SYSTEM USER MANUAL

Open Simulation Framework

openSF ParameterEditor

Code:	OPENSF-DMS-PE-SUM
Issue:	1.0
Approval Date:	15/12/2017
Confidentiality Level:	Unclassified

Prepared by: Gonzalo Vicario/Project Engineer

Reviewed by: Federico Letterio/Project Engineer

Approved by: Antonio Gutierrez/Project Manager

Approval Signature:

This page intentionally left blank

Document Status Log

Issue	Section	Change Description	Date
1.0	All	First issue of this document. Content extracted from [RD 4] and Updated with HMI revamping for Eclipse RCP.	15/12/2017

Table of Contents

List of Tables

1. INTRODUCTION

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 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 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

The openSF ParameterEditor (PE) is a software application that adds extra functionalities to the simulation framework. This software has been developed in the framework of the Sentinel 3 Optical System Performance Simulator contract (Thales Alenia Space France), it is an on-going activity so the application could be subject of updates, bug correction and minor changes in short term. This software is included in openSF default installation since version 2.0.

1.1. Purpose

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

The objective of this document is to provide a clear description of the functionalities of the ParameterEditor.

The intended readerships for this document are openSF users.

1.2. Scope

This document applies to PE v1.8 and openSF v3.7.1 and contains:

- Section 1 talks about the document, giving a description and settling the basis to understand it.
- Section 2 links this document with information from other sources.
- Section 3 explains briefly the functionalities of ParameterEditor and how to start it.
- Section 4 describes one by one all the different functionalities of the ParameterEditor.

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

1.3. Acronyms and Abbreviations

The acronyms and abbreviations used in this document are the following ones:

- **AD:** Applicable Document
- **API**: Application Programming Interface
- **COTS:** Commercial Off-The-Shelf
- **DMS:** DEIMOS Space
- **E2E:** End to end simulation

- **GUI:** Graphical User Interface
- **HMI:** Human-Machine Interface
- **HW:** Hardware
- **I/F:** Interface
- **I/O:** Input/Output
- **ICD:** Interface Control Document
- **IT:** Integration Test
- **HMI:** Human-Machine Interface
- **RD:** Reference Document
- **RDBMS:** Relational Data Base Management System
- **SEPSO:** Statistical End-To-End Performance Simulator for Optical Imaging Sensors
- **SUM:** System User Manual

1.4. Definitions

The definitions of the specific terms used in this document are the following ones:

- **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.
- **Configuration File**: A 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.
- **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, and not all of them shall be accessible from the HMI.

2. RELATED DOCUMENTS

2.1. Applicable Documents

The following table specifies the applicable documents that shall be complied with during project development.

Table 1: Applicable documents

Reference	Code	Title	Issue
[AD 1]	openSF-DMS-ICD-001	openSF Interface Control Document	3.0
[AD 2]	openSF-DMS-ADD-001	openSF Architecture Design Document	2.2
[AD 3]	PE-ID-ESA-GS-464	ESA generic E2E simulator Interface Control Document	1.2.2

2.2. Reference Documents

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

Table 2: Reference documents

Reference	Code	Title	Issue
[RD 1]	OSFI-DMS-TEC-DM	openSF Integration Libraries Developers Manual	1.14
[RD 2]	OSFEG-DMS-TEC-DM	openSF Error Generation Libraries Developers Manual	1.2
[RD 3]	openSF-3.2-Training	openSF Training Course	3.1
[RD 4]	OPENSF-DMS-SUM-001	openSF System User's Manual	3.12
[RD 5]	https://www.eclipse.org/projects/project-plan.php?planurl=http://www.eclipse.org/eclipse/development/plans/eclipse_project_plan_4_6.xml#target_environments	Eclipse Target Environments	-

2.3. Standards

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

Table 3: Standards

Reference	Code	Title	Issue
[STD 1]	ECSS-E-40C	Software development Standard	-
[STD 2]	(www.w3.org/TR/xml11/)	Extensible Markup Language (XML) 1.1	-

3. GETTING STARTED

3.1. Introduction

The ParameterEditor is not only a graphical front-end for creating and editing the parameters involved in a simulation but also an interface to introduce constraints between parameters defined in different configuration files.

The new functionalities provided by this tool are the following:

- User-friendly parameters interface allowing the creation, edition and deletion of them avoiding the XML text editing.
- Enhanced consistency checking of parameters. It includes range, type and dimension check.
- Enhanced editing with excel-like interface for vectors/matrix parameters and plot capabilities.
- Built-in XML text editor with syntax and errors highlighting.
- Interface for rules and constraint definition connecting parameters that can be located in different configuration files.

This last functionality is the most complex and most powerful of the ParameterEditor system, all details regarding it are explained in section 4.7.

3.2. System Requirements

3.2.1. Hardware requirements

Hardware must at least fulfil the following requirements:

- 64-bit processor

3.2.2. Operating system requirements

- Linux. Kernel version 3.16 or higher is recommended.
- Mac OSX, version 10.8 (Mountain Lion) or higher.

3.2.3. Framework pre-requisites

The framework pre-requisites are imposed by the development platform Eclipse and they apply to the JRE installed and the windowing system of the platform detailed in reference [RD 5].

3.3. How to Start the Application

The ParameterEditor can be launched from openSF clicking in the icon with the greek letter π.

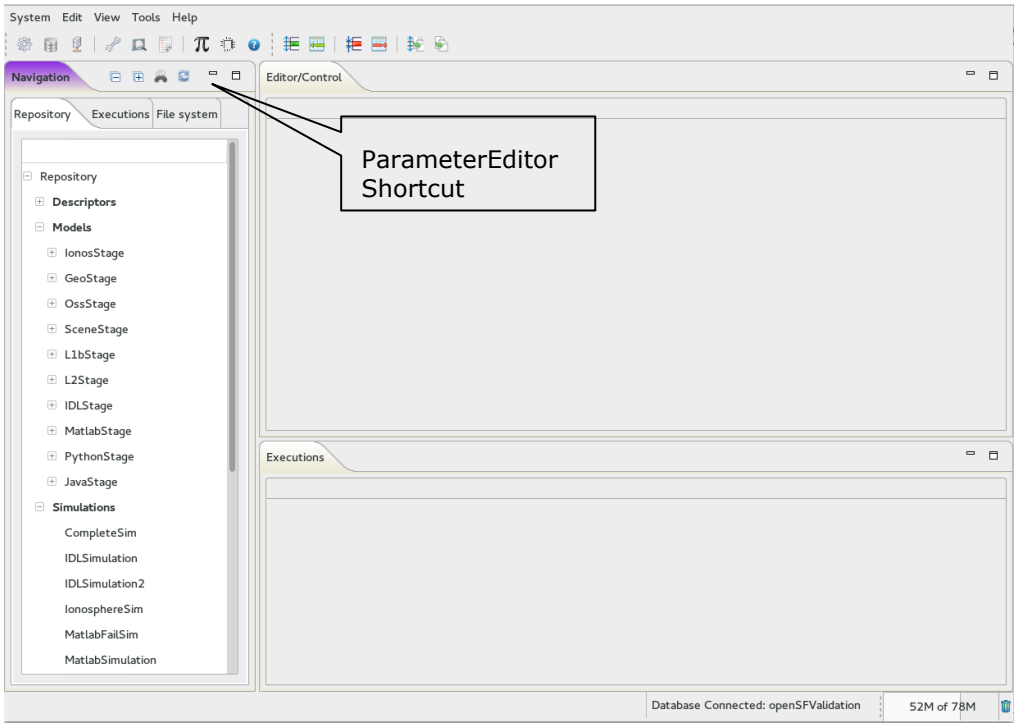

Figure 1: ParameterEditor Shortcut

In addition, it can be launched opening a terminal, going to openSF home folder and launching the start script *./ParameterEditor* or just double-clicking in the executable.

4. PARAMETER EDITOR

4.1. Main Frame

Here is presented the look-and-feel, operational behaviour and design features of the ParameterEditor application. The HMI (Human Man Interface) is based on the Eclipse RCP technology, which gives ParameterEditor a modern look.

The HMI accepts input via devices such as the computer keyboard and mouse and provides articulated graphical output on the display. Thus certain aspects of the HMI implement also the **Object Oriented Interface (OOUI)** paradigm because it is built from different pieces, or objects with several properties and operations.

The main window is based in three split panels allowing users to resize the component they want to focus on. The three graphic components of the ParameterEditor main frame are:

- Parameter File View: Tabbed panel showing the parameters stored in a set of files. An asterisk will appear before the parameters file name anytime a change is performed but it is not saved into the correspondent XML file.
- Log Console view: This panel shows information about the operations performed by the application data layer, exceptions, work flows, etc.
- Side panel containing the function buttons and showing the status of the application, files already opened, etc.

Figure 1 shows the appearance of ParameterEditor main window.

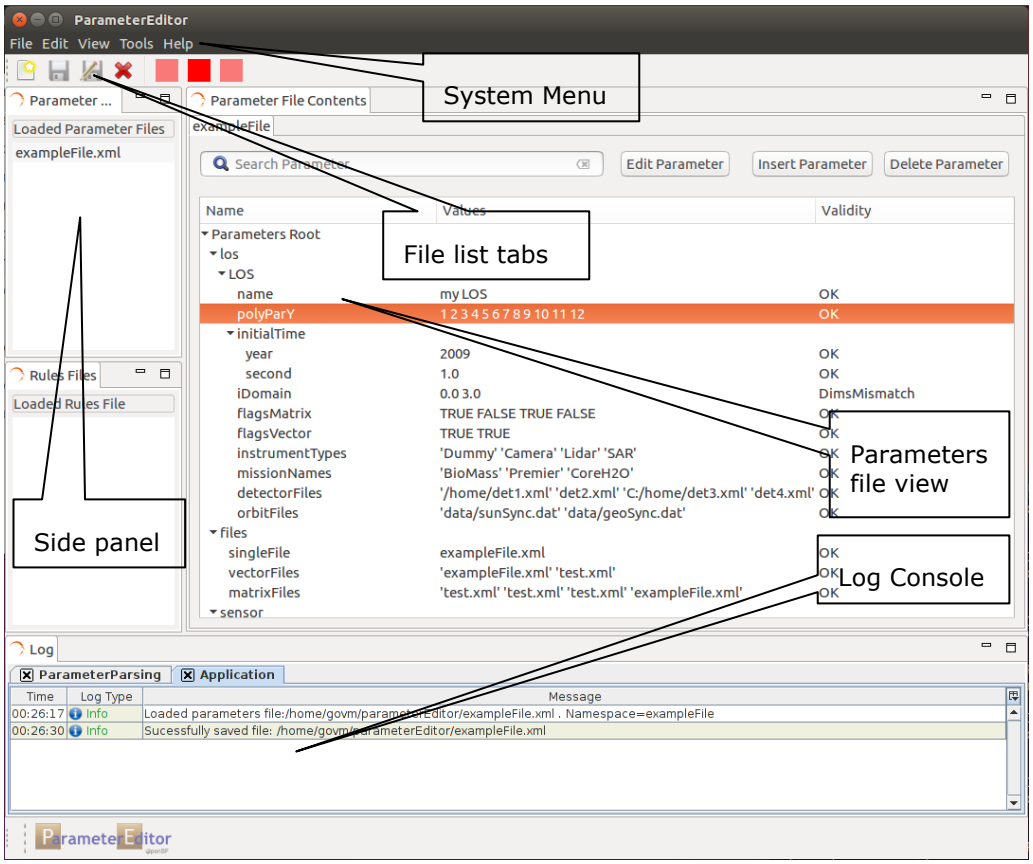

Figure 2: ParameterEditor Main Window Appearance

4.2. ParameterEditor Menu

The ParameterEditor menu provides the following actions, grouped by menu item:

File Menu

This menu gives access to ParameterEditor main functionalities such as:

- Open Parameters File: load a new parameters file or a set of parameters files (Hint: file browser allows the selection of more than one file)
- Save Parameters File: save current parameters file
- Save All Parameters Files: save all parameters files
- Open Rules File: Open a rules file, the system only allows one rules file opened at any time.
- Default Configuration Files: See section 4.9
- Validate: See section 4.7
- Exit: close the application

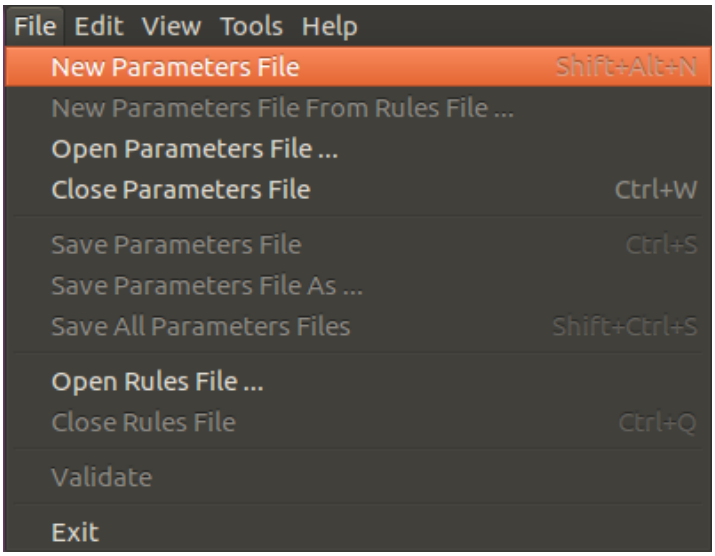

Figure 3: ParameterEditor File Menu

Log Menu

- Clear Logs: see section 4.6

Tools Menu

- XML built-in editor see section 4.4

Tools Menu

- About ParameterEditor

4.3. Parameter Management

This section details the use of the "Parameter File View" and consequently the creation, modification and deletion of parameters.

The functions available from this panel are:

- Navigate through parameters contained in a parameters file
- Create a new parameter
- Delete a parameter
- Find a parameter in a configuration file
- Edit XML file directly accessing to the built-in XML text editor (**KNOWN ISSUE**: not available for this version)

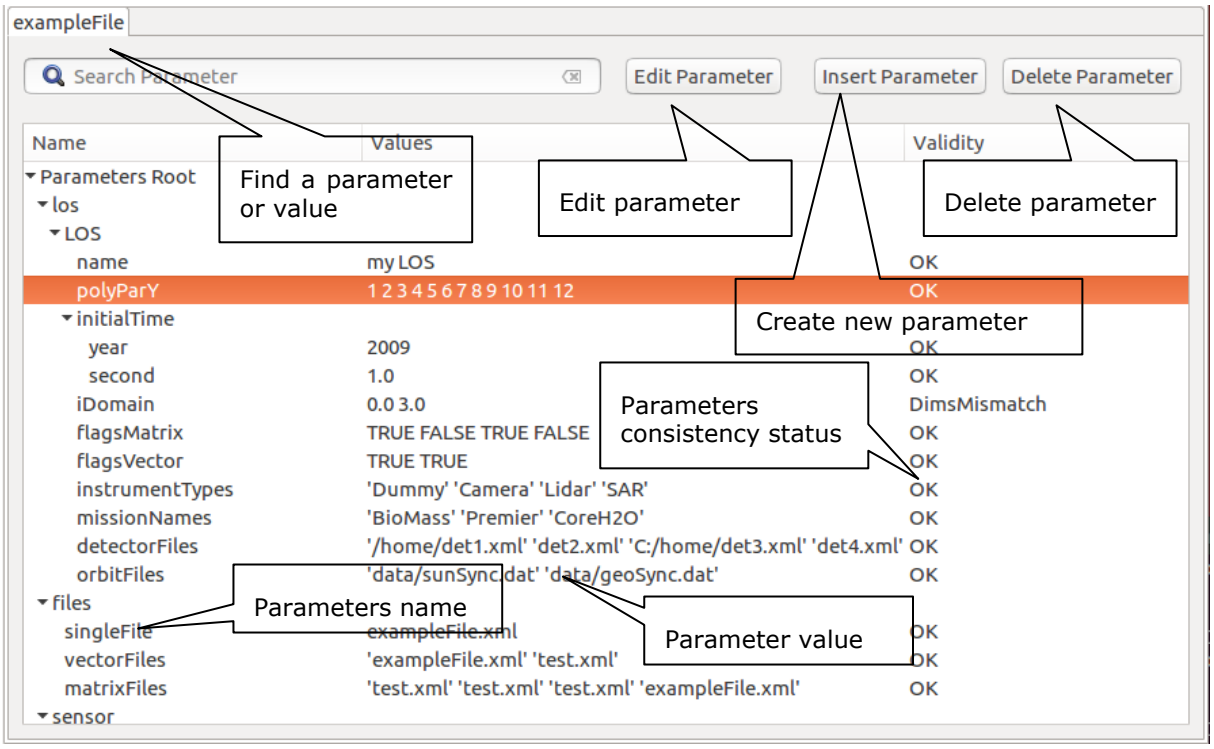

Figure 4: Parameters file, Parameters Tree View

Figure 4 shows the parameter file view and the functionalities it provides access to:

- Find: Assisted finding of a parameter name or a value
- Tree showing parameters names, values and the result of the consistency check. The result of the consistency check can be:
  - *Ok*: all consistency tests passed.
  - *TypeMismatch*: problem with the value and the type specified within the parameter.
  - *DimsMismatch*: when the number of values within a parameter does not match with the specified dimensions
  - *OutOfRange* (Warning): parameter values are not within minimum and maximum specified range. Only available for FLOAT and INTEGER parameters.
  - *Unknown*: an exception has occurred during the check.

4.3.1. Insert Parameter

From the Parameter File View shown in Figure 4 users can access to the "Insert parameter" window. From this view users can create a new parameter in the parameters file that is under editing.

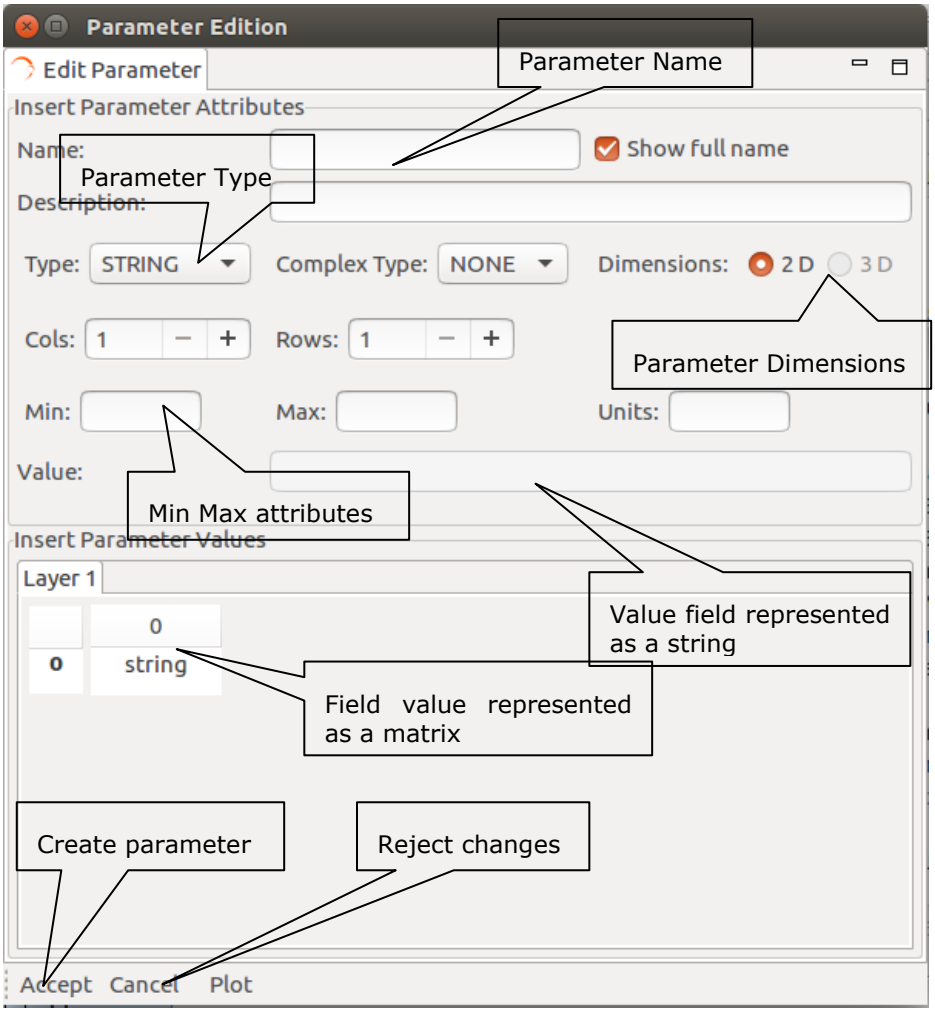

Figure 5: Insert Parameter Window

- Name: parameter name. See section 4.3.1.1 for details.
- Description: text description of the parameter functionality
- Dimensions: Rows and Cols, integer field specifying the number and dimensions of parameter values. When field is edited and the user press enter the matrix view is automatically updated.
- Type: list of the available parameter types, BOOLEAN, FLOAT, INTEGER, STRING, TIME, FOLDER and FILE.
- Complex Type: list of the available complex parameter types, NONE, ARRAY and MATRIX.
- Min/Max: double or integer values specifying the maximum and minimum values allowed for this parameter. Only available for FLOAT and INTEGER types.
- Value: text representation of the parameter values. It is recommended to use matrix view to update this field.

- Matrix View: table allowing the modification of a value in a determined matrix position. Row numbers are shown in vertical and column in horizontal. When a value is edited the field "Value" of the window is automatically reflects the changes.

Accept button creates the parameter in the configuration file and Cancel rejects changes and close the window.

4.3.1.1. Parameter Names

The name field shall be a unique identifier within a configuration file and is a string that reflects the structure of the XML grammar used for openSF [AD 1].

This can be better explained through an example:

- A parameter name "*sensor.band.wavelength*" has a XML representation:

```
<sensor>
    <band>
        <parameter name="wavelength" ….>
    </band>
</sensor>
```

- The short name for the parameter would be "*wavelength*" but the full parameter name reflects all the path within XML configuration file, including parent nodes
- Consequently when users enter a full name with dot separators, the system automatically creates parent nodes as specified

The checkbox "Show full name" next to the name text box is used to alter between showing the full name of the parameter or the short name.

4.3.2. Edit Parameter

The window for edit an existing parameter is the same as the one used for create a new one. This window shall appear anytime user double-clicked on a parameter node in the Configuration File Panel.

Figure 6 shows the editing window for an INTEGER MATRIX parameter.

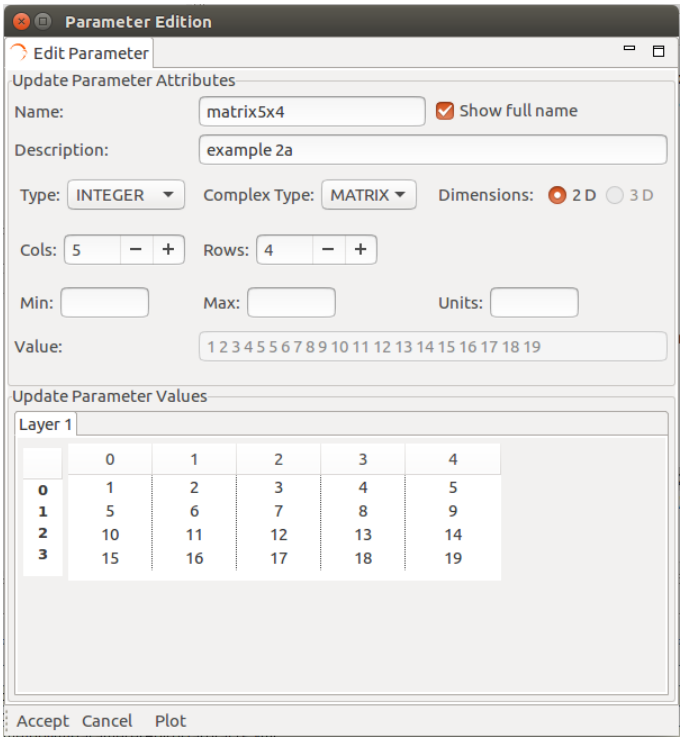

Figure 6: Edit Parameter Interface

4.3.3. Delete Parameter

From the Parameter File View (Figure 7) users are able to delete a parameter or a set of them by clicking on "Delete Parameter".

Note that the parameters tree allows the selection of more than one parameter[1] and delete action will erase from the system all the selected parameters. The system requires confirmation for parameter deletion.

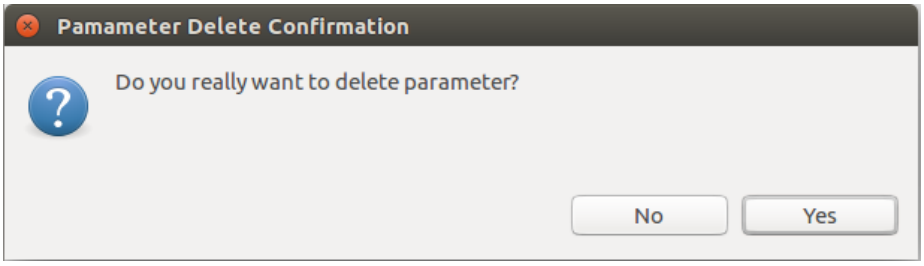

Figure 7: Parameter Deletion, Confirmation Message

4.3.4. Plot Parameter

The application also allow user to graph parameter values in a 3D plot frame. This functionality uses a third-party library to handle 3D plots in Java (JMathPlot http://jmathtools.berlios.de/doku.php).

This functionality is only available for numeric parameters (FLOAT and INTEGER).

Figure 8 shows the 3D plot of an integer matrix parameter.

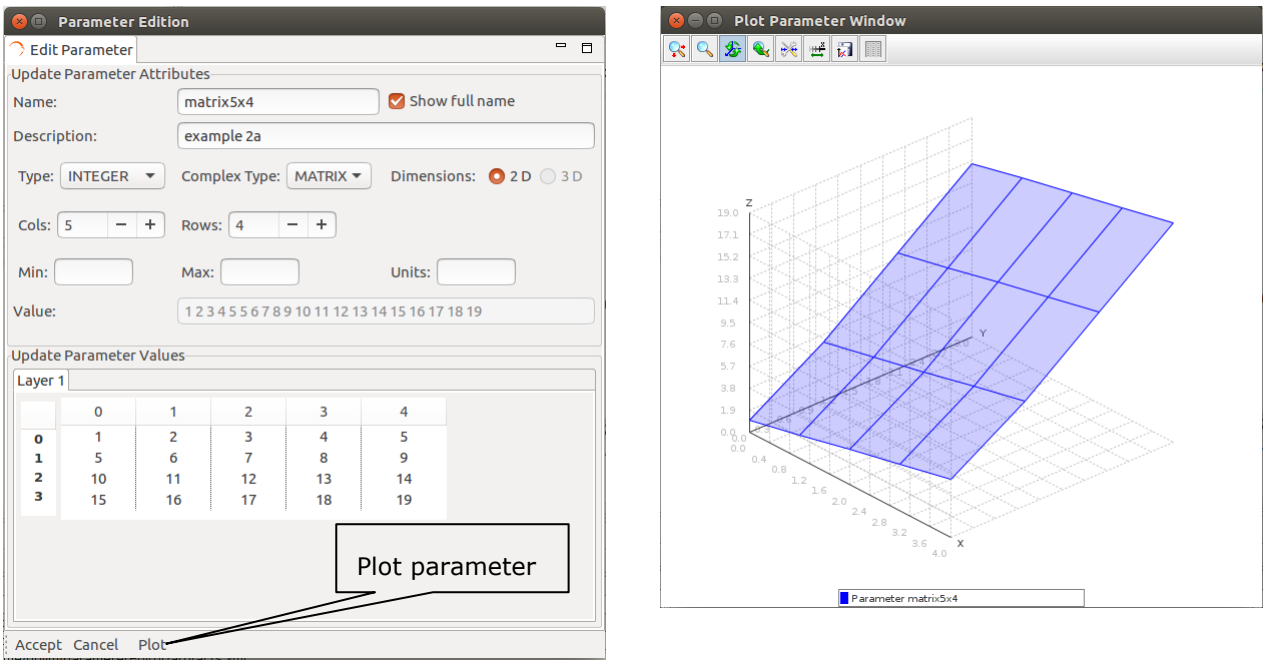

Figure 8: Parameter Plot Visualization

[1] To select more than one parameter, use Shift key for consecutive parameters and Ctrl key for non-consecutive selection.

4.4. Built-in XML Text Editor

The built-in XML text editor is a graphical editor for XML files. This application allows to easily edit xml files without relying in a third-party component.

The features of this software component are:

- ❑ XML syntax highlighting
- ❑ Error highlighting
- ❑ Line number indicator
- ❑ Text editor typical functionalities, copy, paste and cut text, new file, open, save and save as.
- ❑ Lightweight

Figure 9 shows an example of this component editing an openSF configuration file.

KNOWN ISSUE: the XML Text Editor will be available in future releases.

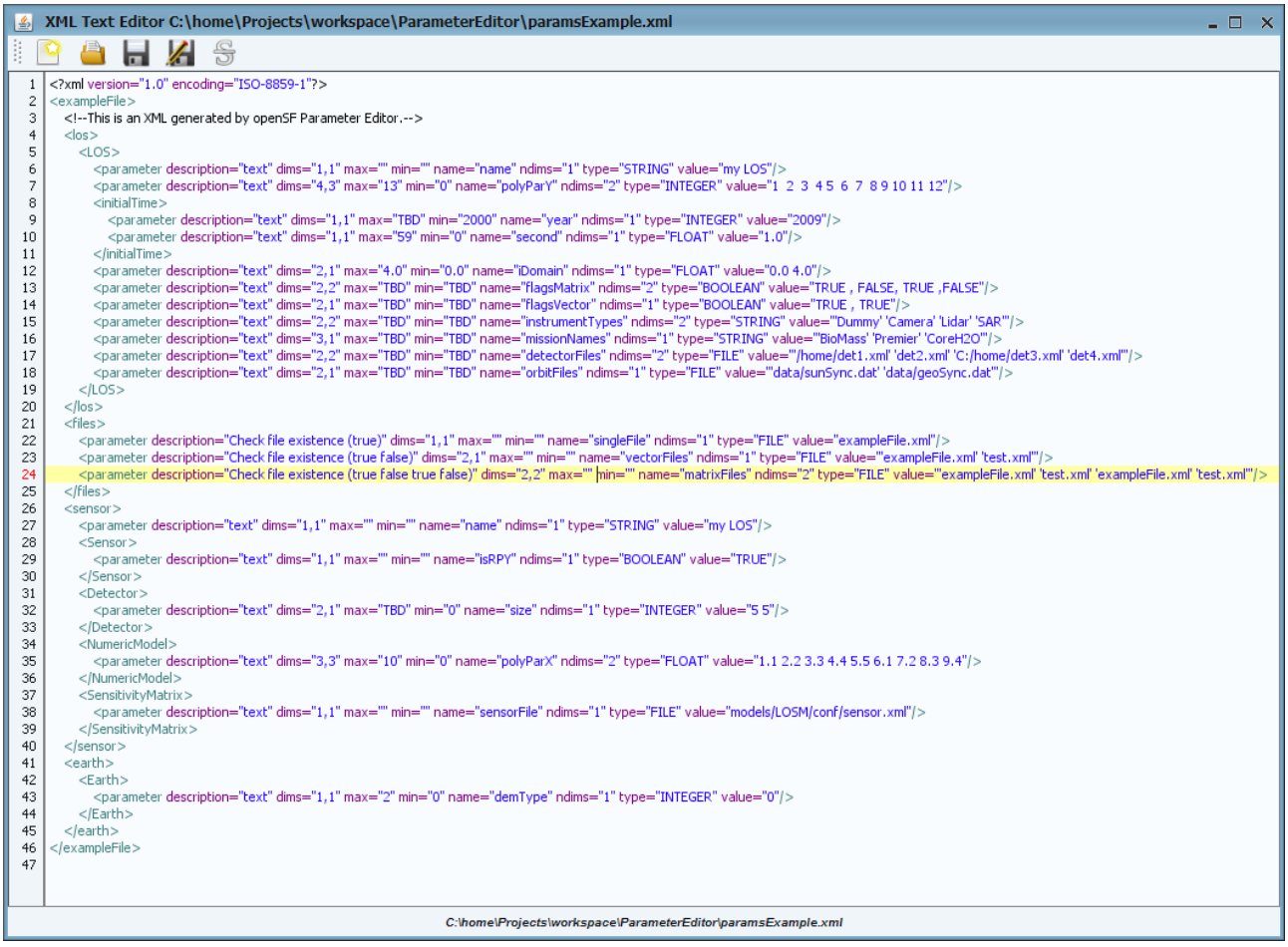

Figure 9: Built-in XML Text Editor

4.5. Side Panel

The ParameterEditor side panel can be found in the right side of the main frame and comprises the global status of the system showing.

- List of loaded configuration files
- Rules file loaded in the system

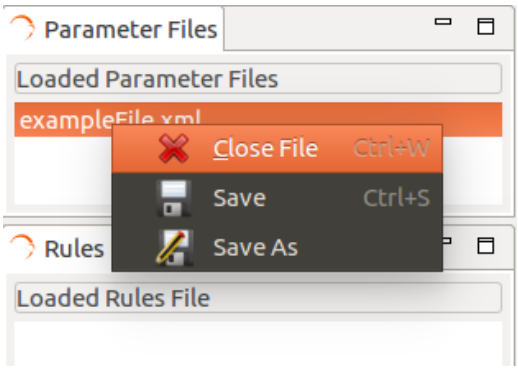

Figure 10: Side Panel Contextual Menu

This panel has also associated a contextual menu that allows to access extra functionality:

- Close configuration file
- Save only one file
- Save a file with other filename
- When Double-clicking on a parameters filename from the list, the tab corresponding for this file is focused or re-opened if it had been closed before.
- Double-clicking on rules filename opens the built-in XML text editor with this file opened.

The two groups, Parameter Files and Rules File, contained within the ParameterEditor side panel are expandable/collapsible through clicking on the arrow at the left-top corner of each block.

4.6. Log Console

The log console is a graphic tabbed panel that shows the system messages produced by the data/file management layer and the validation logic. This panel is resizable and can be found at the bottom of the ParameterEditor main window.

By default starts with two tabs opened, "Application" for ParameterEditor status messages such as workflow information, system status etc… and "ParameterParsion" that shows the errors and warnings of the consistency check.

Log console panel will also show new tabs when some actions are performed. These tabs correspondent to different log categories within ParameterEditor system. A ParameterEditor log message has the following attributes:

- *Time*: system time when log was produced
- *LogType*: depending on the log message impact. Info, Error, Exception, Warning, Debug
- *Message*: text describing the system event

Some of the log categories that can be shown in the console are:

- ***Application***: ParameterEditor status logs.
- ***Parameter Parsing***: logs from the consistency check of the configuration files.
- ***XML Parsing***: logs coming from the ParameterEditor XML parser.
- ***Validation***: logs corresponding to the rules validation process.

Anytime a new log message is registered into the system corresponding tab will change it background colour until user focus.

Figure 11 shows the interface of the Log Console.

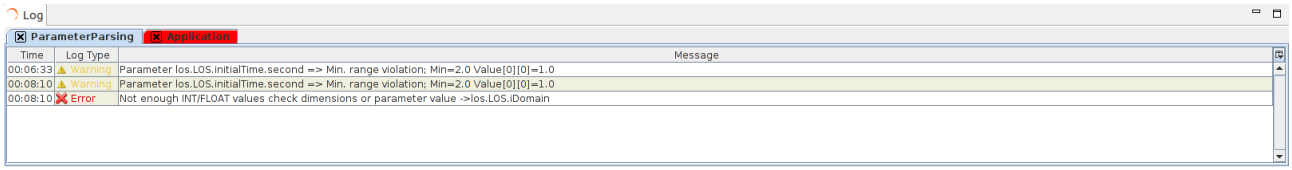

Figure 11: Log Console

4.7. Rules File – Validation

In this section will be described the process of building a rule file and the validation mechanism used within ParameterEditor in order to achieve configuration file constraints checking for parameters that can be defined in different files.

4.7.1. Rules File

The intended readers for this section are openSF experienced users, familiar with XML editing and parameter management.

4.7.1.1. Grammar Definition

A simple grammar has been designed for defining the rules that govern the ParameterEditor. This grammar is based in a XML syntax detailed below.

A single rule is composed by:

1. Unique rule identifier

   <rule id="ID">

2. Test description of the rule

   <rule id="ID" description="Text description">

3. Nested to the identifier, there is an operation tag. The operation types are three:

   - **Condition**: <condition type="ConditionType">. Condition type is a list of the most used logical operators
     - EXIST
     - NOTEXIST
     - EQUALS
     - GREATER
     - GREATEREQ
     - LESSER
     - LESSEREQ
   - **Action**: <action type="ActionType">. The syntax of an action is the same as the one used

- **IF statement**: an "if statement" shall be composed of a condition/s and action/s elements. If more than one condition is specified all of them are applicable to the rule (& like if).
- **Operand** <operandA …> or <operandB …>. This entity is explained in section 4.7.1.2

This grammar allows the user to define rules (conditions, constraints etc…) that will govern the parameter editing in the session definition/edition stage. The rules defined through this grammar will be applied to all configuration files loaded in ParameterEditor application.

In order to ease the creation of rules files it is provided a "Cheat Sheet" with the entities and tags used within the grammar previously described.

4.7.1.2. Operand entity

The operands are the basic "bricks" for the condition/action statements of a rule. The *operand* attributes are:

- *Name*: name of the parameter involved. Root tags of the XML configuration files (namespaces) are used in order to allow pointing to parameters in different files. Ex: *globalConfiguration.parameter* points to a configuration file whose root tag is *globalConfiguration*.
- *Type*: can be set to *attribute* or *constant*
- *Value*: if *Type* is set to "attribute" this field can take the following values: "*dims*" "*value*" "*min*" "*max*" pointing to the possible parameter attributes affected by the rule. If Type is set to "*constant*" this field can contains any desired string.
- *Row*: row or set of rows implied on this operand. Different selection schemas are available:
  - Enumeration: 1;3;7
  - Range: 2:5
  - Combined: 1;2:5;7
- *Column*: same as row but for columns

The *operand* available tags are *operandA* and *operandB* used for first and second statements respectively for *conditions/actions*.

4.7.1.3. XML Rules File

The XML rules file can have any desired root tag but shall contain a "*rulesDefinition*" tag in order to identify the start of the rules definition.

In this XML the tag <*file*> is forbidden as it is used for restricted purposes (see section 4.9).

XML Rules File Sample

```
<!-- Rules definition part-->
<rulesDefinition>
   <file value="C:\home\Projects\workspace\ParameterEditor\ruleAnnex.xml" />
   <rule id="rule01" description="Rule to link one parameter value with the dimension of another one">
      <condition type="EQUALS">
         <operandA name="globalConfig.intTest" type="attribute" value="value"/>
         <operand name="olciConfig.sensor.minMaxTest" type="attribute" value="dims" row="1" column="2"/>
      </condition>
   </rule>
   <rule id="rule02" description="Rule that check that one parameter value is greater than the
```

```
other">
      <condition type="GREATER">
         <operandA name="globalConfig.intTest" type="attribute" value="value"/>
         <operandB name="olciConfig.sensor.floatTest" type="attribute" value="value"/>
      </condition>
   </rule>
   <rule id="rule03" description="Rule to link one parameter value with the dimension of another
one">
      <condition type="LESSEREQ">
         <operandA name="globalConfig.intTest" type="attribute" value="value"/>
         <operandB name="olciConfig.sensor.floatTest" type="attribute" value="value"/>
      </condition>
   </rule>
   <rule id="rule04" description="Rule to show if capabilities">
      <if>
         <condition type="LESSEREQ">
            <operandA name="globalConfig.intTest" type="attribute" value="value"/>
            <operandB name="olciConfig.sensor.floatTest" type="attribute" value="value"/>
         </condition>
         <action type="EQUALS">
            <operandA name="globalConfig.test" type="attribute" value="value"/>
            <operandB name="olciConfig.sensor.floatTest" type="attribute" value="value"/>
         </action>
      </if>
   </rule>
</rulesDefinition>
```

In order to clarify the definition of rules below can be found the translation to pseudo-code/verbose of some of the rules specified in the sample.

- Rule 01: "*value*" field of the parameter identified by "*globalConfig.intTest*" must be equals to "*dims*" attribute at *[1][2]* of parameter "*olciConfig.sensor.minMaxTest*"
- Rule 04: if (value attribute of "*globalConfig.intTest*" is lesser or equals than attribute value of "*olciConfig.sensor.floatTest*") then (attribute value of "*globalConfig.test*" must be equals to attribute value of "*olciConfig.sensor.floatTest*")

4.8. Validation Process

In order to validate a group of configuration files user shall follow the following steps:

- Load the parameters files in ParameterEditor (see section 4.5)
- Create a rules file with the grammar specified in 4.7.1.1 and load it in ParameterEditor (see section 4.5)
- Click on "Validate" button (see section 4.5)
- Check Log Console in order to check if validation process has been successful and the possible restrictions/constraints violated

4.9. Auto-Build Configuration Files

The mechanism for this functionality has been designed but it is not yet implemented in the ParameterEditor application.

4.10. Default values and validation of parameters

The ParameterEditor can load default values for each parameter. The first step is to load the rules file though. This is possible using in the menu File, then the option "Load Rules File" (Figure 12). When loaded, the rules file will be shown on the left side of the main window.

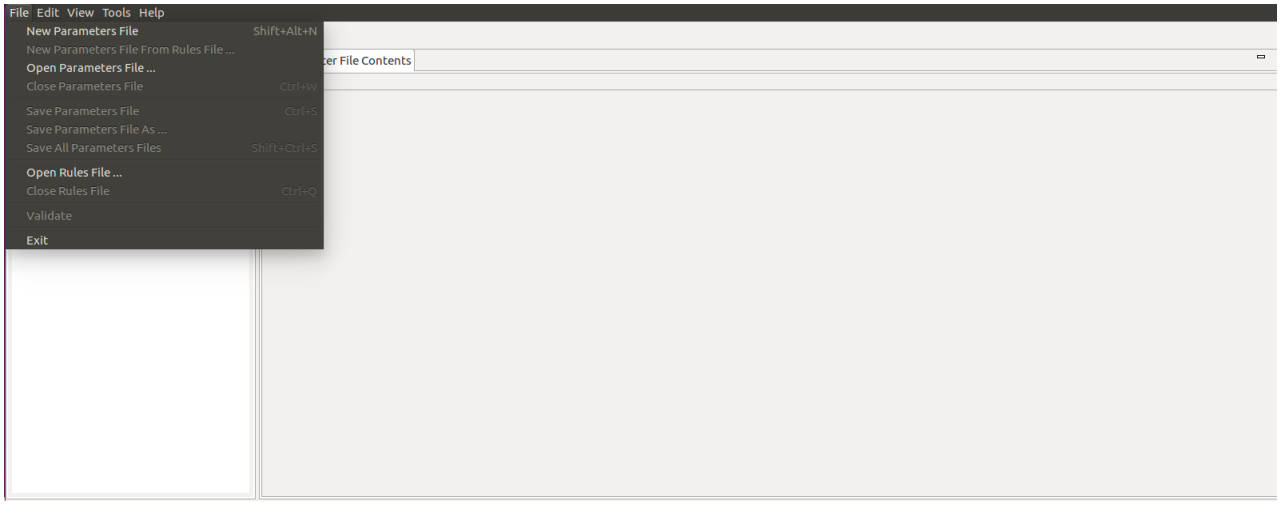

Figure 12: Main Area of the ParameterEditor without Parameters and Rules Files

The user cannot validate if by mistake no rules file is loaded. The file with rules can be hand-made by the user as long as the syntax described in this SUM is fulfilled.

Once the rules and parameters file are loaded, the result will be similar to Figure 13. There will be several changes respect to the initial empty main window:

- On the side panel, a list of the loaded files will be shown.
- On the lower part, there is a quick analysis of the parameters, checking for example that the values are within the min/max ranges, etc.

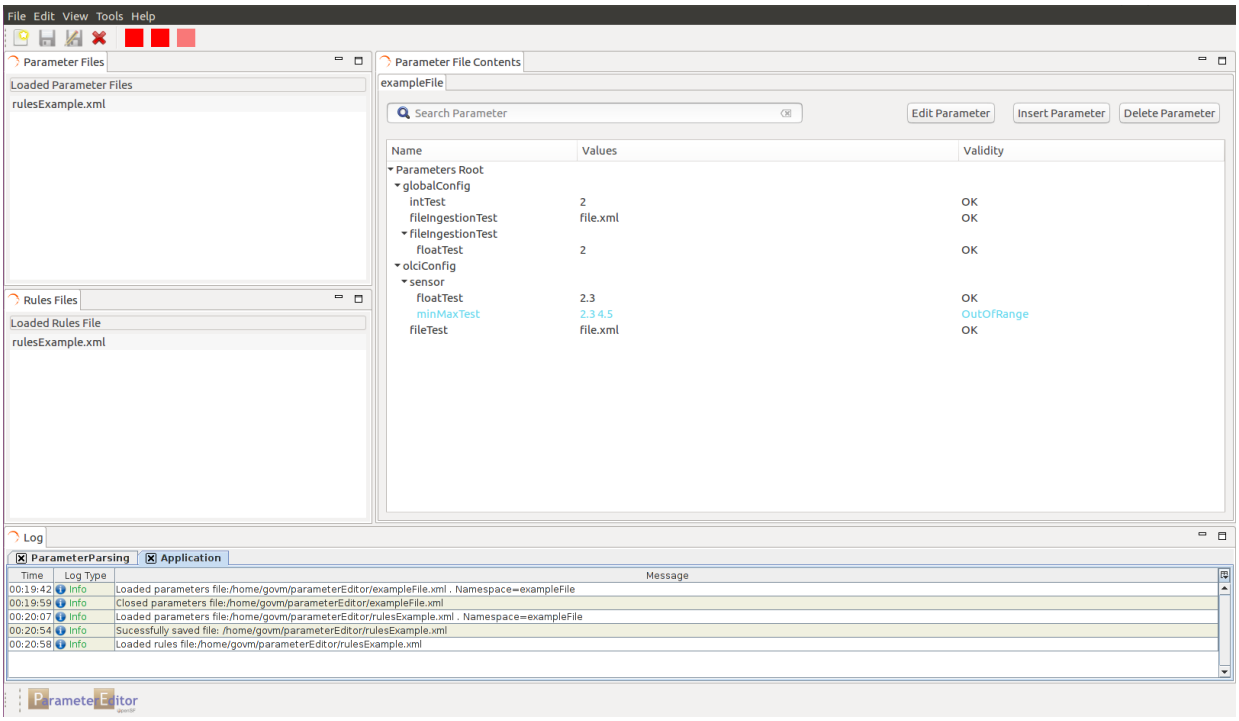

Figure 13: Rules File and Default Configuration Loaded

For a full validation against the file with rules, the user has to select "validate", an option under the File menu. A new window is generated (Figure 14). This window summarizes the results per file, splitting the log per parameters file. If there is an error, the user should return to the main window, make the necessary changes, and select again the validation option (a new pop up with the validation results will appear; several validation pop-ups can co-exist).

It is possible to keep opened several windows with the result of the validation each time the option is selected. Note however that the validation window (Figure 14) is static, that is, it is not automatically updated each time a change in the configuration files is done. Thus, although it is not mandatory, it is recommended that in case errors are detected in the configuration files, the validation window is closed. Then changes are done in the configuration and again the validation option is selected for a new pop-up. This way, at any time only a validation window will be on the field of view of the user reflecting the latest configuration values.

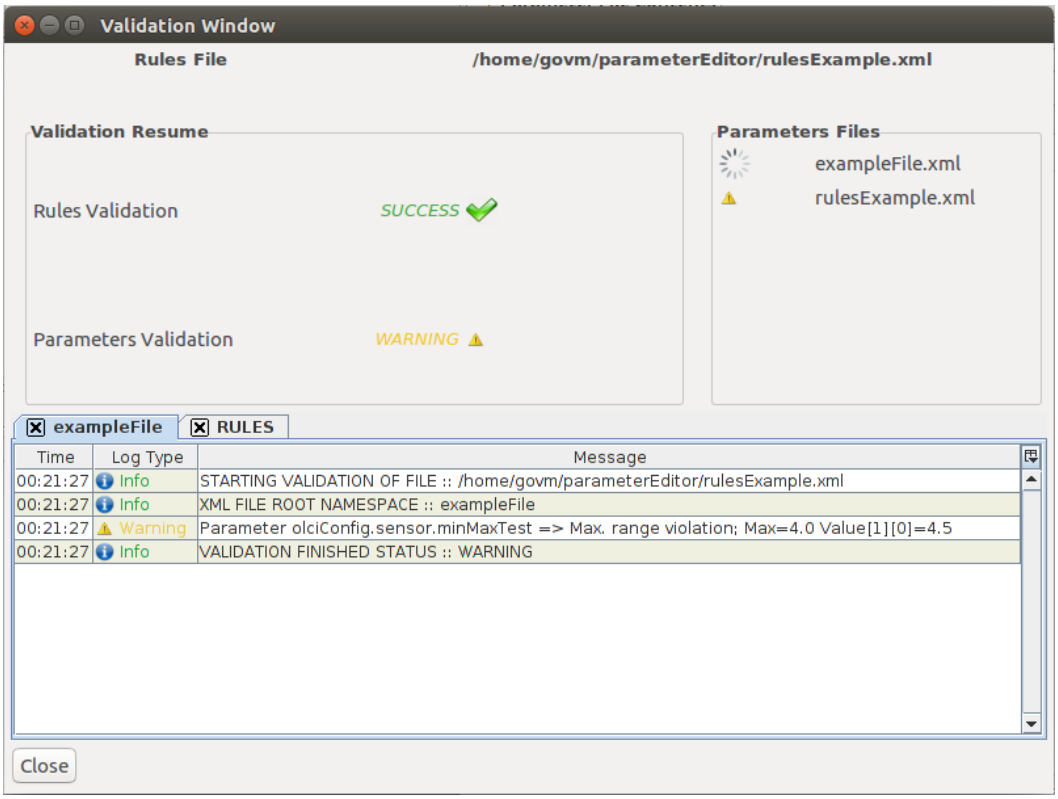

Figure 14: Validation Pop-up Summarizing the Results

When the user finds that validation is fully successful, or also when the user considers that the errors or warnings can be ignored for the purposes of the test involved, it is mandatory to save all the files (there is not an automatic saving). Files can be saved individually or all at once; these are commands "Save" and "Save all" on the File menu.