



ESO - EUROPEAN SOUTHERN OBSERVATORY

EUROPEAN SOUTHERN OBSERVATORY
Organisation Européenne pour des Recherches Astronomiques dans l'Hémisphère Austral
Europäische Organisation für astronomische Forschung in der südlichen Hemisphäre

Reflex User Manual

VLT-MAN-ESO-19000-5037

Issue 2.0
16/12/2010
21 pages

Prepared: V. Forchì

.....
Name Date Signature

Approved: T. Bierwirth

.....
Name Date Signature

Released: M. Péron

.....
Name Date Signature

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	2 of 21

CHANGE RECORD

Issue	Date	Affected Paragraphs(s)	Reason/Initiation/Remarks	Author(s)
0.1	18/01/10	All	First version	V. Forchì
0.2	28/01/10	All	Added installation and troubleshooting sections	V. Forchì
0.3	16/03/10	All	Many corrections	V. Forchì
0.4	07/04/10	4, 5	Added Iteration	V. Forchì
0.5	28/05/10	All	Updated to beta2	V. Forchì
0.6	14/06/10	All	Added ProductRenamer chapter, minor fixes	V. Forchì, C. Garcia
0.7	29/06/10	All	Minor fixes, preparing for first public release.	V. Forchì
1.0	27/07/10	4.4.1, 4.12	Parameter order, actor description	V. Forchì
2.0	16/12/10	4.5, APPENDIX B:	Updated to Reflex 1.1 and some minor fixes	V. Forchì

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	3 of 21

TABLE OF CONTENTS

1.	INTRODUCTION.....	5
2.	INSTALLATION.....	6
2.1	Reflex installation.....	6
2.2	Execute Reflex.....	6
3.	REFLEX OVERVIEW	7
3.1	General concepts	7
3.2	Reflex specific actors.....	7
3.3	Data types	8
3.4	General workflow structure.....	9
3.5	General features	10
4.	ACTOR DOCUMENTATION	11
4.1	DataOrganizer	11
4.1.1	Parameters.....	11
4.1.2	Ports.....	11
4.2	DataSetChooser	11
4.2.1	Parameters.....	11
4.2.2	Ports.....	11
4.3	FitsRouter.....	11
4.3.1	Ports.....	12
4.4	RecipeExecuter.....	12
4.4.1	Parameters.....	12
4.4.2	Ports.....	13
4.5	PythonActor	13
4.5.1	Parameters.....	14
4.5.2	Ports.....	14
4.6	DataFilter.....	14
4.6.1	Parameters.....	14
4.6.2	Ports.....	14
4.7	SofCreator	15
4.7.1	Parameters.....	15
4.7.2	Ports.....	15
4.8	XMLFormatter	15
4.8.1	Parameters.....	15
4.8.2	Ports.....	15
4.9	SopCreator	15
4.9.1	Ports.....	15
4.10	SofCombiner	15
4.10.1	Ports.....	15
4.11	RecipeLooper	15
4.11.1	Ports.....	15
4.12	ProductRenamer	16
4.12.1	Parameters.....	16
4.12.2	Ports.....	16
5.	RECIPE ITERATION	17
6.	TROUBLESHOOTING	18
6.1	Another instance of Reflex is running.....	18

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	4 of 21

6.2	Errors during startup.....	18
6.3	Debug mode.....	18
6.3.1	Logging to file.....	18
APPENDIX A.	SIMPLIFIED SOF AND SOP FORMAT	19
APPENDIX B.	EXAMPLE.PY	20
APPENDIX C.	SOFTWARE REQUIREMENTS	21

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	5 of 21

1. INTRODUCTION

The ESO Recipe Flexible Execution Workbench (Reflex) is an environment which allows an easy and flexible way to execute VLT pipelines. It is built using the Kepler workflow engine (<https://kepler-project.org>), which itself makes use of the Ptolemy II framework (<http://ptolemy.eecs.berkeley.edu/ptolemyII>).

The Kepler project has thorough documentation both for the casual and experienced user (<https://kepler-project.org/users/documentation>).

Reflex allows the user to process his scientific data in the following steps:

- Associate scientific files with required calibrations
- Choose datasets to be processed
- Execute several pipeline recipes

This process, also called a workflow in Kepler terminology, is visually represented as a sequence of interconnected boxes (actors) that process data: the workflow allows the user to follow the data reduction process, possibly interacting with it. The user can visualize the data association and the input files and decide what scientific data he wants to process. It is also possible to visualize intermediate products, using components provided by Reflex, or modify the data flow with custom components.

Reflex uses EsoRex (<http://www.eso.org/sci/data-processing/software/pipelines>) to execute the pipeline recipes, but this is not exposed to the user.

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	6 of 21

2. INSTALLATION

The prerequisite to run Reflex is the installation of EsoRex and the pipeline you are interested in (please refer to the VLT pipelines webpage for more detailed information <http://www.eso.org/pipelines>).

A script that takes care of all the steps required to run a workflow is available at the following URL: ftp://ftp.eso.org/pub/dfs/reflex/uves_install_reflex. The tool downloads and installs the UVES pipeline, that contains one workflow, a sample dataset and reflex.

If you have already installed the pipeline and you have your own dataset, you can skip some steps and proceed to the Reflex installation. Note that at the time of distribution only the UVES pipeline contains Reflex workflows.

Note: at the moment Reflex is supported on Linux only (see APPENDIX C:).

2.1 Reflex installation

This step is not required if you used the installation script.

- Download reflex:

```
$ wget ftp://ftp.eso.org/pub/dfs/reflex/reflex-1.1.tar.gz
```

- Install Reflex:

```
$ cd /path/to/install/reflex
```

```
$ tar xzf /download/location/reflex-1.1.tar.gz
```

2.2 Execute Reflex

If you used the installation script execute:

```
$ /path/to/install/software/bin/reflex
```

Otherwise execute:

```
$ /path/to/install/software/reflex-1.1/eso-reflex/bin/reflex
```

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	7 of 21

3. REFLEX OVERVIEW

3.1 General concepts

Kepler visually represents a workflow as a sequence of actors with a single director: the latter schedules the execution of the actors and the former manipulates the data.

The main components of a workflow are:

- Director: determines the execution order of the actors and tells them when they can act. There are several director types, the default for Reflex is the Dynamic Dataflow (DDF) director. There must only be one director per workflow.
- Actor: represents a single step of execution. The actor takes data from the input ports, processes them and sends the results to the output ports. Each actor can have a number of parameters that control its execution: to edit the parameters double click on the actor.
- Port: each actor can possess one or more ports, which allow it to exchange data with other actors. Ports can be input-only (a triangle pointing into the actor), output only (a triangle pointing out from the actor), or bidirectional (a circle). A port can be singular or multiple: the former can be connected to only one port, the latter can be connected to many. Singular ports are black, multiple ports are white.
- Token: an object that encapsulates data. Actors exchange data in the form of tokens through ports. A token can contain several types of data: integers, strings, floating point, etc.

3.2 Reflex specific actors

Reflex is composed of a number of custom actors that allow the workflow to interact with the VLT pipeline and with FITS files, namely:

- DataOrganizer
- DataSetChooser
- FitsRouter
- RecipeExecuter
- PythonActor
- DataFilter
- SofCombiner
- SofCreator
- SopCreator
- XMLFormatter
- RecipeLooper
- ProductRenamer

All these actors, except for the RecipeExecuter, can be inserted into the workflow from the component tree on the left hand side of the Reflex window, under Projects->ESO (Figure 1).

The RecipeExecuter is not present in the menu and must be instantiated from the menu bar: click on Tools -> Instantiate Component, change the Class name to org.eso.RecipeExecuter and click OK (Figure 2). You will then be presented with a combobox containing all the available CPL recipes (the list is obtained from the output of the command `"esorex --recipes"`): select the desired one and click OK (Figure 3). Once a particular recipe has been chosen it cannot be changed: if you chose the wrong one then you have to remove the actor and instantiate a new one.

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	8 of 21

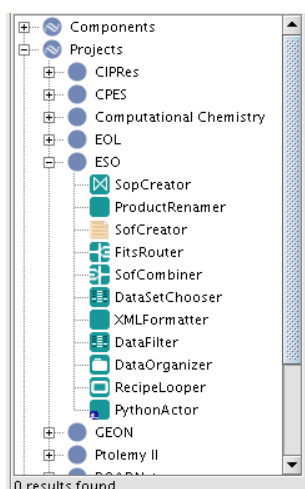


Figure 1: ESO specific actors

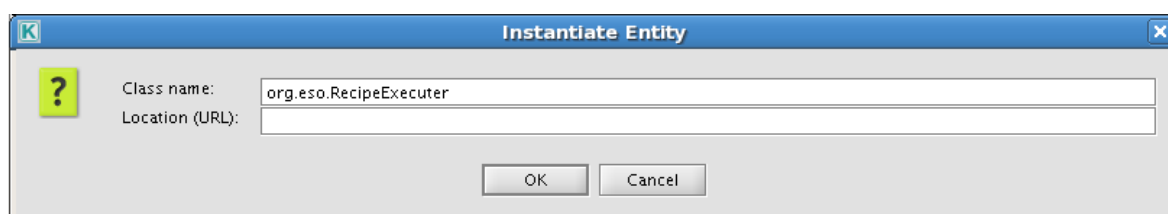


Figure 2: How to instantiate the RecipeExecutor

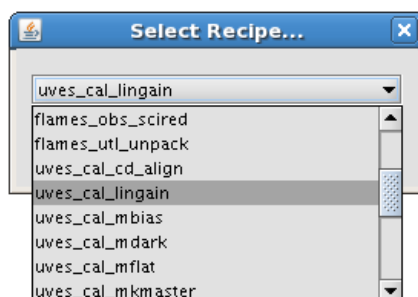


Figure 3: Recipe selection

3.3 Data types

Reflex specific actors exchange data in the form of List of Science Observations (LoSOs), Set of Files (SoF) and Set of Parameters (SoP): they are all represented as an XML structure and are not meant to be read by a human. Use the dedicated actor XMLFormatter to see them.

- LoSO: contains some general information, a SoF and a SoP
- SoF: contains a set of science frames and associated calibration files which are required to process them
- SoP: contains the values of the parameters for the execution of the pipeline recipes.

In the same component tree that contains Reflex actors you can find many other actors, which are part of the standard Kepler distribution; a detailed description can be found in the Kepler User Manual, but they mostly solve general needs, such as:

- Mathematical operations (arithmetic, statistical, logic...)
- File manipulation (open, read, write...)
- File system interaction (list directory content, remove directory...)
- Command execution (shell, ssh, condor...)
- Workflow control (switch, pause, stop...)

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	9 of 21

3.4 General workflow structure

The general structure of a Reflex workflow is depicted in Figure 4. In the top-right hand corner there are a number of parameters required for the workflow execution:

- RAWDATA_DIR: a directory containing all the raw data to be processed (Note: subdirectories are recursively scanned)
- BOOKKEEPING_DIR: a directory where each pipeline recipe execution will create a subdirectory to use as a working directory
- LOGS_DIR: a directory where the recipe logs will be saved
- TMP_PRODUCTS_DIR: a directory where the workflow intermediate products will be saved
- END_PRODUCTS_DIR: a directory where the workflow final products will be saved
- FITS_VIEWER: executable used to visually inspect FITS files
- ESORexArgs: additional parameters passed to esorex by the RecipeExecuter

The data under each directory, except the one containing the final products, are organized in the following way:

BASE_DIR/ActorName/ExecutionTime

e.g. /home/reflex/reflex_tmp_products/Uves_Blue/uves_cal_mbias_1/2010-01-01T12:23:12.123 contains the products of the execution of the actor named uves_cal_mbias_1 at the timestamp "2010-01-01T12:23:12.123".

If you double click one of these parameters you will be presented with a configuration window, where you can select the value of the parameter, either by typing it or browsing the filesystem (do not forget to save your modifications).

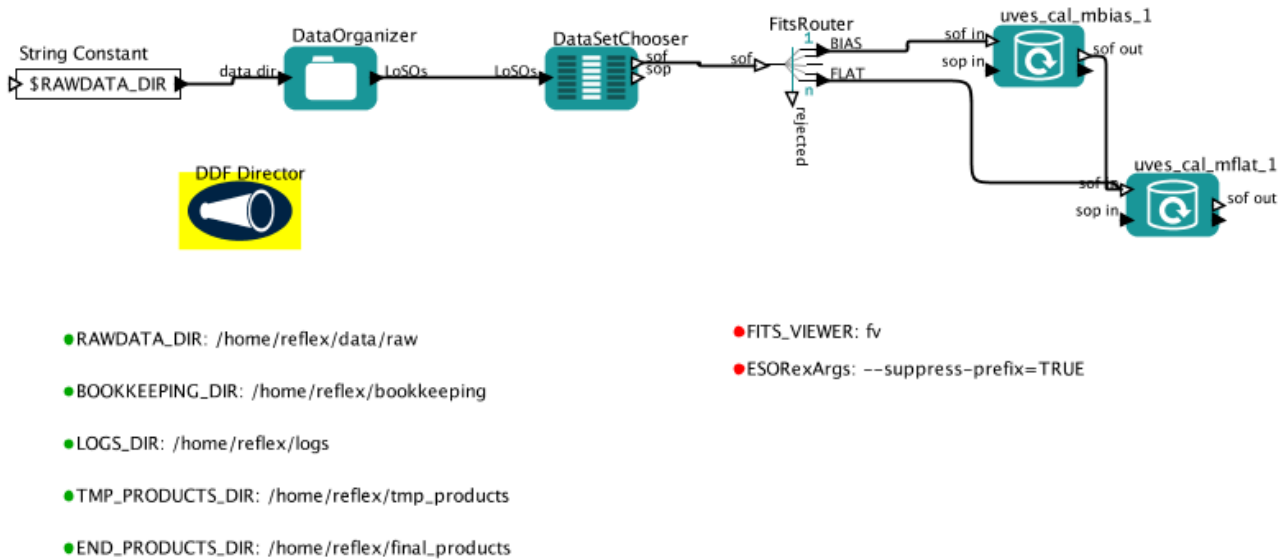


Figure 4

The preferred director for Reflex workflows is the DDF director. In principle the PN director can be used as well, but workflows are guaranteed to work only with the DDF director.

The first step of the workflow execution is the DataOrganizer, which organizes a set of files and groups them together in datasets according to some classification and grouping rules, written with the OCA language (a full set of rules is provided together with each workflow): the output is a set of so called LoSO (List of Science Observations), which contain science frames and all the associated raw calibrations required to process them. Furthermore, if the OCA rules define custom values for the recipe parameters, then these are included in the LoSO as well.

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	10 of 21

The second step is the DataSetChooser, which displays all the LoSOs provided by the data organizer, and allows the user to view their contents and select the ones that are to be processed. The output of this actor is, for each selected LoSO, a SoF (Set of Files), which contains the files to be processed, and a SoP (Set of Parameters), which contains the parameters for the pipeline recipe execution.

It is then possible to connect the output of the DataSetChooser either to a RecipeExecuter or to a FitsRouter, which splits the SoFs and sends the files to different output ports based on their observation type: these ports can then be used to feed the various pipeline recipes of the workflows.

3.5 General features

- Workflow execution: you can start, pause, stop and resume the workflow by using the buttons in the toolbar. The highlighted button indicates which state the workflow is in. Please note that the stop button immediately interrupts any running pipeline recipe, while the pause button lets the current recipe or actor finish before the workflow is actually paused. After pressing the pause button, it is also possible that more than one actor is executed, since this behaviour depends on the scheduling policy. For instance, if there are two actors in parallel, and you pause the workflow while one is being executed, then both of them will be executed before the workflow is actually paused.
- The DDF director supports actor highlighting; this feature is disabled by default: if you want to enable it click on Tools->Animate at Runtime, select an interval (e.g. 10ms) and click ok. From now on the active workflow actor will be highlighted in red. Note: if you pause and resume a workflow the actor is not highlighted upon resume.
- The runtime window (Workflow->Runtime Window) displays the current status of all the recipes and the input/output files of the last execution: this is useful especially if you are using the PN director. Note, however, that if you close the runtime window while the workflow is running then the workflow will stop.

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	11 of 21

4. ACTOR DOCUMENTATION

The following documentation is also available on the workflow, by right-clicking on an actor and selecting Documentation->Display. Furthermore, RecipeExecuter instances display the help of all the recipe parameters.

4.1 DataOrganizer

The DataOrganizer organizes and classifies raw and reduced data: it takes as inputs all the files contained in a given directory, classifies them and organizes them into LoSOs (Lists of Science Observations). Each LoSO contains science frames, all the associated calibrations needed to reduce them and custom parameters for the reduction recipes. If a file is present more than once in the directory structure it will be associated multiple times.

4.1.1 Parameters

- **OCA File:** the filename (full path) containing the OCA rules used for classification and organization.
- **Keywords to be displayed:** a comma separated list of FITS keywords which are extracted from all files in addition to the ones selected by the OCA rules.

4.1.2 Ports

- **data dir:** the directory containing the input data which is scanned recursively.
- **LoSOs:** the produced LoSOs.

4.2 DataSetChooser

The DataSetChooser allows you to view and select the groups of files created by the DataOrganizer: it displays a list of LoSOs and provides the possibility of selecting, deselecting and analyzing them. It produces one token on the output port per selected LoSO.

4.2.1 Parameters

- **Mode:**
 - o Skip: automatically select all LoSOs and do not show any window.
 - o Display: select all LoSOs and allow the user only to view them.
 - o Select: standard mode, allows the user to inspect the LoSOs and select the ones he wants to process.
- **FITS Viewer:** the application used to inspect FITS files

4.2.2 Ports

- **LoSOs:** the input LoSOs
- **sof:** the SoF for the current LoSO
- **sop:** the SoP for the current LoSO
- **name:** the name of the current LoSO (hidden by default)
- **selected:** the number of selected LoSOs (hidden by default)

4.3 FitsRouter

The FitsRouter sorts files based on their category: by default, all files are routed to the rejected directory, but the user can add an additional port and configure the actor to route particular files to the new port. The category is defined by the value of either HIERARCH.ESO.PRO.CATG (keyword added to the file by the pipeline recipes) or DO.CATG (virtual keyword created by the DataOrganizer).

Basic Mode: create an output port named as a category (e.g.: FLAT). All files belonging to that category will be routed to that port.

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	12 of 21

Advanced Mode: create an output port with any name (e.g.: MyPort), and then create a string parameter whose name is the port name plus “_config” (e.g.: MyPort_config). This parameter can be a comma separated list of regular expressions: each file whose category matches at least one of the regular expressions will be routed to this port.

If you want to define the minimum or maximum number of files that the actor sends to a port, then you can define a parameter whose name is the port name plus “_min_number” or “_max_number”, respectively. If there are too many files, then the actor selects the first, and if there are too few files, then an error is reported.

Note: one file can be routed to many ports and, if a file is not routed to any port, it is routed to the rejected port.

4.3.1 Ports

- **sof:** the input SoF
- **rejected:** the rejected SoF

4.4 RecipeExecuter

The RecipeExecuter executes one CPL recipe. The recipe can be chosen only when you instantiate the component: at the moment the user is presented with a list of the available recipes (the output of `esorex --recipes`). After the user selects the recipe the actor queries EsoRex for the recipe parameters and adds them to the actor. Each parameter is set to the default value: if the user wants to change any of the parameters then he can enter his own value or he can write the special value PORT, in which case the value is taken from the input port SoP.

The actor has three modes: Run, Skip and Disabled.

Note: since this actor has very different behaviour depending on the pipeline recipe to which it is associated, it is not present in the left hand side actor list, and instead it must be instantiated from the menu (see section 3.2).

4.4.1 Parameters

- **Recipe:** the selected pipeline recipe (cannot be changed)
- **Mode:**
 - o Run: the recipe will be executed (see lazy mode)
 - o Skip: the inputs are broadcast to the output (according to input and output tags)
 - o Disabled: the recipe is not executed and an empty SoF is generated on the output port
- **Lazy Mode:** if true then the actor will check whether the pipeline recipe has already been executed with the same input files and with the same recipe parameters. If this is the case then the recipe will not be executed, and instead the previously generated products (which are the same as those that would have been generated if the recipe were executed, except for a timestamp) will be broadcast to the output port.
- **Recipe Failure Mode:** it specifies the behavior of the actor if the recipe fails
 - o Stop: the actor produces an error message and the workflows stops. This is the default
 - o Continue: the actor outputs an empty SoF
 - o Ask: the actor pops up a window and asks the user whether he wants to continue or stop the execution
- **Input Files Tag:** a comma separated list of file categories: only input files belonging to one of these categories will be passed to the recipe
- **Output Files Tag:** a comma separated list of file categories: only products belonging to one of these categories will be broadcast to the output port
- **Allow empty inputs:** if true then an empty sof on the input port will be broadcasted to the output without executing the recipe

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	13 of 21

- **Pause before execution:** if true then the execution is paused just before executing the recipe
- **Pause after execution:** if true then the execution is paused after the execution of the recipe.
- **Clear Products Dir:** indicates if and when the products directory for this actor will be cleaned: possible values are BEFORE, AFTER and NEVER
- **Clear Logs Dir:** indicates if and when the log directory for this actor will be cleaned: possible values are BEFORE, AFTER and NEVER
- **Clear Bookkeeping Dir:** indicates if and when the bookkeeping directory for this actor will be cleaned: possible values are BEFORE and NEVER
- **Products Dir:** a directory where the products will be created
- **Logs Dir:** the directory where the EsoRex logs will be saved
- **Bookkeeping Dir:** the directory the recipe will use as a working directory
- **EsoRex default args:** additional parameters passed to EsoRex
- **recipe_param_xx:** dynamically generated by the actor when a recipe is selected. Each recipe parameter is mapped to an actor parameter with this name pattern. The parameter value must be in the format "par_name=par_value": if the user tries to change the parameter name an error will be reported. Note: in case of a pipeline upgrade that modifies the number of parameters of a recipe you can end up with duplicated or missing parameters. In that case it is suggested to instantiate the actor again
- **Reuse Inputs (Expert Mode):** if true then the actor uses the inputs from the last execution, ignoring the input ports
- **Reuse Outputs (Expert Mode):** if true then the actor broadcasts the output from the last execution.

4.4.2 Ports

- **sof in:** the input files
- **sof out:** the output files
- **sop in:** the recipe input parameters
- **sop out:** the parameters used in the last execution of the recipe
- **rejected inputs:** the inputs not matching any of the input tags
- **rejected outputs:** the outputs not matching any of the output tags
- **logs:** the logs generated by EsoRex
- **warnings:** the warnings generated by EsoRex
- **errors:** the errors generated by EsoRex

4.5 PythonActor

The PythonActor executes custom python scripts. The interface between the actor and the script is the following:

- the tokens on the input ports will be passed to the script as command line parameters; SoFs and SoPs will be automatically converted to the simplified format (see APPENDIX A:);
- the outputs will be extracted from the script standard output, that must comply to a specific syntax.

In order to implement this interface the user has to include the module reflex.py and add a couple of statements to define inputs and outputs (see example.py in the Reflex distribution). Upon selecting a script, input and output ports are automatically created, together with a parameter for each output port, named portname_conversion_mode. If the user wants the actor to try to convert the outputs to SoFs or SoPs, the corresponding parameter (e.g.: PORT1_conversion_mode for port PORT1) must be set to either SoF or SoP; if the value is *don't convert* the output is not modified. In order for the outputs to be converted, they must follow the standard described in APPENDIX A:; furthermore all the files must exist on disk. The checksum information is also extracted from the file and added to the final SoF, therefore allowing the lazy mode to work for downstream actors.

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	14 of 21

The Reflex distribution includes the script `example.py`, which explains how to write python scripts that can be used from within the PythonActor. It's a simple script that has two input parameters (`input1` and `input2`) and two output parameters (`output1` and `output2`) and copies the value of the input parameters to the output parameters. The content of the script is listed and commented in APPENDIX B:. Every python script that is compatible with the PythonActor can be executed outside Reflex, provided that `reflex.py` is in the `PYTHONPATH`. In particular, you can execute `example.py` in the following way:

```
$ ./example.py -i /scratch/file1.fits -j /scratch/file2.fits
```

Or

```
$ ./example.py --input1 /scratch/file1.fits --input2 /scratch/file2.fits
```

In both cases the output will be the following:

```
ReflexParameter output1 "/scratch/file1.fits"
```

```
ReflexParameter output2 "/scratch/file2.fits"
```

Note that all parameters are passed to and from the script as strings.

This is a sample command line used by the PythonActor to call one of the scripts in the UVES workflow, formatted and simplified to improve its readability:

```
python /usr/bin/uves_cal_predict_interact.py
--in_sof=
/data/uves_cal_predict_1/2011-01-24T18:21:47.416/lineguesstable_redl.fits:LINE_GUESS_TAB_REDL,
/data/uves_cal_predict_1/2011-01-24T18:21:47.416/lineguesstable_redu.fits:LINE_GUESS_TAB_REDU,
/data/uves_cal_predict_1/2011-01-24T18:21:47.416/orderguesstable_redl.fits:ORDER_GUESS_TAB_REDL,
/data/uves_cal_predict_1/2011-01-24T18:21:47.416/orderguesstable_redu.fits:ORDER_GUESS_TAB_REDU
--enable=true --in_sop=uves_cal_predict:mbox_x=40,uves_cal_predict:mbox_y=40
```

4.5.1 Parameters

- **Run in terminal:** If true, then the script is invoked through an external xterm, which is required for scripts that write on the standard output or require standard input (e.g. some pyraf tasks). Note: since xterm masks the script exit code, if this flag is enabled the script execution is considered successful if the standard error is empty.
- **Python script:** the filename of the script to be executed (full path).

4.5.2 Ports

- **stdout:** the standard output of the python script, meaningful only if "Run in terminal" is false (hidden by default).

4.6 DataFilter

The DataFilter displays all the files in a given SoF and allows the user to view their header, view them in an external application, select or deselect only some of them, and then continue or pause the workflow.

4.6.1 Parameters

- **Mode:**
 - o Skip: automatically select all files and don't show any window.
 - o Display: select all the files and allow the user only to view them.
 - o Select: standard mode, it allows the user to view the headers of the files, to inspect the files with an external application and to select the files he wants to broadcast to the output port.
- **FITS Viewer:** The application used to inspect FITS files

4.6.2 Ports

- **in:** the input SoF
- **out:** the selected SoF
- **rejected:** the rejected SoF

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	15 of 21

4.7 SofCreator

The SofCreator reads all the FITS files contained in a directory and creates a SoF: the file category is based on the header keyword PRO.CATG or DO.CATG (Note: the latter is created by the DataOrganizer, and is not present in any file). If both are missing from the primary header then the Default Category is used.

4.7.1 Parameters

- **Default Category:** the default category to be used to classify FITS files if they do not possess any of the required keywords.

4.7.2 Ports

- **data dir:** the input directory
- **sof:** the output SoF

4.8 XMLFormatter

The XMLFormatter converts XML types used by ESO actors (i.e. LoSOs, SoF and SoP) into a human-readable form.

4.8.1 Parameters

- **Include Header and Footer:** if true then the output string contains a simple header and footer

4.8.2 Ports

- **in:** the input XML string
- **out:** a human-readable formatted string

4.9 SopCreator

The SopCreator converts a string to a SoP

4.9.1 Ports

- **in:** a string in the format described in Appendix APPENDIX A:
- **out:** a SoP

4.10 SofCombiner

The SofCombiner combines SoFs from different actors in a single SoF

4.10.1 Ports

- **sofs input:** the input SoFs
- **sof output:** a SoF containing all the input files

4.11 RecipeLooper

The RecipeLooper is an actor designed to implement looping capabilities within workflows. It has two set of input ports and a control port: at first it receives a token from the “standard” input ports, and then it receives a token from the looping input ports until the control port contains a string token whose value is “false”. When the control port contains a “true” the actor returns to the initial condition, waiting for the next token from sof in and sop in. You can find a more detailed description in chapter 5.

4.11.1 Ports

- **sof in:** the input files
- **sop in:** the initial recipe parameters
- **sof loop:** files to be used for iteration
- **sop loop:** parameters to be used in the current iteration

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	16 of 21

- **iteration complete:** control port
- **sof out:** output files
- **sop out:** output parameters to be used by the recipe executer

4.12 ProductRenamer

This actor renames files based on some FITS keywords and stores them in a destination directory.

It is usually connected to the output of recipes that produce final products. If the recipe produces some products that are not relevant to rename, an output filter can be configured in the RecipeExecuter.

The ProductRenamer is a simple script that can be easily modified to fine tune the names of the output file by manipulating the header keyword strings. Double-clicking on the actor in Reflex opens it up for editing. The script is in Jython, a Java implementation of the Python scripting language, with which it shares much of the syntax.

The script source code is saved in the workflow XML structure, so every change made by the user applies only to the specific instance of the actor he has modified.

4.12.1 Parameters

- **FinalProductDirectory:** the directory where the files are going to be copied/linked or renamed. It is usually set to END_PRODUCTS_DIR
- **SubDir:** if it is not empty, then a subdirectory will be created with this string. The value can contain slashes, in which case intermediate directories will be created. It will usually be END_PROD_SUBDIR, defined in the workflow
- **CopyMode:** there are several options: "copy", "move" or "link"
- **OutputExistMode:** it specifies the behavior in case the target file already exists. Possible values are:
 - o stop: the actor produces an error message and the workflows stops.
 - o append_version: the actor appends a suffix like _1, _2 to the filename.
 - o overwrite: the old file is overwritten.
- **Rename_keywords:** it specifies a set of FITS keywords that collated together will give the pattern to create the output filename. It is also possible to add fixed strings using 'my-string' format. For example, if the value of this parameter is 'MyPrefix_', HIERARCH.ESO.OBS.NAME, then the output filename will be, assuming that the value of the keyword is M51, MyPrefix_M51.fits.

4.12.2 Ports

- **input_files:** the files to be processed
- **renamed_files:** the final list of files produced

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	17 of 21

5. RECIPE ITERATION

It is sometimes useful to be able to visualize the products of a recipe execution, tweak some recipe parameter and execute the recipe again, until the products are as expected.

This can be easily achieved in Reflex by means of the RecipeLooper and the PythonActor.

A typical subworkflow that allows iteration is depicted in Figure 5, and it is composed of the following elements:

- Sof coming from previous actor
- Sop containing recipe parameters' initial values
- A RecipeLooper
- A RecipeExecuter containing the pipeline recipe to be optimized
- A PythonActor containing a custom python script that allows the user to view the results of the recipe execution and decide whether he wants to change some parameters or not. The script should either generate a "true" token on the control port and a SoF to the next downstream actor or a "false" token on the control port and a SoF and a SoP to the RecipeLooper loop input port.

This general structure must be customized for each pipeline recipe. The user must:

- Identify sensible recipe parameters he wants to tweak to optimize the products
- Define some initial conditions for these parameters and provide them to the sop_in port of the RecipeLooper. The simplest way to do this is to define a string in the format described in Appendix APPENDIX A: and connect it to a SopCreator.
- Configure the RecipeExecuter, changing the value of the recipe parameters you want to optimize to PORT (e.g. if you want to optimize a recipe parameter called `par1` look for a parameter in the RecipeExecuter called `recipe_param_nn` whose value is `par1=some_value` and change it to `par1=PORT`).
- Write a python script that allows the user to evaluate the product quality, change the value of the parameters and decide whether he wants to continue or not. The script does not have to be interactive, it can implement an optimization algorithm defined by the user.

Sample implementations of this system are provided with the workflows included in the pipeline distribution.

Note: it is possible to iterate over an arbitrary number of actors, you are not forced to iterate over one RecipeExecuter.

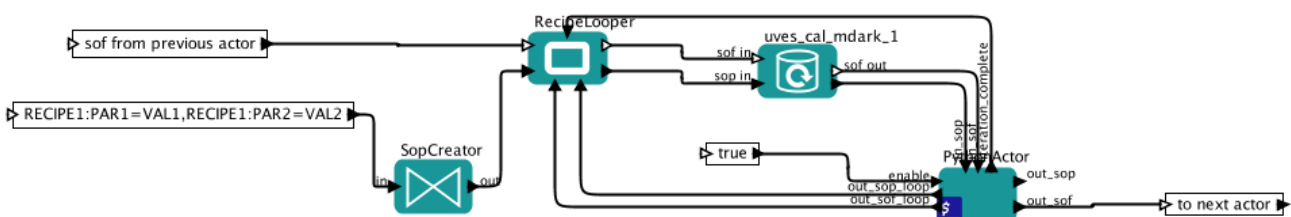


Figure 5: sample looping workflow

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	18 of 21

6. TROUBLESHOOTING

6.1 Another instance of Reflex is running

If the window in Figure 6 displays when you try to run Reflex, then there is already another running instance of Reflex. You can identify the PID with the following command:

```
$ ps aux|grep java|grep reflex|grep $USERNAME
```

and kill it if needed.

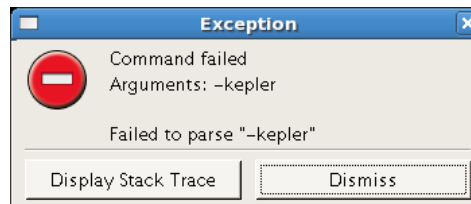


Figure 6

6.2 Errors during startup

During startup you may see some error messages related to an EcoGrid service: you can safely ignore them.

```
ERROR - org.ecoinformatics.seek.ecogrid.EcoGridServicesController - could not add service:The service
type is invalid or null: KeplerSlaveNode. Couldn't be added to list
```

6.3 Debug mode

By default, Reflex displays some information on the console: it is possible to change the level of logging for debugging purposes by editing the file `loader-for-kepler-1.0/src/log4j.properties` in the Reflex distribution.

At the bottom of the file you can find the logging configuration for Reflex specific actors, which by default is:

```
log4j.logger.org.eso=INFO
log4j.logger.org.eso.oqa=WARN
log4j.logger.org.eso.util=WARN
```

Meaning that the logging level of the actors is `INFO` and the logging level of the `oqa` and `util` libraries is `WARN`.

`Log4j`, the logging library used by Reflex, defines the following logging levels, starting from the most verbose: `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `FATAL`.

If you want to put all Reflex actors in debug mode change the first line as follows:

```
log4j.logger.org.eso=DEBUG
```

If you want, for example, to raise the logging level of the `RecipeExecuter` to `DEBUG`, then you have to add the following line:

```
log4j.logger.org.eso.RecipeExecuter=DEBUG
```

Note that the more restrictive condition is applied.

6.3.1 Logging to file

By default Reflex logs only to the console, so that if you want the logs to be saved also to a file, then you have to modify the first line of the configuration file (see 6.3) from

```
log4j.rootLogger=WARN, CONSOLE
```

to

```
log4j.rootLogger=WARN, CONSOLE, R
```

The log filename is defined in the section `LOGGING TO FILE` of the same configuration file.

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	19 of 21

APPENDIX A: SIMPLIFIED SOF AND SOP FORMAT

Some actors (i.e. the PythonActor and the SopCreator) use a simplified version of the SoF and SoP format in order to ease interaction with the user and with custom scripts. These formats are described here.

- SoF: FILE1:TYPE1,FILE2:TYPE2...
 - Example: UVES.2010-01-01T01:01:01.000.fits:DARK,UVES.2010-01-01T01:01:01.001.fits:FLAT
- SoP: RECIPE1:PAR1=VAL1,RECIPE2:PAR2=VAL2...
 - Example: uves_cal_mdark:process_chip=both,uves_cal_mflat:backsub.mmethod=median

In order to extract the information about the SoF in python scripts it is possible to use the function `parseSof()` from the `reflex.py` module: it returns a list of `FitsFile` objects, that contain the name and category of the file.

```
for file in parseSof(insof):
    print 'File name: ' + file.name ', File category: ' + file.category
```

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	20 of 21

APPENDIX B: example.py

```
#!/usr/bin/env python

# import reflex module
from reflex import *

import sys

if __name__ == '__main__':

    # create an option parser
    parser = ReflexIOParser()

    # define inputs (Note: you must define at least long option)
    parser.add_option("-i", "--input1")
    parser.add_option("-j", "--input2")

    # define outputs (Note: you must define at least long option)
    parser.add_output("-o", "--output1")
    parser.add_option("-p", "--output2")

    # get inputs from the command line
    (inputs, args) = parser.parse_args()
    # get output variables
    outputs = parser.get_outputs()

    # read inputs and assign outputs
    if inputs.input1 is not None:
        outputs.output1 = inputs.input1
    else:
        outputs.output1 = 'test1'

    if inputs.input2 is not None:
        outputs.output2 = inputs.input2
    else:
        outputs.output2 = 'test2'

    # print outputs
    parser.print_outputs()

    sys.exit()
```

ESO	Reflex User Manual	Doc:	VLT-MAN-ESO-19000-5037
		Issue:	2.0
		Date:	16/12/2010
		Page:	21 of 21

APPENDIX C: SOFTWARE REQUIREMENTS

Reflex has been tested on Scientific Linux 5.3 32bits plus java6 update 18: in principle every recent linux distribution should be compatible, but this can not be guaranteed.

Workflows may have specific requirements (e.g. python, pyfits, etc.): please refer to the workflow manual for more information.