integrated simulator. This type of user is typically interested in modifying simulation inputs, executing the pre-defined simulations and collecting the results. In order to ensure reliable and repeatable results, this type of user should be not required (and eventually denied) to modify the openSF simulation elements. openSF provides a restricted set of features for these users, known as “Normal Mode”.

By default, openSF is configured in “Normal Mode” and the user role selection toolbar is hidden. If the user desires to create or modify any openSF simulation element, the “Enable user role selection” option can be enabled in the Application Settings (see section 4.5.1) to display the user role toolbar, and thus select the “Developer Mode”.

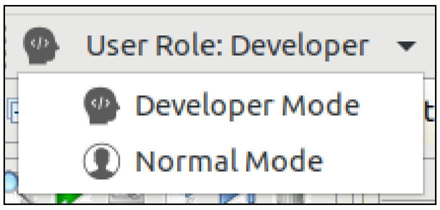


Figure 4-110: User Role selection toolbar

The main differences between these two modes are the following:

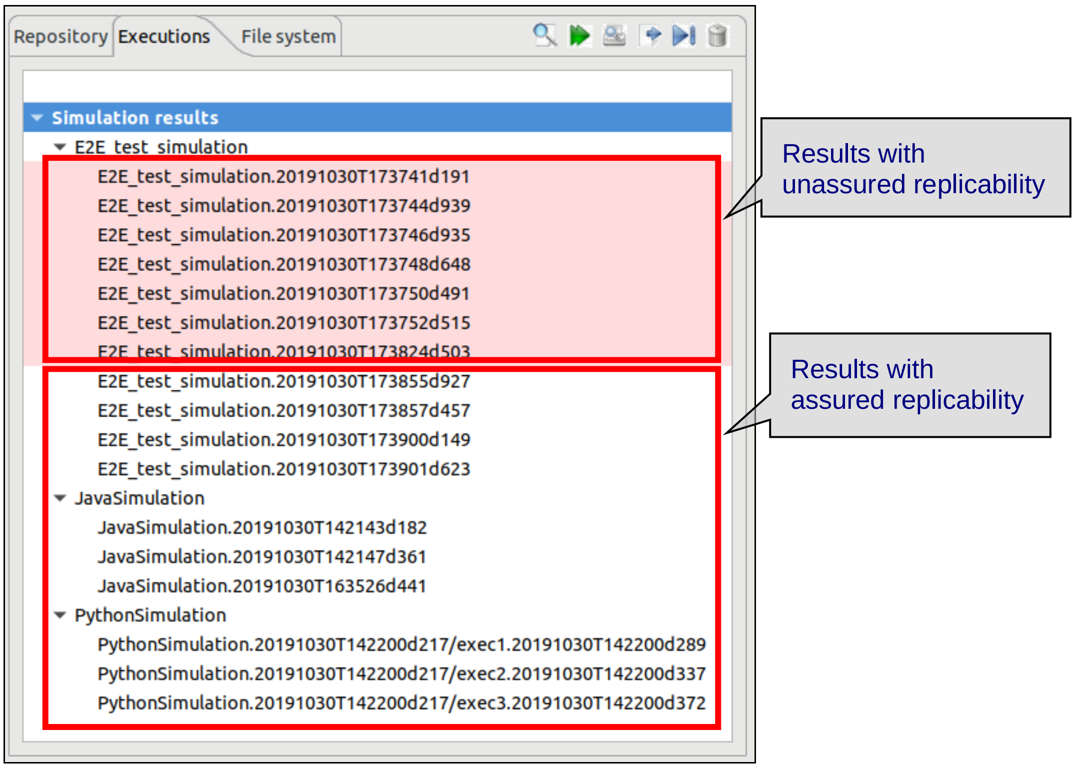
- ❑ Developer Mode:
  - The user can create new simulation elements (e.g. descriptors, modules, simulations).
  - The user can modify previously created simulation elements.
  - The user can execute simulation simulations.

  - The repeatable production of the results is not guaranteed as simulation elements can be modified between two simulation executions.
- ❑ Normal Mode:
  - The user can view the previously created simulation element.
  - The user can create, modify and execute simulations.
  - The reliable and repeatable production of the results is guaranteed.

In both cases the user can delete the existing simulation elements, potentially resulting in cascading deletions of any related element which depends on deleted elements (e.g. deleting a descriptor will cause the deletion of all modules that refer to it, and of all simulations that refers to those deleted modules, eventually reaching all the results of those simulations).

Therefore, if during the development stage of the simulator, one of its module's inputs get modified and a created descriptor is no longer needed but it is desired to keep the rest of the simulator unchanged, the user cannot directly delete it. Instead, while in "Developer Mode", the affected modules shall be first modified to remove that descriptor from its inputs/outputs and only then it can be safely removed. As it might be expected, this kind of practices break the replicability of the results, that's why they can only be performed in "Developer Mode".

openSF applies a simulation fingerprint to keep track of the simulation results whose reliability is not assured. Unreliable results are displayed in the Executions tab of the Navigation pane with a light red background. Replicable results are shown on a white background (see Figure 4-111). At the end of the simulator development stage and before entering in production stage, all the results with an unassured replicability should be removed.


Repository
Executions
File system
Results with unassured replicability
Results with assured replicability

Simulation results
E2E_test_simulation
E2E_test_simulation.20191030T173741d191
E2E_test_simulation.20191030T173744d939
E2E_test_simulation.20191030T173746d935
E2E_test_simulation.20191030T173748d648
E2E_test_simulation.20191030T173750d491
E2E_test_simulation.20191030T173752d515
E2E_test_simulation.20191030T173824d503
E2E_test_simulation.20191030T173855d927
E2E_test_simulation.20191030T173857d457
E2E_test_simulation.20191030T173900d149
E2E_test_simulation.20191030T173901d623
JavaSimulation
JavaSimulation.20191030T142143d182
JavaSimulation.20191030T142147d361
JavaSimulation.20191030T163526d441
PythonSimulation
PythonSimulation.20191030T142200d217/exec1.20191030T142200d289
PythonSimulation.20191030T142200d217/exec2.20191030T142200d337
PythonSimulation.20191030T142200d217/exec3.20191030T142200d372

Figure 4-111: Executions tab. Results colour code based on its replicability

A result is considered as potentially not reliable if any of the following criteria applies:

- ❑ The result was obtained while in “Developer Mode”.
- ❑ The computed fingerprint of the simulation, or any of its components, at calculation’s time does not match the current fingerprint of that simulation and its current components.

4.14. Parameter Perturbations

In this section the openSF perturbation system is detailed. This functionality is intended for adding new capabilities to simulation iteration (section 4.9.2.10).

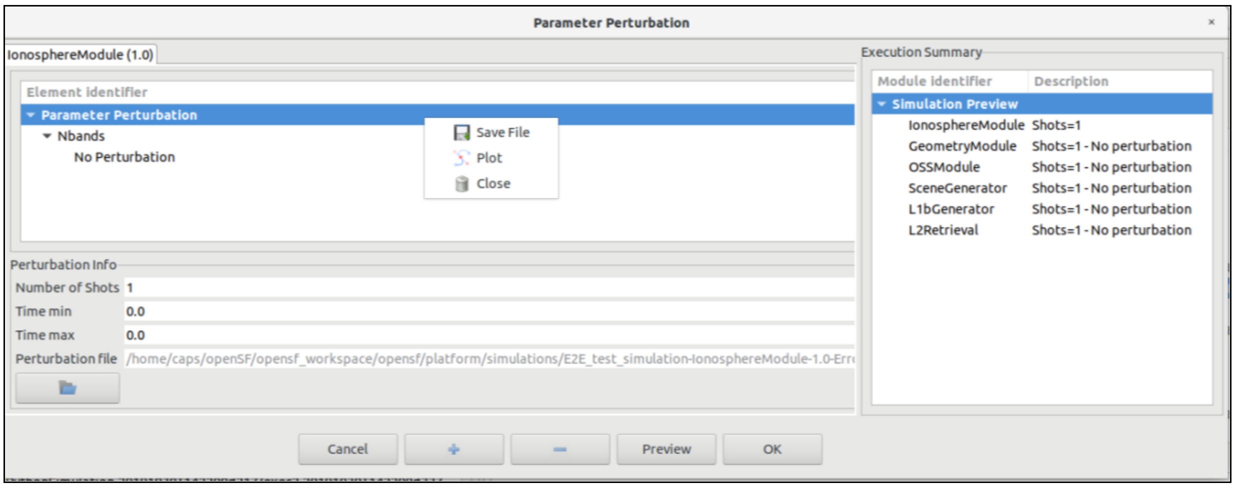
The simulation perturbation system brings users with the following functionalities:

- ❑ Independently define a perturbation function for each module involved in a simulation.
- ❑ Combine different functions for perturbing parameter values (Ex: sinusoidal with amplitude values varying as a normal random distribution)
- ❑ Two different execution schemes, Statistical and Combined modes (section 4.14.3)

4.14.1. Parameter Perturbation interface

The module perturbation interface is composed by the following panels:

- ❑ A tab that shows for each module the “Perturbation Function Tree”. This tree presents module parameters and perturbation function specified.
- ❑ An info panel where perturbation parameters can be modified:
  - Number of shots (integer format)
  - Time min and time max for analytical perturbations (double format)
  - Perturbation file where function definition is stored (XML format)
- ❑ Execution summary panel where simulation execution is outlined prior saving/running it.
- ❑ A tool bar for adding new functions to a module “Perturbation Function Tree”



Perturbation Info

Number of Shots	1
Time min	0.0
Time max	0.0
Perturbation file	/home/caps/openSF/opensf_workspace/opensf/platform/simulations/E2E_test_simulation-IonosphereModule-1.0-Err

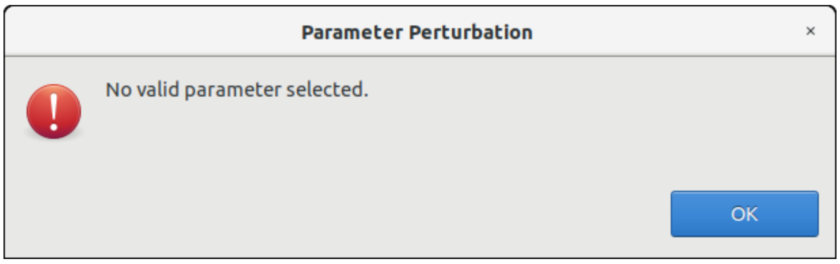
Module identifier	Description
Simulation Preview	
IonosphereModule	Shots=1
GeometryModule	Shots=1 - No perturbation
OSSModule	Shots=1 - No perturbation
SceneGenerator	Shots=1 - No perturbation
L1bGenerator	Shots=1 - No perturbation
L2Retrieval	Shots=1 - No perturbation

Figure 4-112 openSF Perturbation system main window

4.14.2. Defining a new perturbation

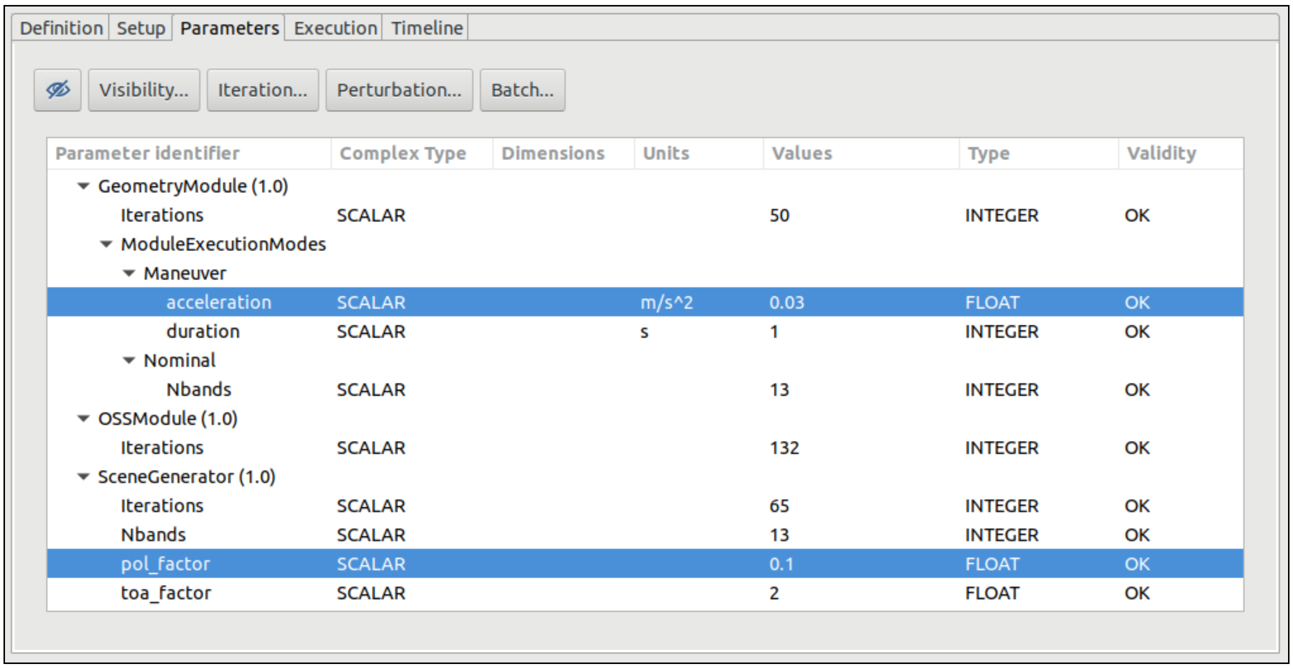
This section describes the steps that an user shall follow in order to add a new perturbation to a simulation.

1. Select from Simulation Creation/Edition interface the desired INTEGER/FLOAT parameters. If no valid parameters are selected the following message will appear.
   a. User can select more than one parameter using the Ctrl key.
   b. Global configuration parameters cannot be perturbed



No valid parameter selected.

Figure 4-113 No valid parameters selected



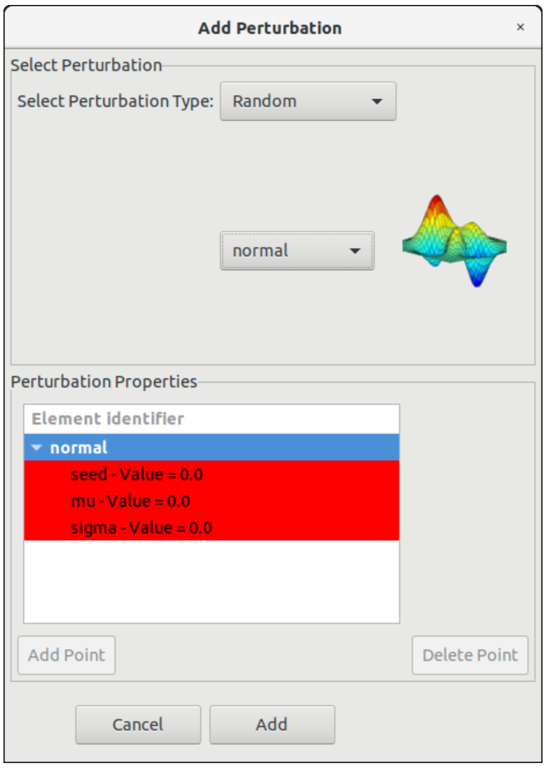
Parameter identifier	Complex Type	Dimensions	Units	Values	Type	Validity
GeometryModule (1.0)						
Iterations	SCALAR			50	INTEGER	OK
ModuleExecutionModes						
Maneuver						
acceleration	SCALAR		m/s^2	0.03	FLOAT	OK
duration	SCALAR		s	1	INTEGER	OK
Nominal						
Nbands	SCALAR			13	INTEGER	OK
OSSModule (1.0)						
Iterations	SCALAR			132	INTEGER	OK
SceneGenerator (1.0)						
Iterations	SCALAR			65	INTEGER	OK
Nbands	SCALAR			13	INTEGER	OK
pol_factor	SCALAR			0.1	FLOAT	OK
toa_factor	SCALAR			2	FLOAT	OK

Figure 4-114 Selection of parameters for perturbation

2. Launch parameter perturbation system using the button in the top part of "*List of parameters*" panel
3. Select a Perturbation Tree leaf and click on "+" button from the Perturbate Simulation toolbar. This action will pop-up the "Select Function" frame.

- Remember that functions can be nested so "+" button is also used for creating complex functions. (see Figure 4-116)

- If user want to setup another perturbation function for a parameter remember to previously delete it selecting the node and pressing “-“ button from the Perturbate Simulation toolbar.


Add Perturbation

Select Perturbation

Select Perturbation Type: Random

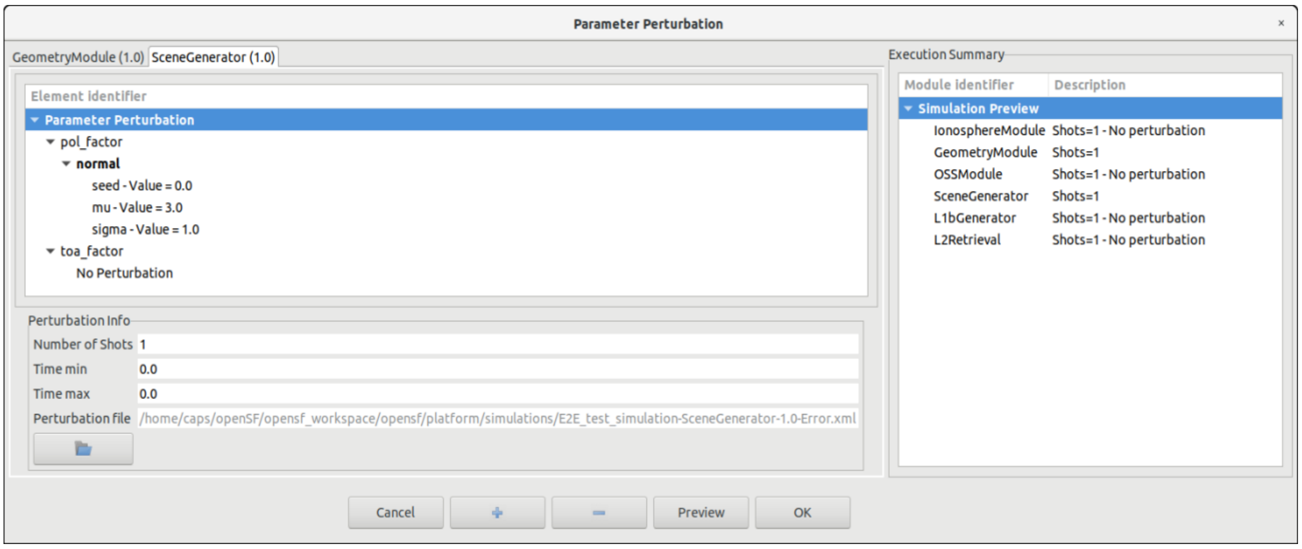
normal

Perturbation Properties

Element identifier
- normal
  - seed - Value = 0.0
  - mu - Value = 0.0
  - sigma - Value = 0.0

Figure 4-115 Adding a perturbation function to a module parameter

4. In the “Add Perturbation” window insert the function parameters checking that table frame changes to white colour (Ex: Random --> normal function must have a sigma greater than 0, second operand of BinaryOperations --> Root must be positive)


Parameter Perturbation

GeometryModule (1.0) | SceneGenerator (1.0)

Element identifier
- Parameter Perturbation
  - pol_factor
    - normal
      - seed - Value = 0.0
      - mu - Value = 3.0
      - sigma - Value = 1.0
  - toa_factor
    - No Perturbation

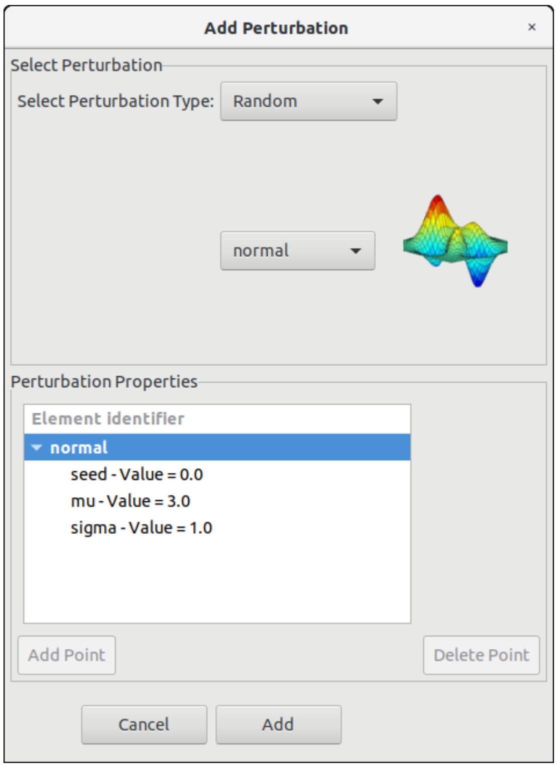
Perturbation Info

Number of Shots: 1
Time min: 0.0
Time max: 0.0
Perturbation file: /home/caps/openSF/opensf_workspace/opensf/platform/simulations/E2E_test_simulation-SceneGenerator-1.0-Error.xml

Execution Summary

Module identifier	Description
Simulation Preview	
IonosphereModule	Shots=1 - No perturbation
GeometryModule	Shots=1
OSSModule	Shots=1 - No perturbation
SceneGenerator	Shots=1
L1bGenerator	Shots=1 - No perturbation
L2Retrieval	Shots=1 - No perturbation

Figure 4-116 Complex perturbation function


Add Perturbation

Select Perturbation

Select Perturbation Type: Random

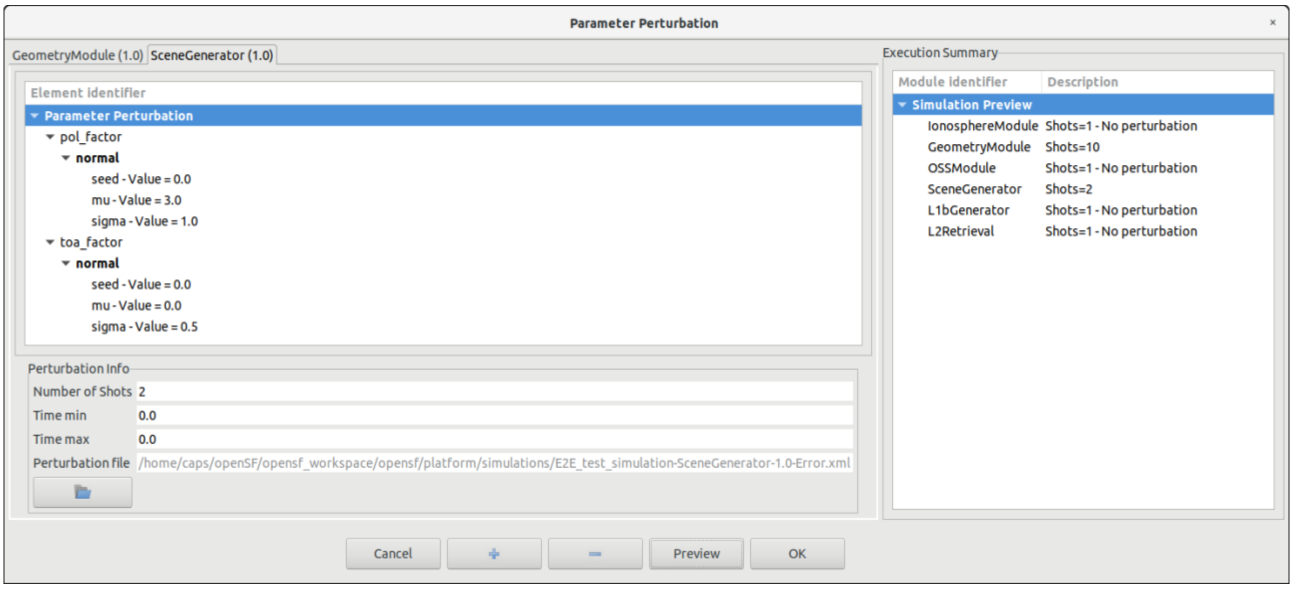
normal

Perturbation Properties

Element identifier
- normal
  - seed - Value = 0.0
  - mu - Value = 3.0
  - sigma - Value = 1.0

Figure 4-117 Random perturbation properties

5. Click on "Add" button to update *Perturbation Function Tree*
6. Change number of shots, min/max time as desired (remember that time is only used in analytical perturbations)
   - If the error function assigned to the parameter is an analytical one, it is assumed that the perturbation per shot actually can be represented by a time series. For example, it could be a perturbation $Y = cos(\Omega * t)$; then t values (in number equal to Number of shots) will be drawn between "Time Min" and "Time Max". Note that the step between points is (tmax-tmin) / number of shots
7. Preview execution, selecting the desired perturbed execution scheme, see section 4.14.3
8. Accept the simulation perturbation, adding it to simulation definition by clicking on "Accept" button.



Element identifier
- Parameter Perturbation
  - pol_factor
    - normal
      - seed - Value = 0.0
      - mu - Value = 3.0
      - sigma - Value = 1.0
  - toa_factor
    - normal
      - seed - Value = 0.0
      - mu - Value = 0.0
      - sigma - Value = 0.5

Perturbation Info

Number of Shots	2
Time min	0.0
Time max	0.0
Perturbation file	/home/caps/openSF/opensf_workspace/opensf/platform/simulations/E2E_test_simulation-SceneGenerator-1.0-Error.xml

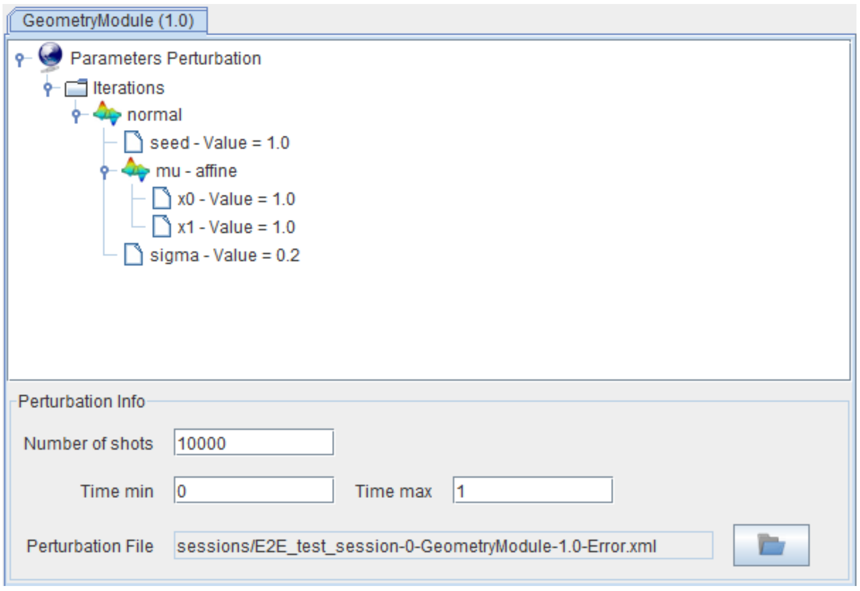
Simulation Preview

Module identifier	Description
IonosphereModule	Shots=1 - No perturbation
GeometryModule	Shots=10
OSSModule	Shots=1 - No perturbation
SceneGenerator	Shots=2
L1bGenerator	Shots=1 - No perturbation
L2Retrieval	Shots=1 - No perturbation

Figure 4-118 Preview of Statistical Mode execution scheme

4.14.2.1. Additional operations

- ❑ Clear module perturbation: Right click on module tab and select “Close”
- ❑ Save perturbation into selected file: Dumps Perturbation Function Tree into an XML file. Right click on module tab and select “Save”
- ❑ Plot perturbation: Right click on the desired tab and select “Plot”. A new window will appear showing the time series and histogram for each perturbed parameter. Figure 4-119, Figure 4-120, Figure 4-121 show how a parameter perturbation can be defined and plotted from HMI.
- ❑ Load an external XML file with errors definition for a set of module parameters. Only errors matching with a module parameter name will be loaded within the system.



- Parameters Perturbation
  - Iterations
    - normal
      - seed - Value = 1.0
      - mu - affine
        - x0 - Value = 1.0
        - x1 - Value = 1.0
      - sigma - Value = 0.2

Perturbation Info

Number of shots	10000
Time min	0
Time max	1
Perturbation File	sessions/E2E_test_session-0-GeometryModule-1.0-Error.xml

Figure 4-119 Complex perturbation for a parameter

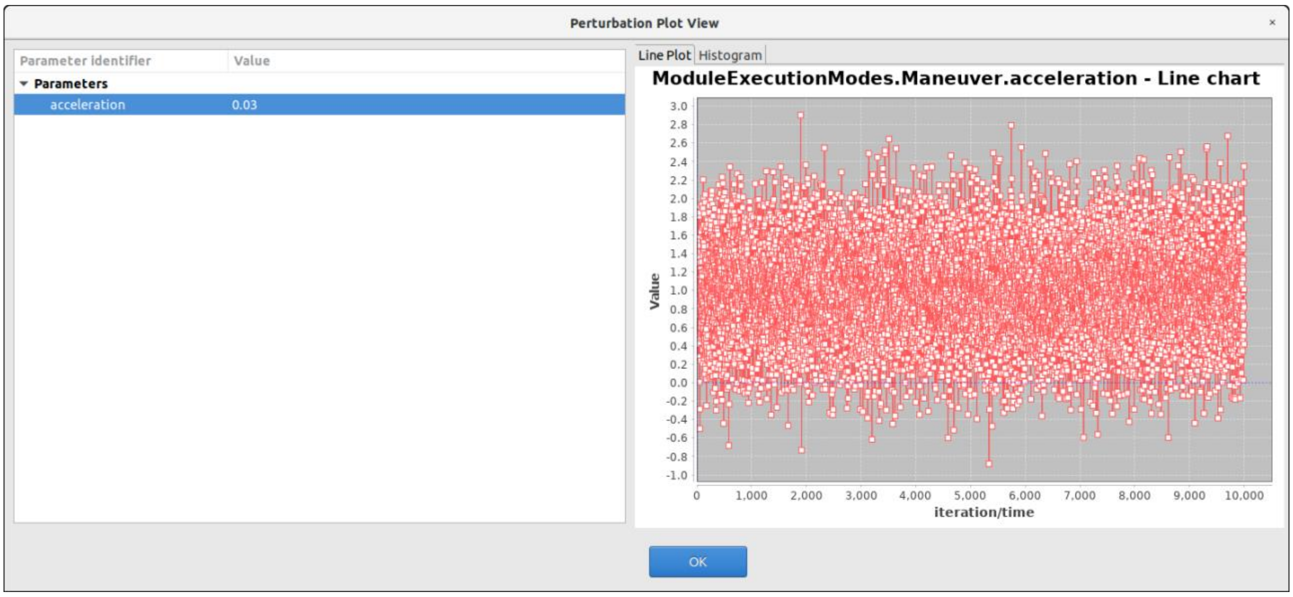


Figure 4-120 Time series line for a parameter perturbation

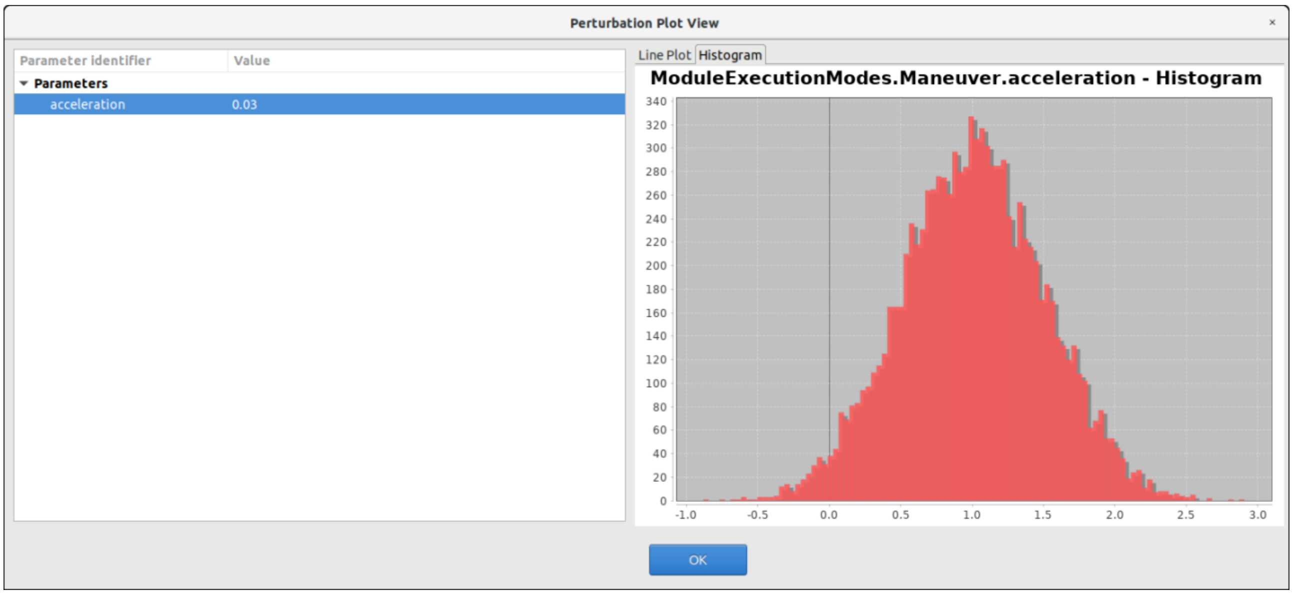
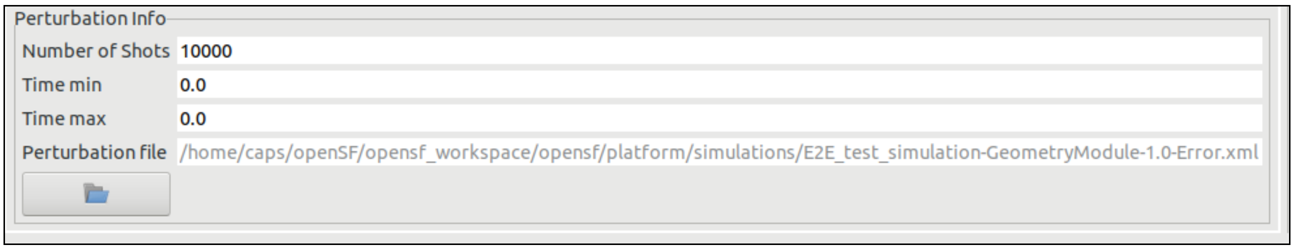


Figure 4-121 Histogram chart for a random parameter perturbation


Perturbation Info

Number of Shots	10000
Time min	0.0
Time max	0.0
Perturbation file	/home/caps/openSF/opensf_workspace/opensf/platform/simulations/E2E_test_simulation-GeometryModule-1.0-Error.xml

Figure 4-122 Loading an external error file

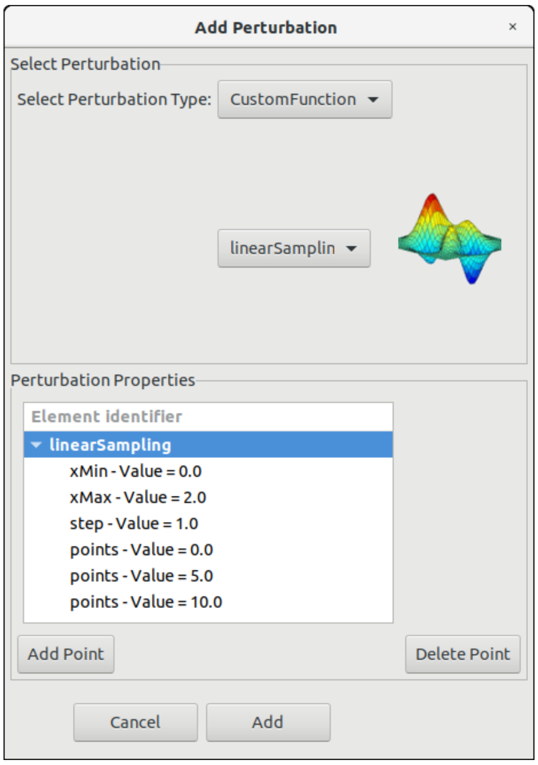
XML Error File syntax, below users can find an error file example with all available functions.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<errorsFile>
  <simTime value="50" />
  <parameter name="Affine function">
    <affine>
      <float value="50" />
      <float value="5" />
    </affine>
  </parameter>
  <parameter name="Bias function">
    <bias>
      <float value="10" />
    </bias>
  </parameter>
  <parameter name="Linear function">
    <linear>
      <float value="15" />
    </linear>
  </parameter>
  <parameter name="Parabolic function">
    <parabolic>
      <float value="1" />
      <float value="1" />
      <float value="1" />
    </parabolic>
  </parameter>
  <parameter name="Polynomial function">
    <polynomial>
      <float value="1" />
      <float value="1" />
      <float value="1" />
      <float value="1" />
    </polynomial>
  </parameter>
  <parameter name="Step function">
    <step>
      <float value="3" />
      <float value="10" />
      <float value="20" />
    </step>
  </parameter>
  <!-- Test XML Comment: Sinusoidal Function -->
  <parameter name="Sinusoidal function">
    <sinusoidal>
      <float value="10" /> <!-- Amplitude -->
      <float value="10" /> <!-- Frequency -->
      <float value="0" /> <!-- Phase in degrees -->
    </sinusoidal>
  </parameter>
  <parameter name="Tangent function">
    <tangent>
      <float value="10" />
      <float value="1" />
      <float value="0" />
    </tangent>
  </parameter>

  <parameter name="Linear sampling">
    <linearSampling xMin="1.0" xMax="10.0" step="1">
      <float value="1" />
      <float value="3" />
      <float value="5" />
      <float value="7" />
      <float value="3" />
      <float value="2" />
      <float value="2" />
      <float value="10" />
      <float value="4" />
      <float value="3" />
    </linearSampling>
  </parameter>
  <parameter name="Polynomial sampling">
    <polynomialSampling xMin="1.0" xMax="10.0" step="1">
      <float value="1" />
      <float value="2" />
      <float value="1" />
      <float value="2" />
      <float value="1" />
      <float value="2" />
      <float value="1" />
      <float value="2" />
      <float value="1" />
      <float value="2" />
    </polynomialSampling>
  </parameter>
  <parameter name="Spline sampling">
    <splineSampling xMin="1.0" xMax="20.0" step="1">
      <float value="2" />
      <float value="3" />
      <float value="2" />
      <float value="3" />
      <float value="2" />
      <float value="3" />
      <float value="2" />
      <float value="3" />
      <float value="10" />
      <float value="2" />
      <float value="3" />
      <float value="2" />
      <float value="4" />
      <float value="7" />
      <float value="2" />
      <float value="3" />
      <float value="2" />
      <float value="3" />
      <float value="2" />
      <float value="7" />
    </splineSampling>
  </parameter>
  <parameter name="Beta distribution">
    <beta seed="1" v="2" w="5" xMin="0.0" xMax="1.0" />
  </parameter>
  <parameter name="Gamma distribution">
    <gamma seed="1" location="0.0" scale="0.5" shape="9" />
  </parameter>
  <parameter name="Exponential distribution">
    <exponential seed="1" a="1" b="1.5" />
  </parameter>
  <parameter name="Normal distribution">
    <normal seed="1" mu="100.0" sigma="10.0" />
  </parameter>
  <parameter name="Uniform distribution">
    <uniform seed="1" xMin="0" xMax="1" />
  </parameter>
  <parameter name="Poisson distribution">
    <poisson seed="1" mu="10" />
  </parameter>
  <parameter name="Truncated gaussian distribution">
    <truncatedGaussian seed="1" mu="0.5" sigma="0.2" xMin="0.4" xMax="0.6" />
  </parameter>
  <parameter name="Uniform discrete distribution">
    <uniformDiscrete seed="1" i="0" j="1" />
  </parameter>

  <parameter name="Custom PDF">
    <customPDF seed="24" xMin="0.0" xMax="12.0" step="1">
      <float value="7" />
      <float value="43" />
      <float value="21" />
      <float value="10" />
      <float value="2" />
      <float value="6" />
      <float value="23" />
      <float value="31" />
      <float value="7" />
      <float value="2" />
      <float value="7" />
      <float value="43" />
      <float value="21" />
    </customPDF>
  </parameter>
</errorsFile>
```

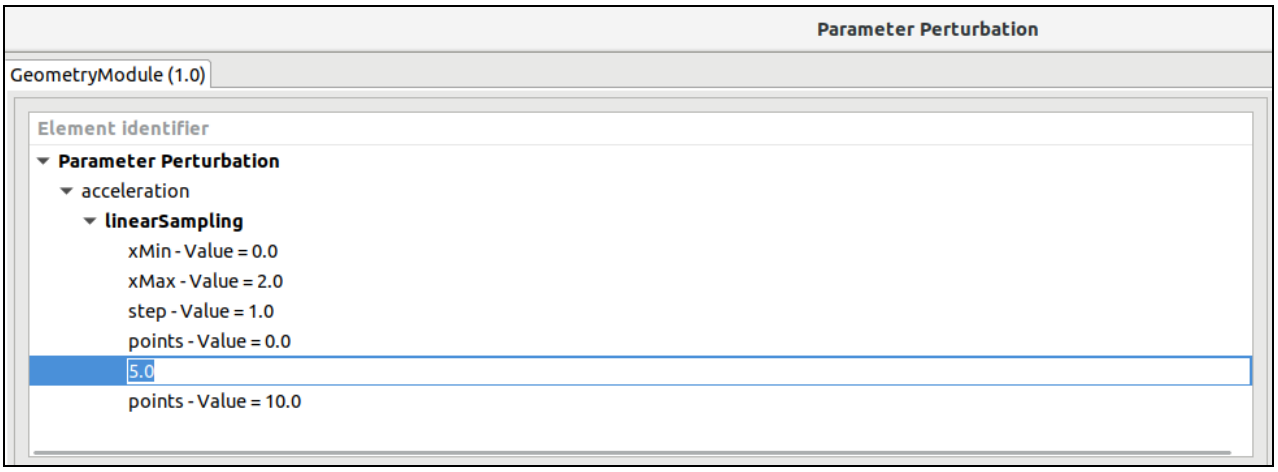
- Functions with a variable number of properties (such as sampling functions, custom probability density function etc …): users are able to add/del points through “+” and “-“ buttons of the “Select Function” interface. See section 4.14.4 for details



Element identifier
linearSampling
xMin - Value = 0.0
xMax - Value = 2.0
step - Value = 1.0
points - Value = 0.0
points - Value = 5.0
points - Value = 10.0

Figure 4-123 Function with variable number of properties (points)

- Additionally, users can change the double value of a leaf by double clicking on the Perturbation Function Tree



Element identifier
Parameter Perturbation
acceleration
linearSampling
xMin - Value = 0.0
xMax - Value = 2.0
step - Value = 1.0
points - Value = 0.0
5.0
points - Value = 10.0

Figure 4-124 Editing a double value of the Perturbation Tree

4.14.3. Perturbed execution modes

openSF iteration of parameters has been previously based on the combination of all possible values. This approach results in an exponential increasing number of executions depending on the number of parameters being iterated and the number of parameter values (i.e. the iteration of two parameters with ten different values each one, involves one hundred executions !!)

In many cases the solution provided by openSF was not the optimal one, statistical modules (as in SEPSO case), modules that need to run in different modes for each simulation (Sentinel 3 processors) or modules that need to be executed more than one time depending on a parameter (BIOMASS acquisitions). For coping with the needs of such simulators, two new possibilities have been implemented for handling iteration/perturbation of parameters or multiple runs. Each module can be configured with different parameter perturbations/iterations, no extra simulations shall be setup just a loop is introduced executing the module "*N*" times. Note that the output files/folders of each module will be the same and consequently **it is module responsibility the handling of this issue**.

The other approach is the possibility of setting up a number of simulations equal to the number of shots configured for the perturbations of each module (this approach has a constraint, all module perturbations shall be set-up with the same number of shots, the no-perturbed ones will be re-run with the same parameters).

The new execution schemes for perturbed simulations are:

1. ***Statistical Mode***: users are able to execute one module as many times as specified. Users shall note that module implementation shall be aware of this fact because openSF will execute the module on with same input and output files for each iteration. It is module responsibility to smartly handle this situation (Ex: Merge output etc…). In this case the number of module repetitions is completely independent.

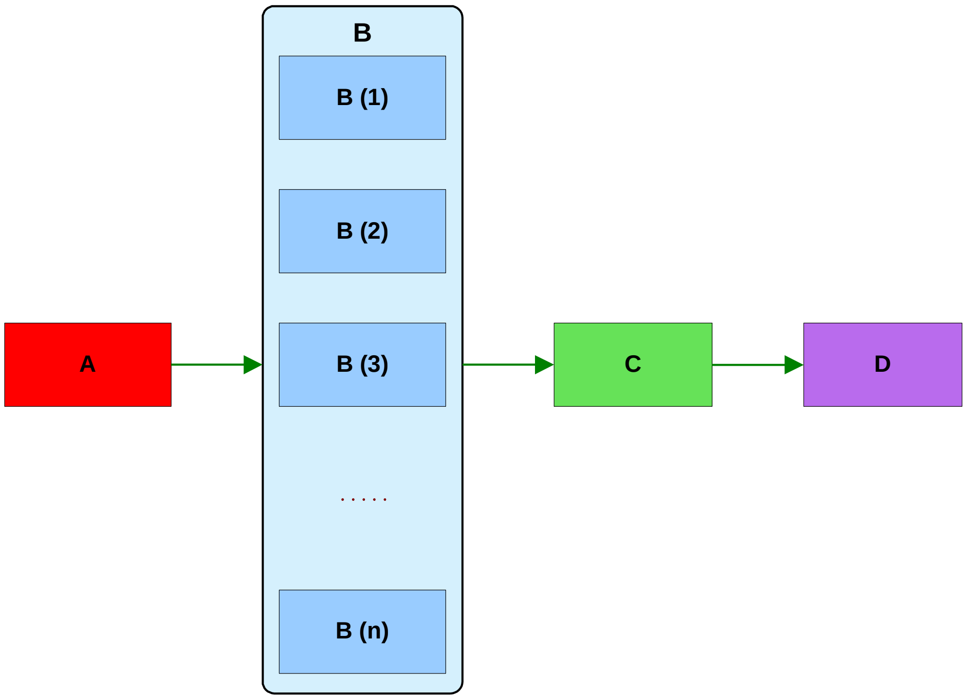


Figure 4-125 Statistical mode execution scheme

2. ***Combined Mode***: creates as many simulations as shots are specified. All module parameters are perturbed at the same time for each simulation. For this execution scheme it is required that all module perturbations have the same number of shots.

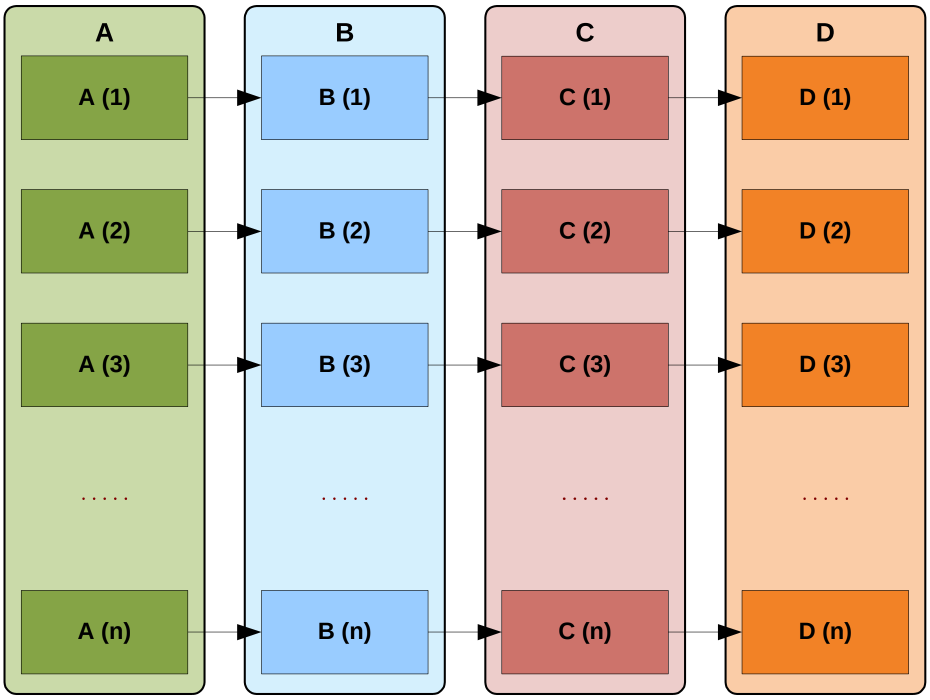
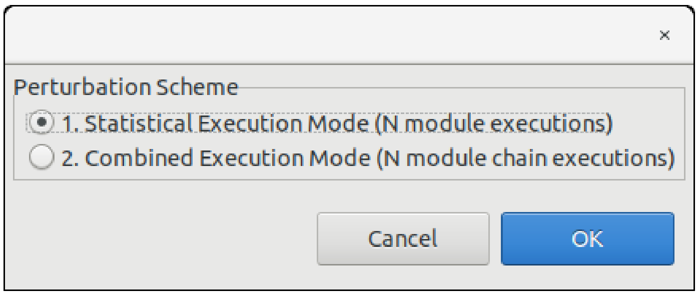


Figure 4-126 Combined mode execution scheme


Perturbation Scheme

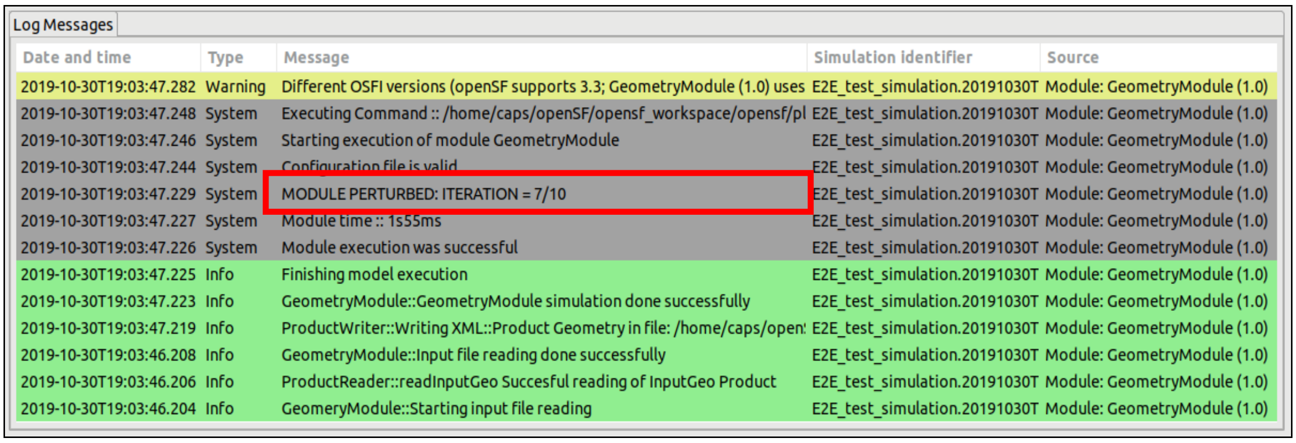
[✓ 1. Statistical Execution Mode (N module executions)]
2. Combined Execution Mode (N module chain executions)

Figure 4-127 Execution mode selector

4.14.3.1. Differences between execution modes

- In Statistical mode, modules shall handle output files as openSF specifies always the same filename for each shot.
- openSF in Combined mode creates parameter values in the moment the perturbation is accepted, in the other hand Statistical mode sets values just before module is executed, when XML configuration file is written.
- In order to make the Statistical mode repeatable by script, openSF creates a set of configuration files, one per shot specified, using as name convention "File_N.xml" where N is the iteration number. This is not needed in Combined mode as configuration files are stored in different simulation folders.

- ❑ Combined mode simulations are not handled differently by openSF, just as new Executions of a Simulation, Statistical mode instead adds a new log message specifying the module iteration number.


Log Messages

Date and time	Type	Message	Simulation identifier	Source
2019-10-30T19:03:47.282	Warning	Different OSFI versions (openSF supports 3.3; GeometryModule (1.0) uses	E2E_test_simulation.20191030T	Module: GeometryModule (1.0)
2019-10-30T19:03:47.248	System	Executing Command :: /home/caps/openSF/opensf_workspace/opensf/pl	E2E_test_simulation.20191030T	Module: GeometryModule (1.0)
2019-10-30T19:03:47.246	System	Starting execution of module GeometryModule	E2E_test_simulation.20191030T	Module: GeometryModule (1.0)
2019-10-30T19:03:47.244	System	Configuration file is valid	E2E_test_simulation.20191030T	Module: GeometryModule (1.0)
2019-10-30T19:03:47.229	System	MODULE PERTURBED: ITERATION = 7/10	E2E_test_simulation.20191030T	Module: GeometryModule (1.0)
2019-10-30T19:03:47.227	System	Module time :: 1s55ms	E2E_test_simulation.20191030T	Module: GeometryModule (1.0)
2019-10-30T19:03:47.226	System	Module execution was successful	E2E_test_simulation.20191030T	Module: GeometryModule (1.0)
2019-10-30T19:03:47.225	Info	Finishing model execution	E2E_test_simulation.20191030T	Module: GeometryModule (1.0)
2019-10-30T19:03:47.223	Info	GeometryModule::GeometryModule simulation done successfully	E2E_test_simulation.20191030T	Module: GeometryModule (1.0)
2019-10-30T19:03:47.219	Info	ProductWriter::Writing XML::Product Geometry in file: /home/caps/open:	E2E_test_simulation.20191030T	Module: GeometryModule (1.0)
2019-10-30T19:03:46.208	Info	GeometryModule::Input file reading done successfully	E2E_test_simulation.20191030T	Module: GeometryModule (1.0)
2019-10-30T19:03:46.206	Info	ProductReader::readInputGeo Succesful reading of InputGeo Product	E2E_test_simulation.20191030T	Module: GeometryModule (1.0)
2019-10-30T19:03:46.204	Info	GeomeryModule::Starting input file reading	E2E_test_simulation.20191030T	Module: GeometryModule (1.0)

Figure 4-128 Statistical mode iterations log message

4.14.4. Error Functions

4.14.4.1. Deterministic Functions

Deterministic functions are those whose value it is known in all the time domain.

- ❑ **Affine**

Calculates the perturbation as an affine value. An affine transformation consists in a linear transformation and a translation.

- $error = a_1 + a_0 * t$

- ❑ **Bias**

Calculates the perturbation as a constant value.

- ❑ **Linear**

Calculates the perturbation as a linear value:

- $p = a * t$

This is a particular case of affine transformation when translation variable is equals to 0.

- ❑ **Parabolic**

Calculates the perturbation as a parabolic value.

- $p = a_0 + a_1 * t + a_2 * t_2$

- ❑ **Polynomial**

Calculates the perturbation as a generic polynomial value. This function has as many float parameters as degrees of the desired polynomial plus one.

- **Step**

Calculates the perturbation as step function.

- if simTime < t => $p = a_0$
- if simTime > t => $p = a_1$

- **Sinusoidal**

Calculates the perturbation as sinusoidal function

- p = a * sin(2 * pi * f * t + phi)
- f(Hz)
- phi(deg)
- t(secs)

- **Tangent**

Calculates the perturbation as tangent function

- p = a * tan(2 * pi * f * t + phi)
- f(Hz)
- phi(deg)
- t(secs)

Remember that the tangent function have singularities when the angle evaluated is (+-)*n*pi/2.

4.14.4.2. Sampling Functions

The openSF error generation plugin implements three interpolation methods, linear, polynomial and spline sampling.

In order to define the points of the interpolation there is a common set of variables that are listed below.

- xMin: Min value of abscise axis
- xMax: Max value of abscise axis
- step: Increment between abscise values

The number of points must be :

$$\frac{xMax - xMin}{step} = nValues$$

- **Linear Sampling**

This function makes an interpolation with the given points assuming it follows a linear rule.

- **Polynomial Sampling**

This interpolation method builds a polynomial grade n, being n the number of specified points. This interpolation minimizes the Least Square Error. Ref: Neville Method.

❑ **Spline Sampling**

Interpolate the given “n” points with Cubic Splines Method.

How to use the sampling functions

The sampling functions are useful for cases where the perturbation is a function known at discrete instants. That is, F = {yj, xj}, j=1, …n. In such a case, openSF provides with the functionality of interpolating according different methods: for a given time xt calculate the corresponding perturbation in the discrete series {yj, xj} such that yt = F(xt)

The sampling functions configuration has to include yj values, because the xj have to be equally sampled, providing only the minimum (xMin), maximum (xMax) and sampling (step).

- The "linear sampling" method uses a linear interpolation between points
- The "polynomial sampling" method interpolates using a Neville polynomial
- Eventually, the "spline sampling" method interpolates with splines

Summarizing, these functions are useful in those cases where the perturbation values for example come from measurements whose underlying module is not fully known or cannot be represented by an analytical equation (a gauss distribution, a beta distribution, a combination of gaussian and linear function and so on).

4.14.4.3. Nondeterministic Functions

Common random function implementation with seed management for testing purposes. If seed is set to zero openSF initializes pseudo-randomly the seed (used for non-repeatable executions, montecarlo etc…).

❑ **Beta Distribution**

Generates random values with Beta function as probability density function.

❑ **Gamma Distribution**

Generates random values with Gamma function as probability density function.

❑ **Exponential Distribution**

Generates random values with Exponential function as probability density function.

❑ **Normal Distribution**

Generates random values with Gaussian function as probability density function.

❑ **Uniform Distribution**

Generates random values following a Uniform Distribution.

❑ **Poisson Distribution**

Returns the perturbation as a generated random value with Poisson function as pdf.

❑ **Truncated Gaussian Distribution**

Returns the perturbation as a generated random value with Truncated Gaussian function as pdf.

- **Uniform Discrete Distribution**

Returns the perturbation as a generated random value with Uniform Discrete function as pdf.

- **Distribution with custom Probability Density Function**

Returns the value of a random variable generated with a custom pdf given. It is only recommended to use it by expert developers/scientists.

4.14.4.4. Binary and Composite Operations

Error Generation Libraries implements the basics mathematical operations in binary mode. The operations implemented are:

- **Addition**
- **Subtraction**
- **Multiplication**
- **Division**
- **Exponentiation**
- **Root**

4.15. Monte Carlo studies in openSF

This section provides guidelines to implement a Monte Carlo (MC) analysis in openSF, taking into account current limitations. As a preparation, it is advisable to read sections 4.9.2.10 through 4.9.2.12 and 4.14, which explain the iterative and perturbation capabilities of openSF.

Full support for MC studies is intended in the future, as the current implementation is known to have limitations. With the current status of openSF, MC analysis can be simplified by taking some precautions when developing simulation modules.

Three different approaches are suggested to implement a MC analysis. The best approach depends on the specific constraints of the project and the implementation of the module on which it is desired to perform the MC analysis. These different approaches originate in the fact that, although openSF can perform iterations both in local and global parameters (parametric analyses), it can only introduce perturbations in the local ones (Monte Carlo analyses).

4.15.1. Approach 1: One module MC with local parameter

The first option to implement a MC analysis in openSF considers only one module needs to be executed multiple times. The parameter to be perturbed only needs to be injected in a single MC module. The layout of this approach is shown in the Figure 4-129.

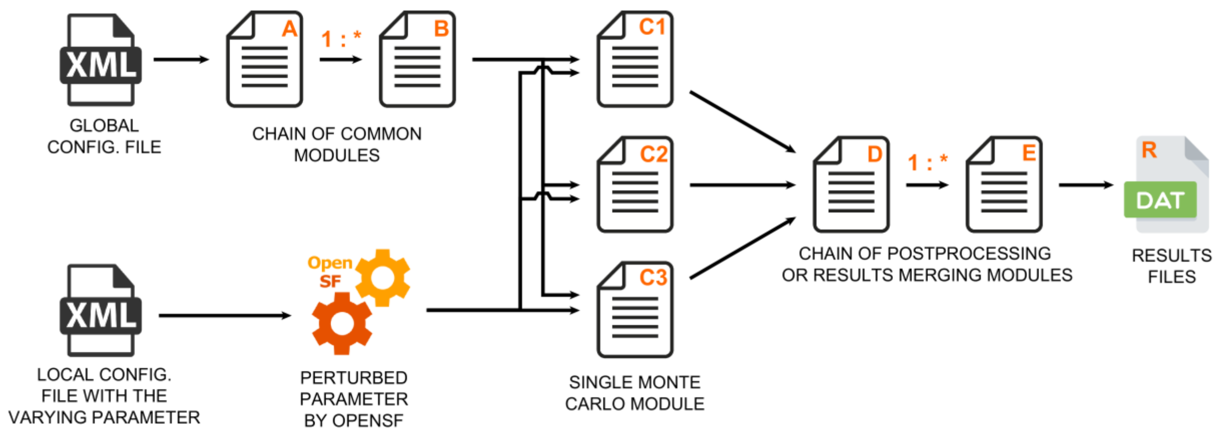

XML
GLOBAL CONFIG. FILE
A
1 : *
B
CHAIN OF COMMON MODULES
C1
C2
C3
D
1 : *
E
CHAIN OF POSTPROCESSING OR RESULTS MERGING MODULES
R
DAT
RESULTS FILES
XML
LOCAL CONFIG. FILE WITH THE VARYING PARAMETER
Open SF
PERTURBED PARAMETER BY OPENSF
SINGLE MONTE CARLO MODULE

Figure 4-129 Monte Carlo chain in Statistical mode

In the most generic version of this approach, the module on which the Monte Carlo will be performed is preceded by a chain of modules and generates results that will feed into another set of modules. These post-processing modules can be used to merge the multiple results of the MC.

The parameter(s) on which the MC is applied must be defined as part of the module's local configuration, with the perturbation specified in openSF through the "Perturbation" menu – see details in the section §4.14 on how to configure the desired number of shots and use the **Statistical** execution mode.

In the chain shown in Figure 4-129, the modules A through B are only executed once, and the same happens with the modules D through E.

Notice that openSF only initiates the execution of a module when all expected inputs are available. Therefore, to build a sequential execution the chain needs to define the inputs of a module as the outputs of the previous one.

To use this approach, the developer must consider the following points. Alternative approaches should be considered in case the following points don't apply to the problem at hand.

- The value of the varying parameter can only be defined in the local configuration file of the MC module.
- openSF passes the same input and output files/folders to all the executions of the MC module. This implies that the MC module knows the varying parameter and dynamically modifies the name of outputs files to avoid overwriting them.
- To correctly process the results of the MC module, downstream modules must be able to read and understand those results. Some degree of agreement is necessary between the MC module and the downstream modules, either by sharing the naming convention used to generate output files and recognise from the names the varying parameter, or by reading the result files and obtaining that information from its contents.
- Considering that openSF initiates the execution of a module when all inputs are available, it is not possible to simply specify a folder as output of the MC module to pass results to downstream modules. After the first MC module terminates, the presence of the folder indicated to openSF that the following module can be started.
  Possible alternative solutions are listed below, knowing that the most applicable depends on the design of the simulator.

- Pass the number of shots to the MC module, and at the end of execution generate a "MC_Completed" flag file in case the expected number of output files is available (i.e. all shots have finished). The "MC_Completed" flag file is essentially used to trigger the execution of downstream modules.
- Pass the number of shots to the first post-processing module. This module is launched as soon as openSF verifies that a results folder exists, and enters an active polling loop until it finds the expected number of files is available.
- In case the modules design/implementation cannot accommodate the above approaches, divide the whole Monte Carlo study in two completely independent simulations. The first simulation executes the modules up to the MC modules, without any post-processing modules; then the second, executes the remaining post-processing modules, manually configured and launched by the user after the completion of the first one.

The options discussed above do not consider handling of errors in MC modules. In those cases, either the modules need to implement error-checking mechanisms, or the user needs to check the correct execution of all shots to ensure reliable results.

4.15.2. Approach 2: Multiple modules MC with local parameter(s)

Another approach has to be used when more than one module is executed with perturbed parameters for each of the Monte Carlo shots. The chain presented in Figure 4-130 shows two main disadvantages: all the modules are executed for each of the MC shots, with the consequent increase in the computation time; and, it is not possible to combine all results generated by the MC modules.

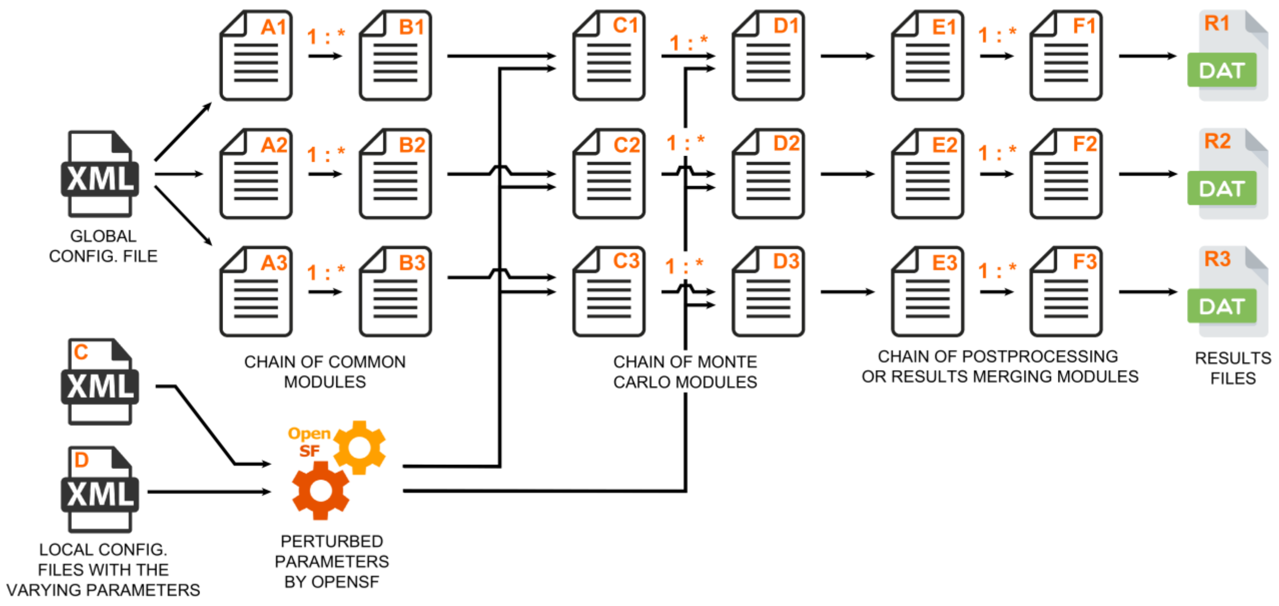

XML
GLOBAL CONFIG. FILE
A1
A2
A3
1 : *
B1
B2
B3
CHAIN OF COMMON MODULES
C1
C2
C3
D1
D2
D3
CHAIN OF MONTE CARLO MODULES
E1
E2
E3
F1
F2
F3
CHAIN OF POSTPROCESSING OR RESULTS MERGING MODULES
R1
R2
R3
DAT
RESULTS FILES
C XML
D XML
LOCAL CONFIG. FILES WITH THE VARYING PARAMETERS
Open SF
PERTURBED PARAMETERS BY OPENSF

Figure 4-130 Monte Carlo chain in Combined mode

In the most generic version of this approach, a chain of modules on which the Monte Carlo will be performed is preceded and followed by a set of normal modules.

As in the previous approach, the parameter(s) affected by the MC must be defined in the local configuration file, with the perturbation specified in openSF through the "Perturbation" menu – see details in the section §4.14 on how to configure the number of shots and use the **Combined** execution mode.

This approach relies in assembling an independent simulation for each perturbation shot. Each shot is considered independently, rerunning common modules, and storing the results in a different simulation folder. In this case, the MC module is not obliged to provide a different name for the result files, although it might be desired in order to gather all the results in a common folder.

The use of the same value of a perturbed parameter in more than one module is supported by simply defining the desired parameter in the local configuration files of the affected modules, and in openSF define the exact same perturbation for all of those parameters.

To keep the simulations executions coherent, all parameter perturbations of the MC modules need to have the same number of shots. The non-perturbed modules will be re-run with the same configuration parameters and the inputs produced by the previous modules of its simulation.

4.15.3. Approach 3: Multiple modules MC with global parameter

In case none of the previous approaches is applicable, or the parameter to vary is not local but global, there is still one additional approach that can be used to implement a Monte Carlo analysis in openSF.

As this approach relies on an external tool to generate the perturbed parameter values, it should only be used when the varying parameter needs to be injected into multiple modules and, either the varying parameter must be defined in the global configuration file, or else no perturbation capabilities are needed.

The typical chain for this approach is shown in Figure 4-131. To setup this approach create a normal simulation in openSF, without taking into account MC. Use the "Iteration" features, as described in the section §4.9.2.10, to customize the global parameter to be perturbed. The perturbed parameter values can be introduced manually, or by providing a file containing the values – this file is typically generated by an external tool. Consider also using a "batch simulation configurator file" as explained in the section 4.9.2.11. Executing the simulation essentially results in a parallel execution for each of the values of the varying parameter.

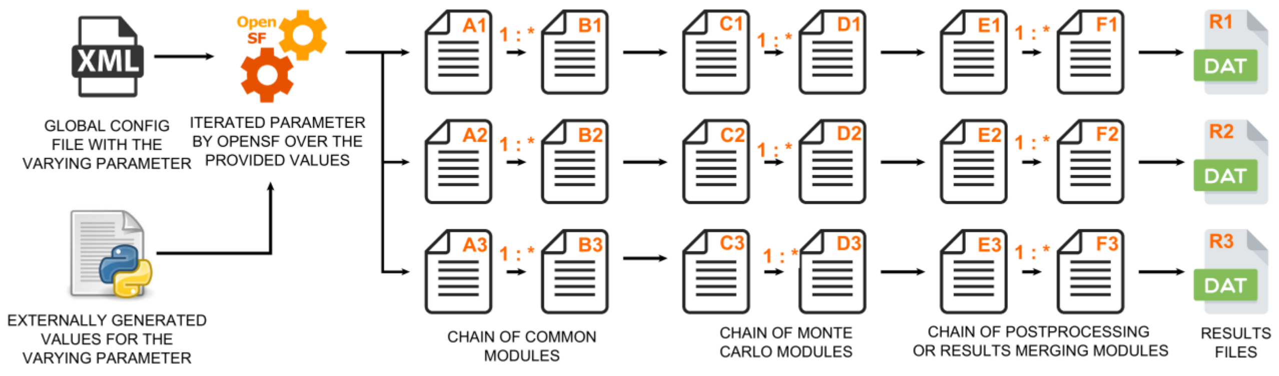

XML
GLOBAL CONFIG FILE WITH THE VARYING PARAMETER
Open SF
ITERATED PARAMETER BY OPENSF OVER THE PROVIDED VALUES
EXTERNALLY GENERATED VALUES FOR THE VARYING PARAMETER
A1
1 : *
B1
C1
D1
E1
F1
R1
DAT
A2
B2
C2
D2
E2
F2
R2
A3
B3
C3
D3
E3
F3
R3
CHAIN OF COMMON MODULES
CHAIN OF MONTE CARLO MODULES
CHAIN OF POSTPROCESSING OR RESULTS MERGING MODULES
RESULTS FILES

Figure 4-131 MC with a global parameter

As the varying parameter is global, all the modules can access it and decide according to its value. As with the previous approach, each shot of the MC is executed as an independent simulation implying that all the common modules are executed repeatedly and that there is supported way to gather/aggregate all the results.

This last limitation could be bypassed by implementing post-processing modules considering the following:

1. Define the total number of shots of the MC in the global configuration file.
2. Considering that the all simulations files are stored as subfolders within the main simulation folder, at the beginning of the module execution, retrieve the simulation folder path and go one level up.
3. Index all the existing folders and their contents.
4. Count the number of MC results and if it is lower than the expected number of shots, end the execution of the post-processing module.
5. When the number of existing results is the same as the expected shots (i.e. when executing the last simulation) perform the post-processing over all the indexed results.

The choice between using this or a similar solution or to simply defining another processing simulation to aggregate the results and manually execute it depends on design decisions particular to each project.

4.16. Time Based Scenario Orchestration

This section provides details regarding openSF time-based scenario orchestration, which is an enhanced ability to customize simulation iteration (section 4.9.2.10).

The time-based scenario execution considers a time-driven execution of a simulation, where a distinct simulation with different parameters is invoked for each time segment in a sequence. An example of a scenario of instrument operational modes that can be defined for a SAR mission is shown in Figure 4-132.

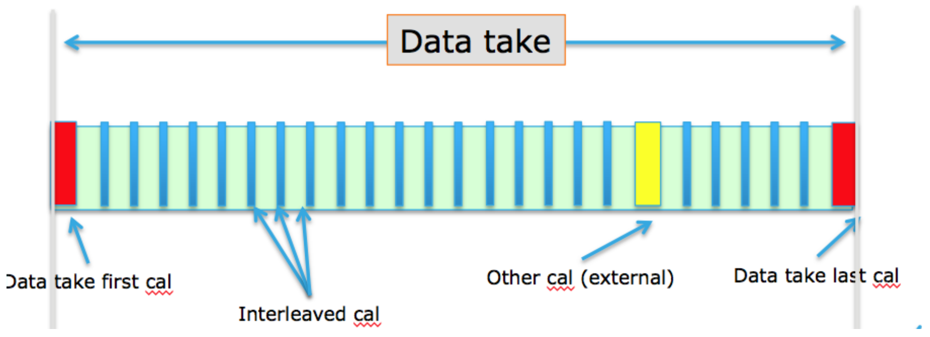
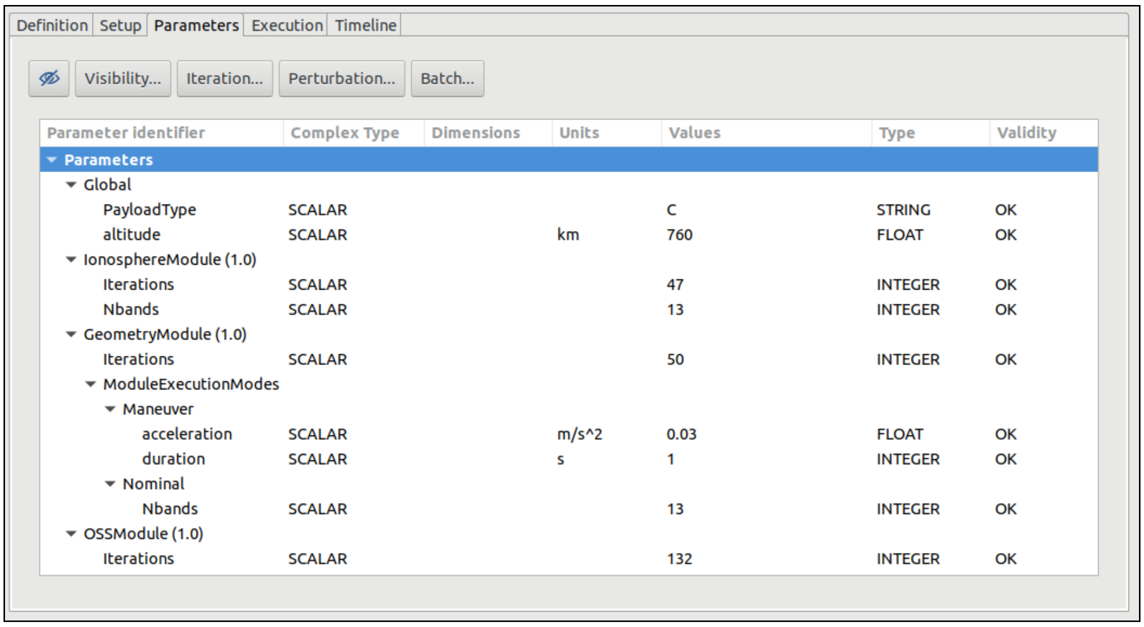


Figure 4-132 Example instrument operational mode scenario

To understand the concepts and definitions supporting the time-based orchestration refers to [AD-E2E].

4.16.1. Time Based orchestration interface

In a simulation editing window, each specific module time mode is rendered as a node in the tree of parameters – see Figure 4-133. This is only for display purposes, i.e., when rendering the parameters in the simulation editing window parameters tree, so it is not reflected in the repository.



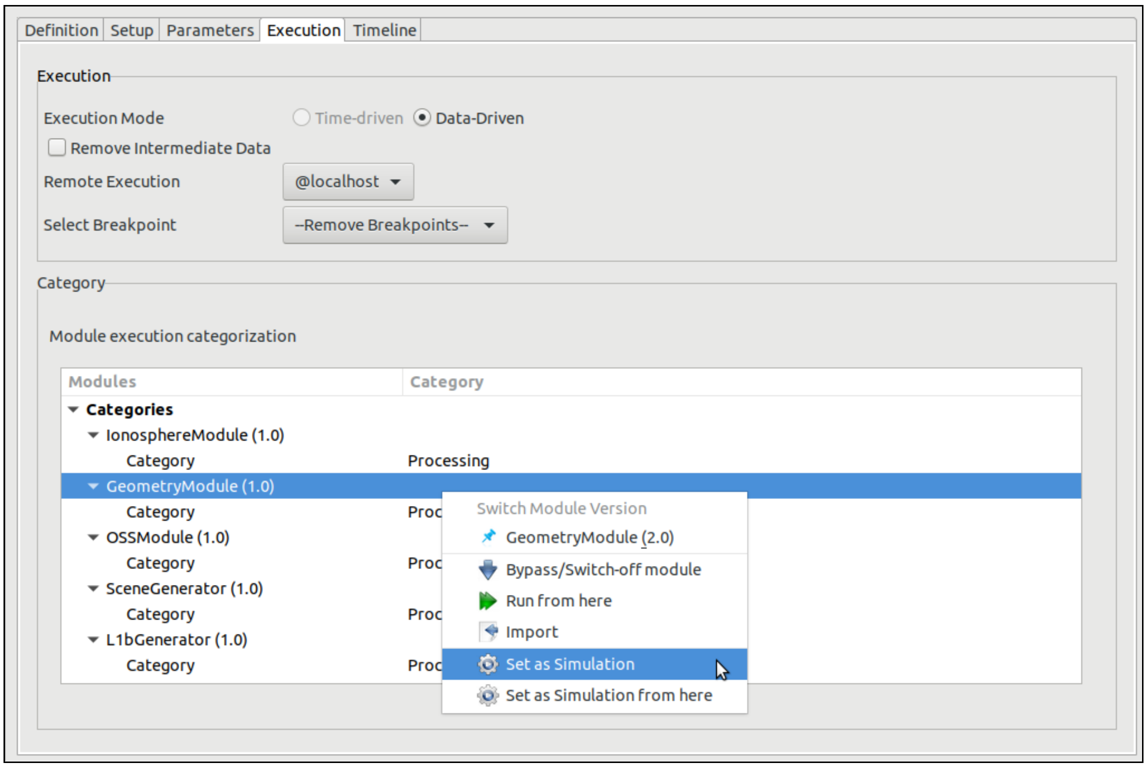
Definition | Setup | Parameters | Execution | Timeline

Visibility... Iteration... Perturbation... Batch...

Parameter identifier	Complex Type	Dimensions	Units	Values	Type	Validity
Parameters						
Global						
PayloadType	SCALAR			C	STRING	OK
altitude	SCALAR		km	760	FLOAT	OK
IonosphereModule (1.0)						
Iterations	SCALAR			47	INTEGER	OK
Nbands	SCALAR			13	INTEGER	OK
GeometryModule (1.0)						
Iterations	SCALAR			50	INTEGER	OK
ModuleExecutionModes						
Maneuver						
acceleration	SCALAR		m/s^2	0.03	FLOAT	OK
duration	SCALAR		s	1	INTEGER	OK
Nominal						
Nbands	SCALAR			13	INTEGER	OK
OSSModule (1.0)						
Iterations	SCALAR			132	INTEGER	OK

Figure 4-133 Module parameters folder organization on a per-mode basis

In the execution tab (Figure 4-134), a simulation radio button indicates whether the execution mode is *time-driven* or *data-driven* based on the availability and configuration of an appropriate timeline (in the timeline Pane). Classification of each module in the processing chain according to Simulation/Processing categories is done (using the context menu) through the Execution pane in the simulation edit window.



Definition | Setup | Parameters | Execution | Timeline

Execution

Execution Mode ○ Time-driven ● Data-Driven
☐ Remove Intermediate Data
Remote Execution @localhost
Select Breakpoint --Remove Breakpoints--

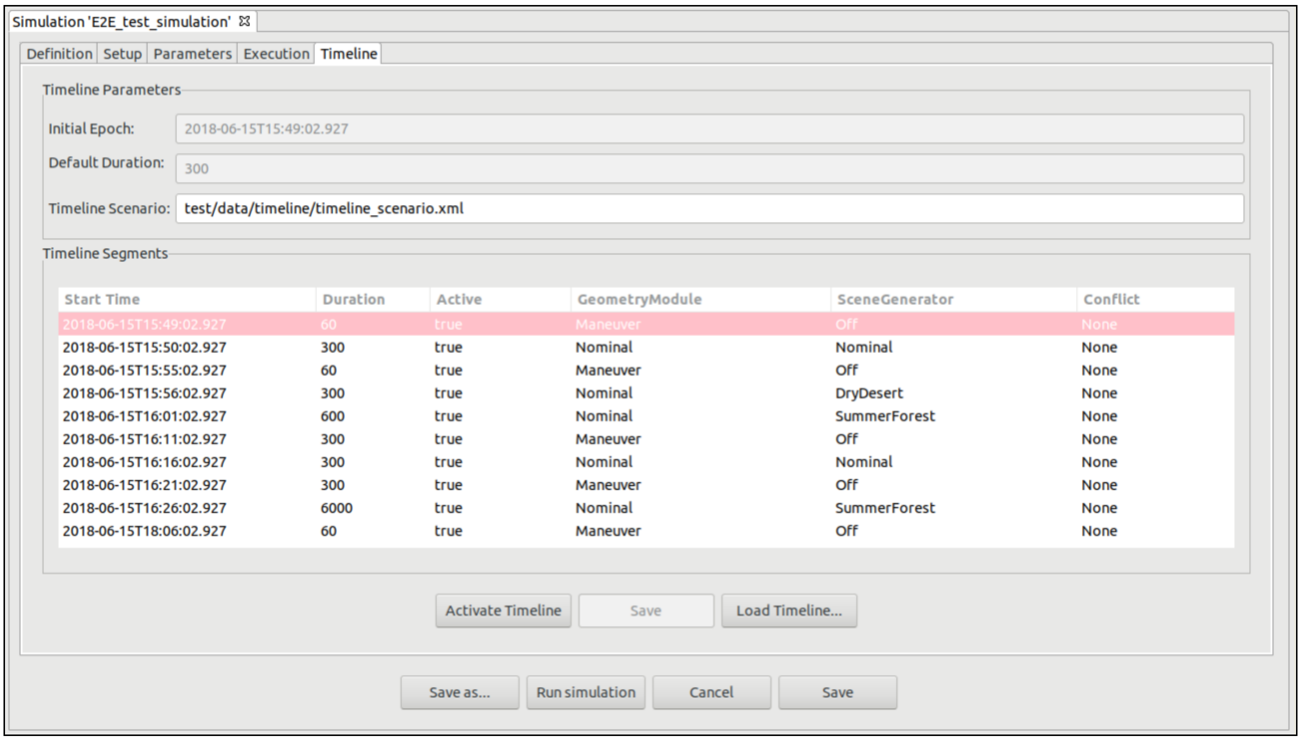
Category

Module execution categorization

Modules	Category
Categories	
IonosphereModule (1.0)	
Category	Processing
GeometryModule (1.0)	
Category	Proc
OSSModule (1.0)	
Category	Proc
SceneGenerator (1.0)	
Category	Proc
L1bGenerator (1.0)	
Category	Proc

Figure 4-134 Module categorization by Mode

The interface for editing the timeline definition is available in the 'Timeline' tab in the Simulation Editing window – see Figure 4-135. This panel allows to define and enable the global timeline parameters and the actual list of time segments to be executed.



Initial Epoch: 2018-06-15T15:49:02.927
Default Duration: 300
Timeline Scenario: test/data/timeline/timeline_scenario.xml

Start Time	Duration	Active	GeometryModule	SceneGenerator	Conflict
2018-06-15T15:49:02.927	60	true	Maneuver	Off	None
2018-06-15T15:50:02.927	300	true	Nominal	Nominal	None
2018-06-15T15:55:02.927	60	true	Maneuver	Off	None
2018-06-15T15:56:02.927	300	true	Nominal	DryDesert	None
2018-06-15T16:01:02.927	600	true	Nominal	SummerForest	None
2018-06-15T16:11:02.927	300	true	Maneuver	Off	None
2018-06-15T16:16:02.927	300	true	Nominal	Nominal	None
2018-06-15T16:21:02.927	300	true	Maneuver	Off	None
2018-06-15T16:26:02.927	6000	true	Nominal	SummerForest	None
2018-06-15T18:06:02.927	60	true	Maneuver	Off	None

Figure 4-135 Timeline management view

The global timeline parameters are: (a) the Initial Epoch, (b) the timeline definition file and (c) the default time segment duration. Timeline segment are displayed below as rows in a table where each column corresponds to the time segments attributes.

It should be noted that the initial epoch parameter is stored in the global configuration file, so this file must have been defined prior to the simulation's Timeline definition. If no previous timeline parameters are present on the global configuration file, openSF will initialize them to the default values. However, they are only saved to the global configuration if the user actually *Activates* the timeline.

To define a Time-Driven simulation, the user can load a previously configured timeline scenario through the *Load Timeline* button, or manually configure one. The timeline scenario file must be compliant with the structure described in [AD-E2E], and the modes referred by the scenario file must match the ones specified defined in the local configuration files of the relevant modules.

If the user chooses to manually configure a custom scenario, a valid file path shall be provided for the global *TimelineFile* parameter. The user can define as many timeline segments as desired, saving the modifications when he is satisfied with the result. It should be noted that if the filename defined for the scenario file does not match any existent file, it is shown in grey and it will only change to black when the user presses the *Save* button and the file is effectively created on disk.

Regardless of whether the scenario file is loaded or generated, it has to be activated with the *Activate Timeline* button before the simulation can be run. To do so, no unsaved modifications can be present. It shall be noted that the timeline activation not only configures the simulation as *Time-Driven* but it also stores the initial epoch timeline parameter in the simulation's Global Configuration File, so it affects to all the simulations that share the same GCF. If, after the timeline scenario activation, any modification is done, the file has to be saved and activated again.

For editing the timeline segments the interface has a set of toolbar buttons to add, remove or duplicate a time segment row. These three actions can be performed either accommodating the existing segments (e.g. add-shift) or without affecting the existing segments. Accommodation is accomplished by adjusting the start times of the segments so that there are neither time segment overlaps nor gaps in the timeline.

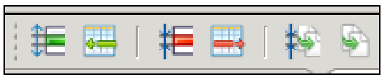
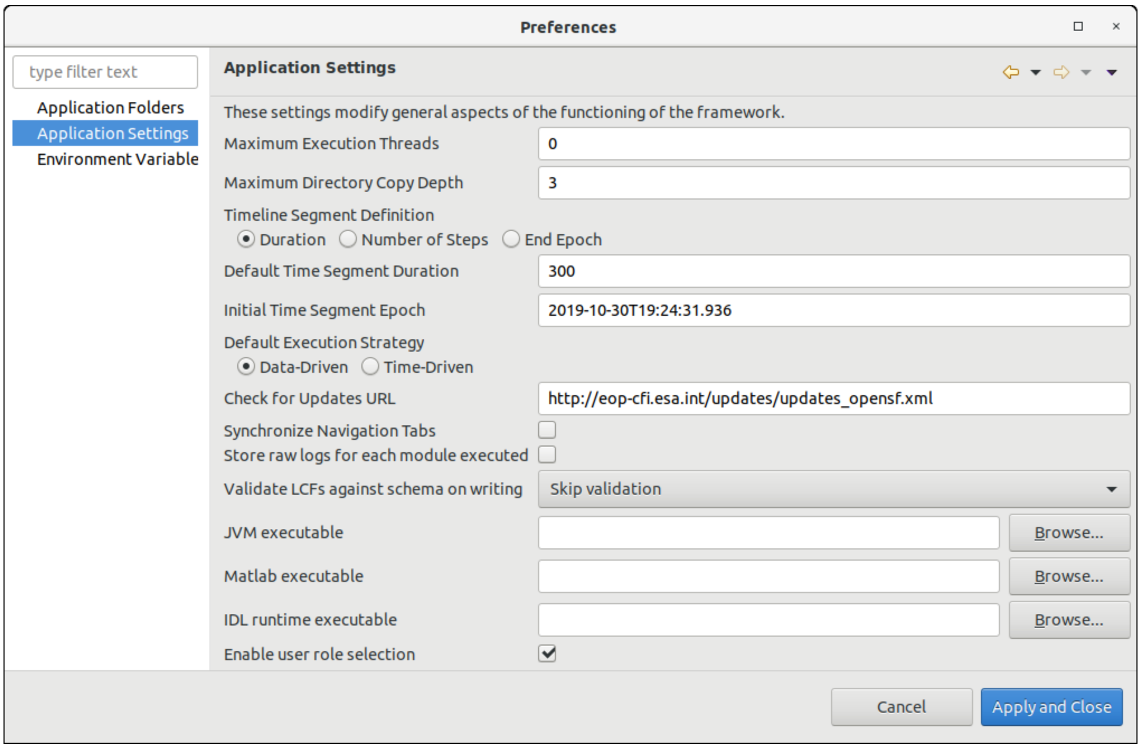


Figure 4-136 Timeline editing toolbar

The addition of each time segment uses default values from the global timeline parameters defined above. Within the timeline table on each row, it is possible to select the mode (active true or false) for each module to be executed during each time segment.

The default representation of time segments in the timeline table can be configured via the *System preferences -> Application Setting -> Timeline visual. This* allows using: *duration*, *number of steps* or *end epoch* as time segment identifier (first column).



Preferences

type filter text
Application Folders
Application Settings
Environment Variable

Application Settings

These settings modify general aspects of the functioning of the framework.

Maximum Execution Threads	0
Maximum Directory Copy Depth	3
Timeline Segment Definition	[✓ Duration] Number of Steps End Epoch
Default Time Segment Duration	300
Initial Time Segment Epoch	2019-10-30T19:24:31.936
Default Execution Strategy	[✓ Data-Driven] Time-Driven
Check for Updates URL	http://eop-cfi.esa.int/updates/updates_opensf.xml
Synchronize Navigation Tabs	☐
Store raw logs for each module executed	☐
Validate LCFs against schema on writing	Skip validation
JVM executable	
Matlab executable	
IDL runtime executable	
Enable user role selection	✓

Cancel | Apply and Close

Figure 4-137 - Timeline preferences

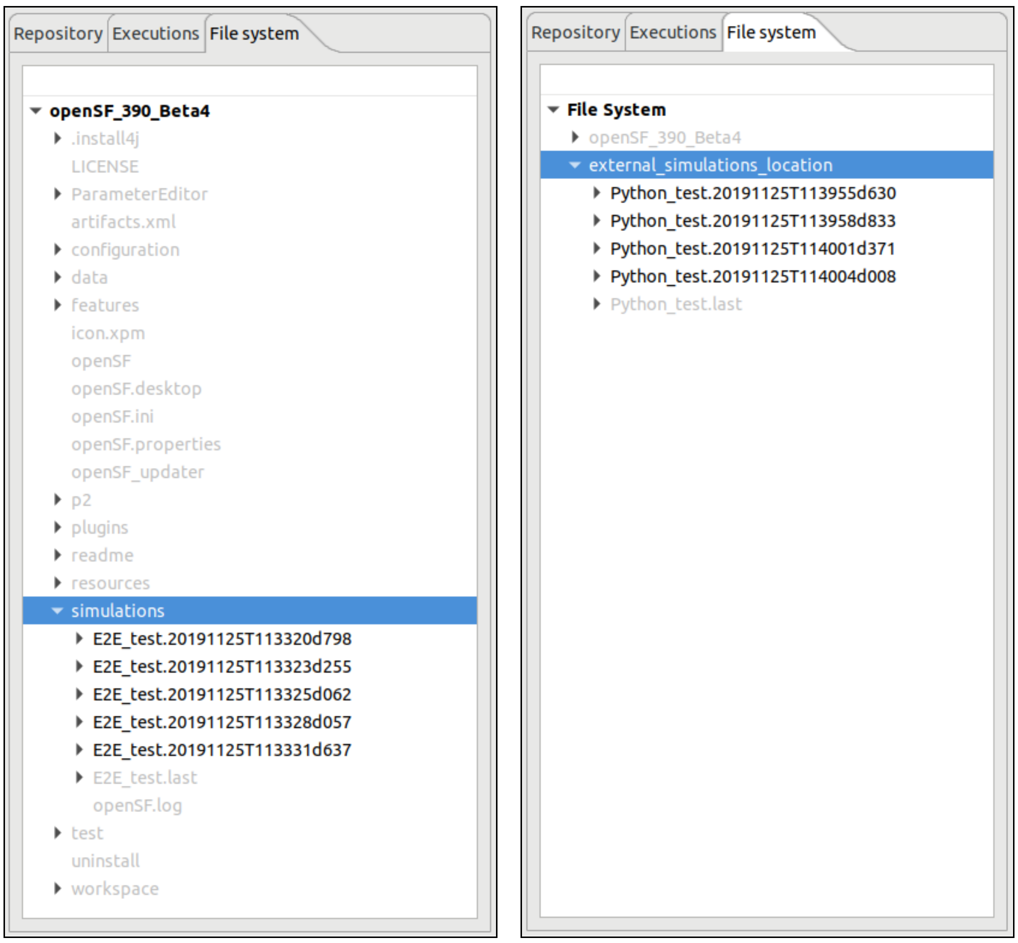
Each timeline segment time definition shall then be introduced by the user with (a) a start time, and (b) one of the following: duration, number of steps or end epoch. If the user inputs duration then from the start time the system can compute the other two alternative values (and so on for any other selection). Then in the timeline configuration file all the four time-related values will be written for each segment. Therefore, when the user switches preference in the global definitions it’s simply changing the "view" over the time data. Being a global configuration, it actually represents the specific user preference for viewing one of the three alternative values. If the simulation is changed or if a given timeline configuration file is supplied (which shall be self-contained) to another user then the displayed column shall be the one corresponding to the users' preference.

Furthermore, the default execution strategy when launching a simulation can be chosen between time-driven and data-driven

4.17. File system

The last tab in the side-bar, named "File system" is a way to easily access both the folder structure under the openSF installation directory and the simulations execution directory.

Organized in the tree-like structure, the user can easily locate every needed file. This structure is refreshed every time an operation involving files is performed or when the user presses the "refresh" button.



```
▾ openSF_390_Beta4
    ▸ .install4j
      LICENSE
    ▸ ParameterEditor
      artifacts.xml
    ▸ configuration
    ▸ data
    ▸ features
      icon.xpm
      openSF
      openSF.desktop
      openSF.ini
      openSF.properties
      openSF_updater
    ▸ p2
    ▸ plugins
    ▸ readme
    ▸ resources
    ▾ simulations
        ▸ E2E_test.20191125T113320d798
        ▸ E2E_test.20191125T113323d255
        ▸ E2E_test.20191125T113325d062
        ▸ E2E_test.20191125T113328d057
        ▸ E2E_test.20191125T113331d637
        ▸ E2E_test.last
          openSF.log
    ▸ test
      uninstall
    ▸ workspace
```

```
▾ File System
    ▸ openSF_390_Beta4
    ▾ external_simulations_location
        ▸ Python_test.20191125T113955d630
        ▸ Python_test.20191125T113958d833
        ▸ Python_test.20191125T114001d371
        ▸ Python_test.20191125T114004d008
        ▸ Python_test.last
```

Figure 4-138: File system view with the simulations directory inside (left) and outside (right) of the the openSF installation directory

The contents displayed in the File System tree change depending whether the simulations executions directory is located inside the openSF installation directory (default behaviour after a clean installation) or outside of it (custom defined by the user from the preferences page, see section 4.5.1).

In the first case, the openSF workspace is added as root node of the File System tree and the selection and focus is set to the location of the simulations directory, expanding as many nodes as required to do so. In the second case, both the openSF workspace and the custom location of the simulations executions directory are added to a dummy "File System" node, expanding and focusing the simulations node as done before. See Figure 4-138.

The File System tree uses a color code to ease finding the resulting files of the executions. Therefore, if a tree node represents a folder containing the results of an execution which is also present in the currently selected database, that node, its parents and all its children are rendered in black.

By contrast, if a tree node represents either a file or a folder which is not part of an execution result, or which has been computed while connected to a different database than the currently selected one, the tree node is rendered in light grey.

4.17.1. Tool execution

As declared in section 4.5.4, the openSF system can associate external tools to a series of file extensions.

In case the user right-clicks over a file name whose extension is associated to one or several product tools, a menu showing some actions will pop-up.

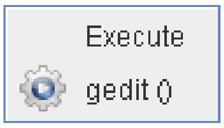


Figure 4-139: IO file pop-up menu

Once the desired action is selected, a dialog will show up asking the user for completing the executable command line. openSF HMI presents the location of the selected file. It presents the absolute path for "external" tools and path relative to $OPENSF_HOME if it is an "internal" tool.

Users can accept the default parameters or can add extra. Users can also make use of the openSF environment variables (described in section 3.5.1) writing the dollar symbol and its name. For example, $ OPENSF_HOME or $DEBUG_MODE.

Once accepted the parameters, the tool program will be executed in a separate thread (so the openSF operations are not interrupted).

4.18. Persistent storage - Database and file system

Most information systems must store information in a persistent way. openSF system trusts in a relational database to store structural information and the file system to store the input/output/configuration files. The following list shows which openSF elements are stored in database and which into the file system.

Element	Storage
System. Configuration	File system. $ OPENSF_HOME /openSF.properties
System. Tools	Database
Repository. Descriptors	Database
Repository. Modules	Database
Repository. Simulations	Database
Repository. Simulation script	File system. <simulations_folder>/<name> /<name>.sh
Executions. Results	Database
Executions. Logs	File system. $OPENSF_HOME /simulations/openSF.log (global) File system: simulation_folder/log/simulation.log (for each simulation) File system: simulation_folder/log/MODULE.log (for each simulation and module, only if enabled in application settings)
Execution. Dumped log simulation	File system
Execution. Input/output/configuration files	File system <simulations_folder>/<name>/<filename>

4.18.1. Database maintenance

Currently, openSF can connect to a local or remote MySQL server (see section 4.5.2). If the last database to which openSF was connected is not accessible during openSF initialization, it will try to connect by default to a database allocated in a local MySQL server, named “openSF” with a user named “openSF” with password “openSF”.

The user (or the database server administrator) is responsible to regularly back up, de-fragment, clean and perform similar maintenance operations to guarantee the database integrity. Users can execute the following script to perform a manual backup to the openSF database.

```
~/openSF$ mysqldump --user=openSF --password=openSF openSF > openSFdb.bk.sql
```

In case of a major corruption problem it is possible for the user to directly create a new empty database by using the “openSFdb.sql” script included in $OPENSF_HOME this way:

```
~/openSF$ mysql --database=openSF --user=openSF --password=openSF <
data/openSFdb.sql
```

4.19. Table of keyboard shortcuts

This section describes the list of “acceleration keys” accessible for the user. These combinations of keys, when pressed, access the desired platform functionality without using the mouse and menu systems.

Here is the complete list:

Module	Action	Acceleration key
System	Exit the system	Control + x
System	Exit the system	Alt + F4

4.20. Error messages

openSF platform controls its correct behaviour with an error handling system. Users are informed about the nature of the error and a possible way to correct it.

Here is a list of different kinds of raised errors:

Module	**Operation**	**Error**	**Comments**
System. Configuration	Adding a new variable	Validation error	Follow the instructions to correct the value
System. Tools	Accepting changes	Tool addition failed	The user has chosen a duplicated identifier. Please provide a different identifier
		Validation error	Follow the instructions to correct the value
	Deleting a tool	Database error	Possible database failure. Is the database running?
	Executing a tool	File IO error	Follow the instructions
Repository. Descriptors	Accepting changes	Descriptor modification failed	Possible database failure. Is the database running?
		Descriptor addition failed	Duplicated identifier chosen. Please provide another identifier.
		Validation error	Follow the instructions to correct the value
	Adding an IO file	Validation error	Follow the instructions to correct the value
		A descriptor shall not have associated two files with the same id	Please choose another identifier

Module	**Operation**	**Error**	**Comments**
	Deleting a descriptor	Database error	Possible database failure. Is the database running?
Repository. Modules	Accepting changes	Validation error	Follow the instructions to correct the value
		Module addition failed	Duplicated identifier. Please provide a different identifier
	Deleting a module	Database error	Possible database failure. Is the database running and configured?
	Creating a new version	Database error	Possible database failure. Is the database running and configured?
Repository. Results	Deleting a result	Database error	Possible database failure. Is the database running and configured?
Repository. Simulations	Accepting changes	Validation error	Follow the instructions to correct the value
		Database error	Possible database failure. Is the database running and configured?
		Simulation addition failed	Duplicated identifier. Please provide a different identifier
	Adding a simulation	Simulation identifier cannot be void	Please provide a valid identifier before adding a simulation
	Deleting a simulation	Database error	Possible database failure. Is the database running and configured?
	Generating a script	File IO error	Follow the instructions
	Iterating parameters	Invalid list of values	Please input a comma-separated list of valid values (no blanks)
		Validation error	Follow the instructions to correct the value
	Removing a simulation	There is no simulation selected	Please select a simulation to remove
	Running a simulation	Cannot run an unnamed simulation	Please input a valid identification to the simulation

Module	Operation	Error	Comments
		Missing configuration files	Provide the missing GCF or LCFs in order to run the simulation
		Validation error	Follow the instructions to correct the value
		File IO Error	Follow the instructions
Repository. Simulations	Accepting changes	Simulation modification failed	Possible database failure. Is the database running and configured?
		Simulation addition failed	Duplicated identifier. Please provide a different identifier
		Validation error	Follow the instructions to correct the value
	Deleting a simulation	Database error	Possible database failure. Is the database running and configured?
	Executing a simulation	Validation error	Follow the instructions to correct the value
	Setting limits	Validation error	Follow the instructions to correct the value
Executions. Log	Dumping the log	File IO error	Follow the instructions

In general, every time an input value is needed, the platform will perform a validation process. If the input does not comply with the needed format, the user will be informed with a self-explained message.

Errors not shown as part of the graphical interface are not controlled messages. They correspond to messages from the standard output or error stream.

When executing a simulation, modules raise their own error messages and they are intercepted by the system and shown as log messages in the execution view.

5. TUTORIAL: CREATING AN E2E SIMULATION

This chapter will show user how to create and end-to-end simulation within openSF software. The simulation chain used is the one installed as validation scenario during openSF deployment.

This chapter is divided in the following sections:

- **Scenario Description**, showing the outline of the E2E simulation, logic entities, input and output identification, etc…
- **Module Development Guidelines** detailing the module development process aimed to be integrated in openSF.
- **Framework Structure Definition**, which details the steps that shall be taken in order to create a whole simulation scenario within openSF HMI. This section also gives some guidelines about the recommended folder structure for a simulation project that will be integrated within openSF.
- **Product Tools Specification**, including the definition of data exploitation tools.

This tutorial should be complemented by the information in the openSF Training course material [RD-TM] available on the openSF web page (https://eop-cfi.esa.int/index.php/openSF).

5.1. Scenario Description

The outline of sample of a test simulation scenario is shown in Figure 5-1. The drawing of this diagram is the first step to integrate a simulation scenario within openSF.

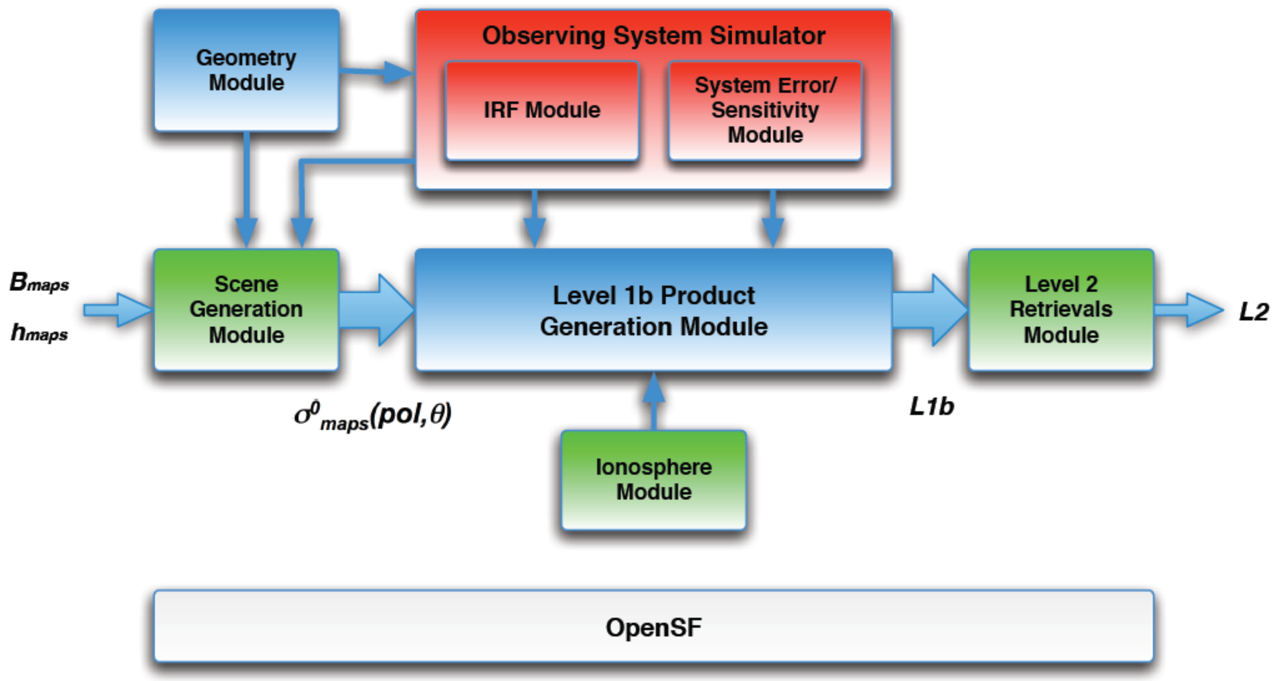

Geometry Module
Observing System Simulator
IRF Module
System Error/ Sensitivity Module
B_{maps}
h_{maps}
Scene Generation Module
Level 1b Product Generation Module
Level 2 Retrievals Module
L2
$\sigma^0_{maps}(pol,\theta)$
L1b
Ionosphere Module
OpenSF

Figure 5-1: Outline of a test simulation scenario

After this point users shall identify and define the openSF entities that will be part of the simulation scenario. The entities that take part in this tutorial E2E simulation are listed in the following sections.

5.1.1. Descriptors - Input and Output Files

The definition of the descriptors (file sets) shall be done together with the module definition as input and files generated are the interfaces for simulation modules. This is described in the [AD-ICD].

The files showed in this section have been extracted from the validation test data set. These can be found in the openSF installation test folder. See section 0. Note that even though they are XML files, they are input/output files and *not* configuration files as defined in AD-E2E, and therefore they do not use the same format.

- ❑ **Input_Ionosphere**: input used for the Ionosphere module.
  - *InputIonos.xml*
- ❑ **Product_Ionosphere**: file generated by the Ionosphere module.
  - *Ionosphere.xml*
- ❑ **Input_Geometry**, input used for the Geometry computation module.
  - *InputGeo.xml*

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<openSFProduct>
  <InputGeo>
    <temperature type="FLOAT" value="30"/>
    <humidity type="FLOAT" value="56"/>
  </InputGeo>
</openSFProduct>
```

Figure 5-2: Product File Example

- ❑ **Product_Geometry**, file generated by the Geometry computation module.
  - *Geometry.xml*
- ❑ **Product_OSS**, file generated by the Observing System module.
  - *Instruments.xml*
- ❑ **Input_Scene**, input used for the Scene Generator, it is composed by a map input file and the outputs of the Geometry module and the Observing System module.
  - *Maps.xml*
  - *Geometry.xml*
  - *Instruments.xml*
- ❑ **Product_Scene**, output from the Scene Generator module.
  - *Scene.xml*
- ❑ **Input_L1b**, input used for the L1b processor composed by the outputs coming from Scene Generator, Observing System and Ionosphere modules.

- *Scene.xml*
- *Instruments.xml*
- *Ionosphere.xml*

❑ **Product_L1b**, descriptor that represents the level 1b simulated product.

- *L1b.xml*

❑ **Product_L2**, descriptor that represents the level 2 simulated product. This file can be considered as the global simulation output.

- *L2.xml*

5.1.2. Modules

This is the list of modules identified within the test simulation chain. Each module has associated an input and output description to allow a proper orchestration of the simulation scenario. Other module configuration items such as executable file and XML configuration file are described in [AD-ICD].

❑ **IonosphereModule**

- Input descriptor: **Input_Ionosphere**
- Output Descriptor: **Product_Ionosphere**

❑ **GeometryModule**

- Input descriptor: **Input_Geometry**
- Output Descriptor: **Product_Geometry**

❑ **OSSModule**

- Input descriptor: **Product_Geometry**
- Output Descriptor: **Product_OSS**

❑ **SceneGenerator**

- Input descriptor: **Input_Scene**
- Output Descriptor: **Product_Scene**

❑ **L1bGenerator**

- Input descriptor: **Input_L1b**
- Output Descriptor: **Product_L1b**

❑ **L2Retrieval**

- Input descriptor: **Product_L1b**
- Output Descriptor: **Product_L2**

© DEIMOS Space S.L.U, 2019

5.2. Module Development Guidelines

The module development process is described in [AD-ICD] in section 4.8. Refer also to section 4.9.6.2 for the Precautions for module developers to ensure safe module parallelisation.

5.3. Framework Structure Definition

This task consists in defining all the entities specified in section 5.1 into the openSF HMI. This can be performed following the instructions specified in the openSF reference manual, chapter 4 of this document. Furthermore, a step by step example of the creation of this simulation in the openSF HMI can be found in the openSF Training course material [RD-TM] available on the openSF web page (https://eop-cfi.esa.int/index.php/openSF).

5.3.1. Folder Structure Guidelines

This section gives some tips and recommendations about the folder structure within a simulation project that is integrated into openSF. This section is aimed to ease the integration process. As mentioned, the following instructions are not mandatory as users can choose whatever structure they.

Simulation Project Structure

- ❑ *modules* folder where all files regarding to the simulation algorithms are stored including executable, configuration and input files
  - *src* for modules source code
  - *bin* for modules binaries
  - *lib* for the libraries (example libProducts.dll for input output management that can be common to all modules within the simulation chain)
  - *cots* folder for storing third party applications and libraries used within the modules
  - *test* folder for system and unit tests
  - *data*
    - ➢ *conf* for global and module configuration files
    - ➢ *input* for filed used as input
- ❑ *tools* folder where store source code and related files for project specific tools (ex: end-to-end comparator)
  - *bin* for tools binary files
  - *lib* for tools library files
  - *src* for tools source files
- ❑ *doc* folder where useful documentation of the project can be located

openSF team recommends storing all the data regarding to the project in folders using OPENSF_HOME as root directory or a subfolder of it. Example: */home/tester/openSF/E2Etutorial/* being */home/tester/openSF/* the openSF home folder. This will help in the framework integration process as relative paths to the files can be used. The resultant folder structure for the E2E tutorial presented in this chapter is shown in Figure 5-3.

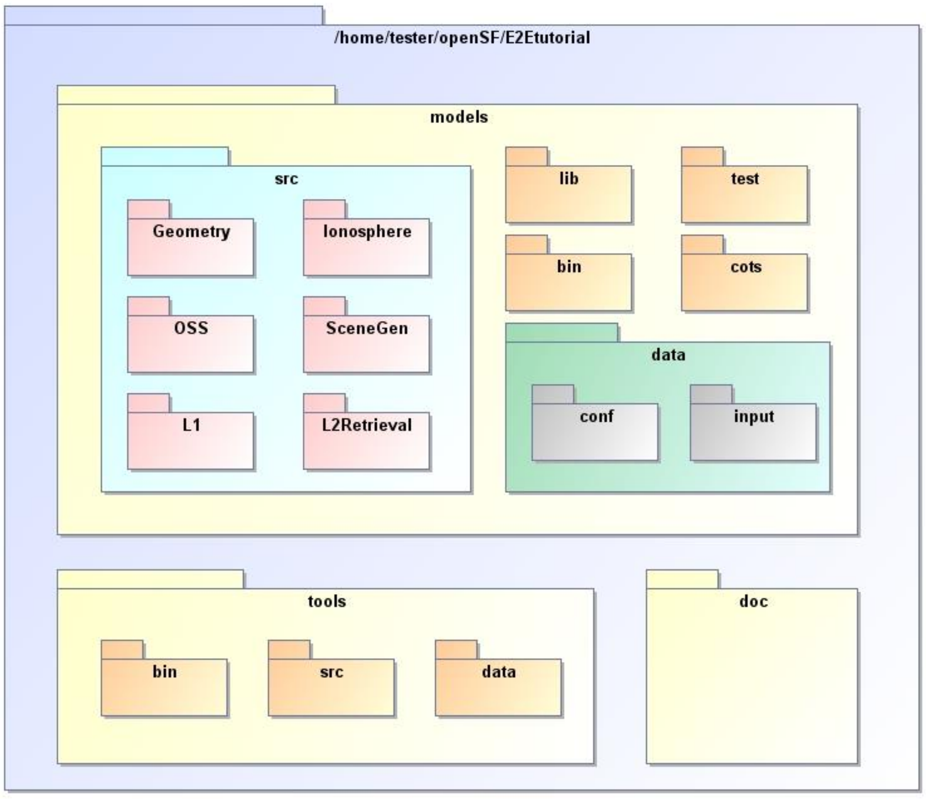


Figure 5-3: E2E Tutorial folder structure

5.4. Product Tools Specification

The definition of product tool is detailed in section 4.5.4; it is recommended that users take a look to this section before going on reading.

5.4.1. Simulation Products Exploitation

Following the mechanism described in section 4.5.4 openSF users are able to plug-in any product tool in order to visualize, post-process or archive the simulation products.

The election of this product tools depends on the type of simulation products (definitively files) users want to analyse. A list of popular product tools used in openSF related simulation projects is shown in section 4.5.4.5.

In the case of the test simulation scenario where all product files are XML and the tool associated can be the user-preferred text editor (Notepad, Gedit, Emacs etc…)

5.4.2. Closing the Loop in an E2E Simulation

Usually the target of performing and E2E simulation is to validate the output of a set of algorithms comparing the input and the output of the simulation. Other objectives can be the sensitivity and stability analysis of a full processing chain over a set of simulation parameters.

In any of the mentioned cases it is necessary to perform a comparison between two points of the simulation chain in order to analyse the results. This connection ***closes the loop*** within a simulation scenario.

In order to close the loop in openSF, users can follow the following strategies depending on the simulation scenario.

- ❑ Development of a product specific processing tool.
- ❑ Development of a new module and insert it into the simulation chain as a new processing stage.
- ❑ Use of a cots comparison tool. This mechanism is recommended when the product format can be compared directly without any pre-processing step.

For the tutorial scenario possible points to close the loop are the Scene input (*Scene.xml*) and the L1b or L2 product (*L1b.xml* or *L2.xml*). This action would require the development of a product specific tool that performs a simple processing of the Scene input in order to compare with the L1b or L2 products. This situation can be also solved including a new module in the simulator where this comparison is performed.

6. ANNEX A: INSTRUCTIONS TO BUILD THE FRAMEWORK

This annex explains how the openSF framework is built. This section is oriented only for developers that need to build openSF from sources due to project specific customizations.

The openSF development team recommends the use of Eclipse IDE as it is the platform used for developing the framework. It is just a recommendation, as the platform uses Maven as a build system and thus can be built directly from the command line, or with another Java IDEs such as NetBeans.

6.1. Pre-requisites to Build the Framework

- openSF source files
- Java Development Kit, version 8 or later
- Apache Maven tool, version 3 or later
- If tests are to be executed, MySQL database to execute and test the framework.
- If installer packages are to be generated, install4j with a valid license, version 5.1.

6.2. How to Build the openSF Platform

Apache Maven is a Java-based build system that uses Project Object Model (POM) files to orchestrate compilation and packaging of applications. Maven is able to automatically pull dependencies from the Internet, a functionality that is used to download the Eclipse RCP Java files and native launchers.

Due to the above, building openSF requires a connection to the Internet, at least the first time that the build is attempted: This is necessary in order for Maven to download its own plugins, including the Tycho system that builds Eclipse RCP applications, and the Eclipse runtime files.

Once unpacked, the openSF tree contains the following relevant files and folders:

- `build.sh`: support script performing most of the Maven build steps.
- `openSF.build`: folder containing the majority of the "release engineering" architecture. In particular, the main POM file that other files in the project reference.
- `openSF.build/generate-installers.py`: support script that gathers the files for each platform into the right structure and calls install4j to generate the installer packages.
- `openSF.build/dependencies-mvn2osgi`: folder containing a separate POM project that needs to be processed first and separately, due to a technical limitation of Tycho/Maven.
- `openSF.build/installers`: folder containing install4j project files in order to generate installation packages for openSF (see §6.3). Also contains a folder to place the external packages to be bundled in, like documentation files, example modules, ParameterEditor, etc.
- `openSF.product`: folder where the output of the build will be generated (under "target").
- `platform`: folder containing the main source code for the openSF framework.
- `platform.tests`: folder containing unit tests for the openSF platform.

The output of this step is a series of "Eclipse packaged product" ZIP files, one for each platform, available at the `openSF.product/target/products/` folder. Note that a single build in one machine generates the files for all platforms[11], since the platform-dependent components are downloaded from Eclipse and don't need to be built.

These files are not full installations of openSF: they are only one of the multiple components that are needed to build a fully-functional installer package, as described in §6.3. However, they *can* be used as-is during development and testing, if they are unpacked on top of an *existing* openSF installation, they will in effect "upgrade" that installation.

6.2.1. Simplified Procedure

Using the support script provided, it is easy to build the openSF platform files

```
$ ./build.sh
```

The support script runs the two-step build process outlined in the following simulation, accepting two environment variables that change its behaviour:

- ❑ `MVN`: path to the Maven executable, defaults to "mvn", so Maven is expected to be available in the system PATH.
- ❑ `SKIP_TESTS`: if defined to "1", skips the phase in which the openSF unit tests are run (the Maven target used is "package" instead of the next step "verify").

6.2.2. Detailed Procedure

Any build configuration changes before the process can be performed on the various files that define the project settings. For example, the file `openSF.build/pom.xml` defines the system platforms to build for; the file `platform/plugin.properties` configures some strings in the program, etc.

In order to build the platform, first the project under `openSF.build/dependencies-mvn2osgi` must be built and installed (in the local Maven repo). This step is required due to a limitation of the Maven/Tycho build, and only needs to be performed once as long as this project does not change. It makes available some OSGi-enabled JARs for the second project to find and use.

```
$ mvn -f openSF.build/dependencies-mvn2osgi/pom.xml clean install
```

After the dependencies project is installed for Maven to find its outputs, the main project can be built:

```
$ mvn -f openSF.build/pom.xml clean verify
```

The "verify" target builds the product and runs unit tests. If such tests fail, the build stops immediately and the product files are not generated. If such behaviour is not desired, there are two different options:

- ❑ Passing the `-fae` ("fail at end") flag to Maven. In this case, the test failure will be noted and the build will be reported as "failed", but the product files will still be generated.
- ❑ Using the "`package`" target instead of "`verify`", which skips the execution of the unit tests.

[11] However, if building on Windows, the fact that the Windows file system does not save the "executable" bit will mean that the launchers in the generated Linux and Mac OSX product files generated will require a later `chmod +x`.

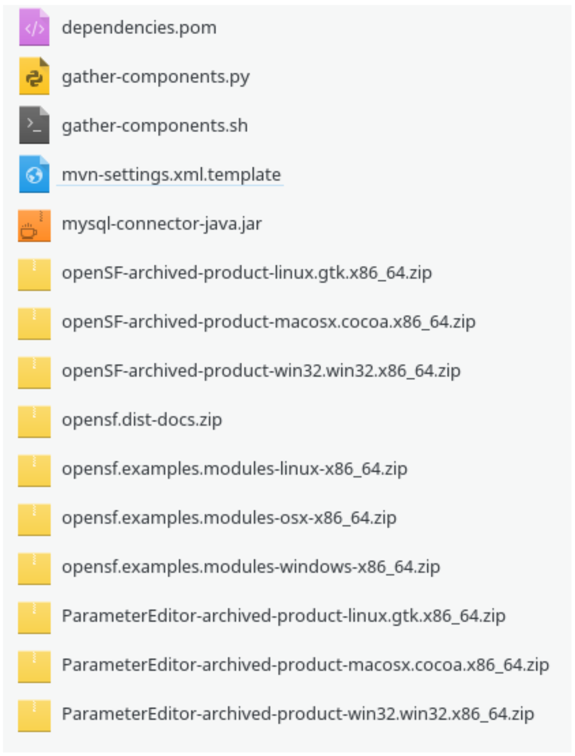
6.3. How to Build the Installer Packages

As mentioned before, the output of the last step is a series of ZIP files in the Maven-generated "target" folder under `openSF.product`. They contain the "archived" installations of the RCP products for each platform, which can be used during development as mentioned in §6.2. However, a working openSF installation is made up of several components, which if missing will make the system work partially, or not at all. Those components are:

- Documentation files, the set of PDFs shown in the help menu. Without these files, no help files will be shown at all in the HMI.
- ParameterEditor, which is built separately as another RCP archived product, but with the same procedure described for openSF. If this platform-specific component missing, openSF will not be able to launch PE as an external configuration file editor.
- Example modules and database, which is a set of dummy modules, configuration and input files, along with a database with simulations including them. Since they include C++ modules along with Java and Python examples, they are platform specific, built using CMake and require linking against the OSFI library. If missing, only the "empty" database template file is available in the HMI, but no openSF functionality is lost.
- Template openSF.properties file, placed by the installer in the installation folder after some variables have been replaced. Without this file, openSF might not start.

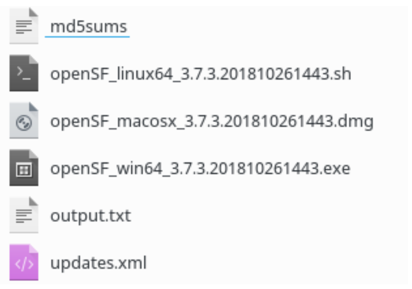
In order to generate the installer, the `openSF.build/generate-installers.py` script can be used. It requires the other components to be present at `openSF.build/installers/external`. The user can place them there manually, or, if these outputs are uploaded to some internal Maven repository (e.g. as part of some continuous integration build system), the `dependencies.pom` and `gather-components.py` files in that same folder are designed to download them from such a repository automatically. In either case, the layout should look like Figure 6-1.

Finally, if install4j is available, running the installer generation script will create files similar to those displayed in Figure 6-2. The script accepts settings through several environment variables including the product version and installer signing capabilities; see the script itself for details.



dependencies.pom
gather-components.py
gather-components.sh
mvn-settings.xml.template
mysql-connector-java.jar
openSF-archived-product-linux.gtk.x86_64.zip
openSF-archived-product-macosx.cocoa.x86_64.zip
openSF-archived-product-win32.win32.x86_64.zip
opensf.dist-docs.zip
opensf.examples.modules-linux-x86_64.zip
opensf.examples.modules-osx-x86_64.zip
opensf.examples.modules-windows-x86_64.zip
ParameterEditor-archived-product-linux.gtk.x86_64.zip
ParameterEditor-archived-product-macosx.cocoa.x86_64.zip
ParameterEditor-archived-product-win32.win32.x86_64.zip

Figure 6-1: External components



md5sums
openSF_linux64_3.7.3.201810261443.sh
openSF_macosx_3.7.3.201810261443.dmg
openSF_win64_3.7.3.201810261443.exe
output.txt
updates.xml

Figure 6-2: Generated installers (one release and one development build)

END OF DOCUMENT

© DEIMOS Space S.L.U, 2019