



# The ESO Recipe Flexible Workbench EsoReflex

Enrique García Dabó, Andrea Modigliani,  
Artur Szostak, L. Coccato

[http://www.eso.org/sci/software/pipelines/reflex\\_workflows](http://www.eso.org/sci/software/pipelines/reflex_workflows)

# Overview

- Basic concepts
- Implementing a workflow with EsoReflex components
- EsoReflex python library functionality
- Frequent tasks to debug EsoReflex workflows



# EsoReflex Workflow Basics (1)

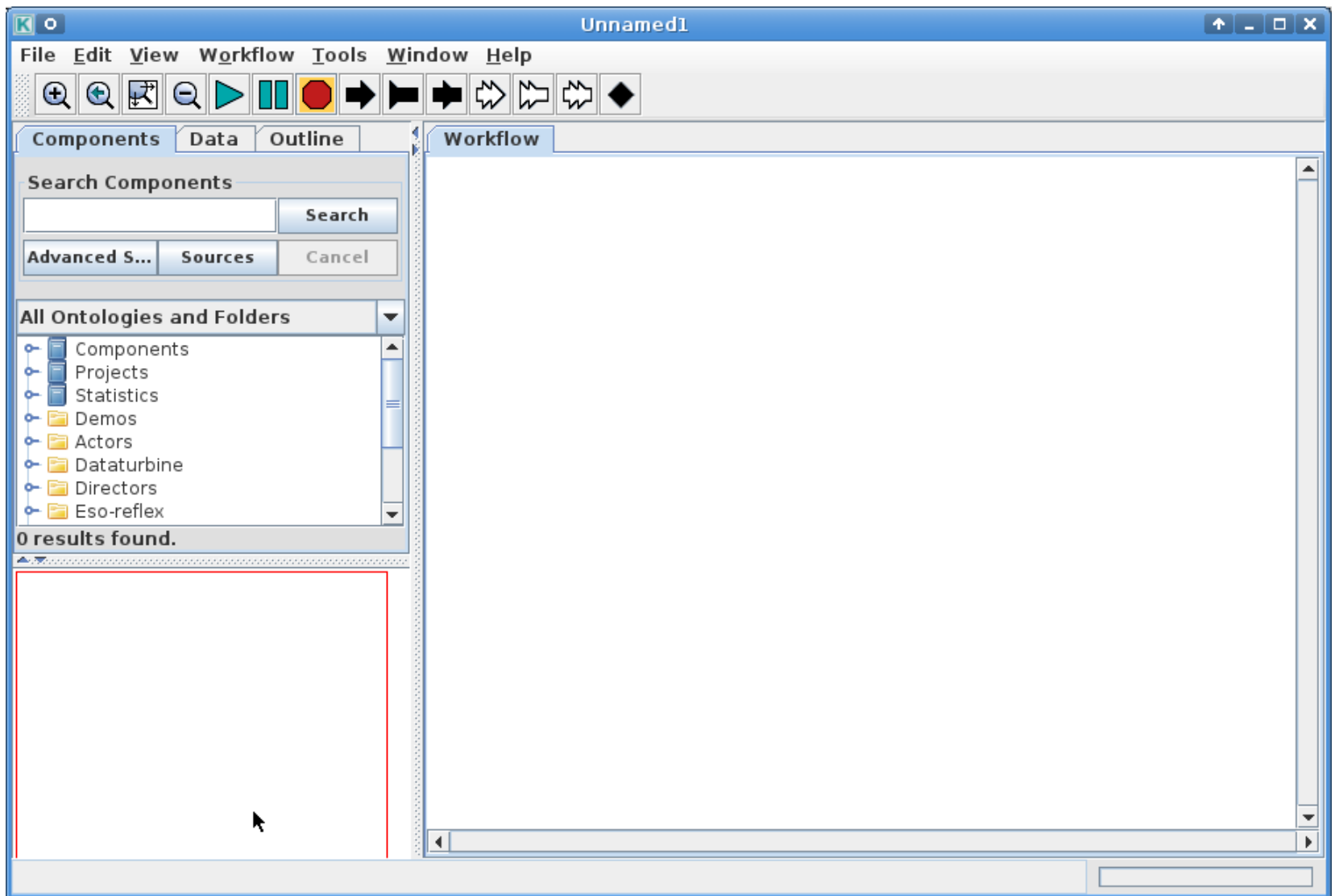
- A workflow is a set of connected actors that represent a pipeline data reduction cascade as encoded in OCA rules.
- Each actor is an independent step, e.g. recipe processing step, interactivity element, data source or data sink.
- Connections between actors indicate how output ports from one step are connected to input ports for the next.



# EsoReflex Workflow Basics (2)

- The underlying compute engine passes around tokens in sequence from one actor to the next over the connections.
- Tokens are associated with data and can have different types. The most common for Pipeline workflows are SOFs and SOPs.
- An actor will typically only execute when it has received one token on every input port, and will produce an output on all its output ports.

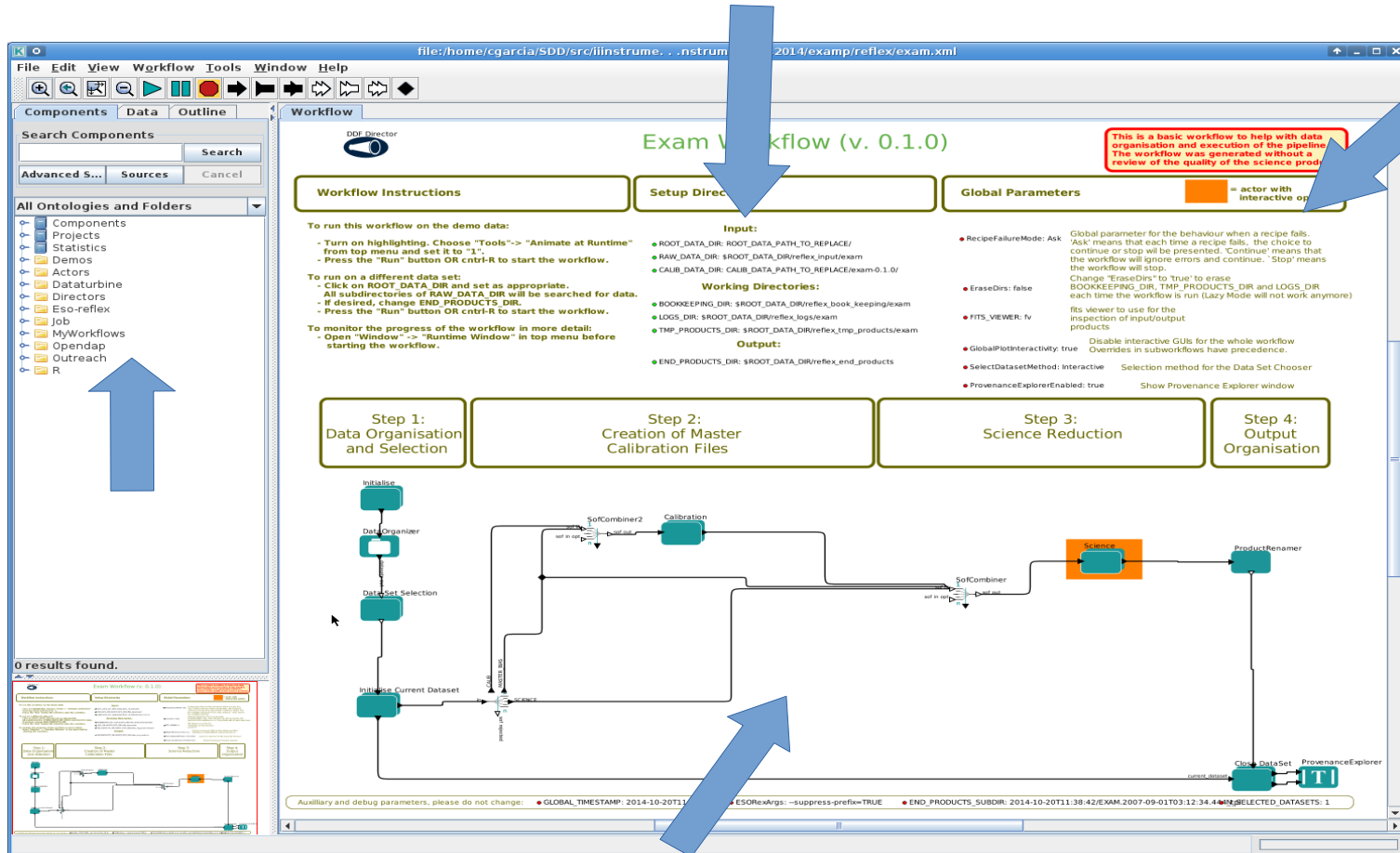
# EsoReflex empty canvas



# Typical workflow GUI layout

General Information

Main configuration



Canvas with main actors

# Designing a workflow (1)

- Steps previous to use Reflex:
  - Design **modular pipelines**: i.e. with recipes performing granular tasks. Avoid cases where I/O of the recipes depend on recipe parameters.
  - Think about the **supported observing modes**. Optional calibrations sometimes imply quite different reduction cascades.
  - Start with an **stable version of the pipeline** I/O and parameters.
  - Carefully design the FITS file categories.
  - Design the workflow layout in paper

# Designing a workflow (2)

- Steps using Reflex tool:
  - Create Data Organisation rules (OCA rules) which **mimic the workflow layout** as closely as possible
  - Use a **workflow template** (iiinstrument)
  - Create the graphical workflow layout
  - Skip interactivity development until the end.





# Data Organisation: OCA rules

- OCA rules are the mechanism used to perform **Data organisation** in a Reflex workflow
- OCA rules and workflow graphical layout are closely related.
- OCA rules are written in a text file with a given syntax.
- There are three types of rules:
  - **Classification**. Based on FITS keywords, it specifies the category of the file we have (“I am a raw flat”)
  - **Organization**. Files are grouped by keywords and each group triggers an action (“These flats create a master flat”)
  - **Association**. Each action can associate static calibrations or products created by other actions (“To create a master flat I need a master bias”)

# Data Organisation: OCA rules

- Classification:

```
if DPR.CATG like "%SCIENCE%" and DPR.TYPE like "%OBJECT%" then
```

```
{
```

```
    REFLEX.CATG = "RRRECIPE_DOCATG_RAW";
```

```
    REFLEX.TARGET = "T";
```

```
}
```

```
if DPR.CATG like "%CALIB%" and DPR.TECH like "%IMAGE%"
```

```
    and DPR.TYPE like "%STD%" then
```

```
{
```

```
    REFLEX.CATG = "RRRECIPE_CALIB_DOCATG_RAW";
```

```
}
```

# Data Organisation: OCA rules

- Organisation:

```
select execute(CALIB_IMG) from inputFiles where REFLEX.CATG == "RAW"  
      group by INS.FILT1.NAME, OBS.ID, OBS.TARG.NAME, TPL.START;
```

# Data Organisation: OCA rules

- Association:

```
action CALIB_IMG
```

```
{
```

```
minRet = 0; maxRet = 1;
```

```
select file as STATIC_MASK from calibFiles where REFLEX.CATG == "STATIC_MASK";
```

```
recipe rrecipe_calib;
```

```
product IMG_CALIBRATED { REFLEX.CATG = "IMG_CALIBRATED"; PRO.CATG = "IMG_CALIBRATED"; PRO.EXT="tpl_0000.fits";}
```

```
}
```

```
action COMBINE_IMG
```

```
{
```

```
minRet = 1; maxRet = 1;
```

```
select file as IMG_CALIBRATED from calibFiles where PRO.CATG == "IMG_CALIBRATED";
```

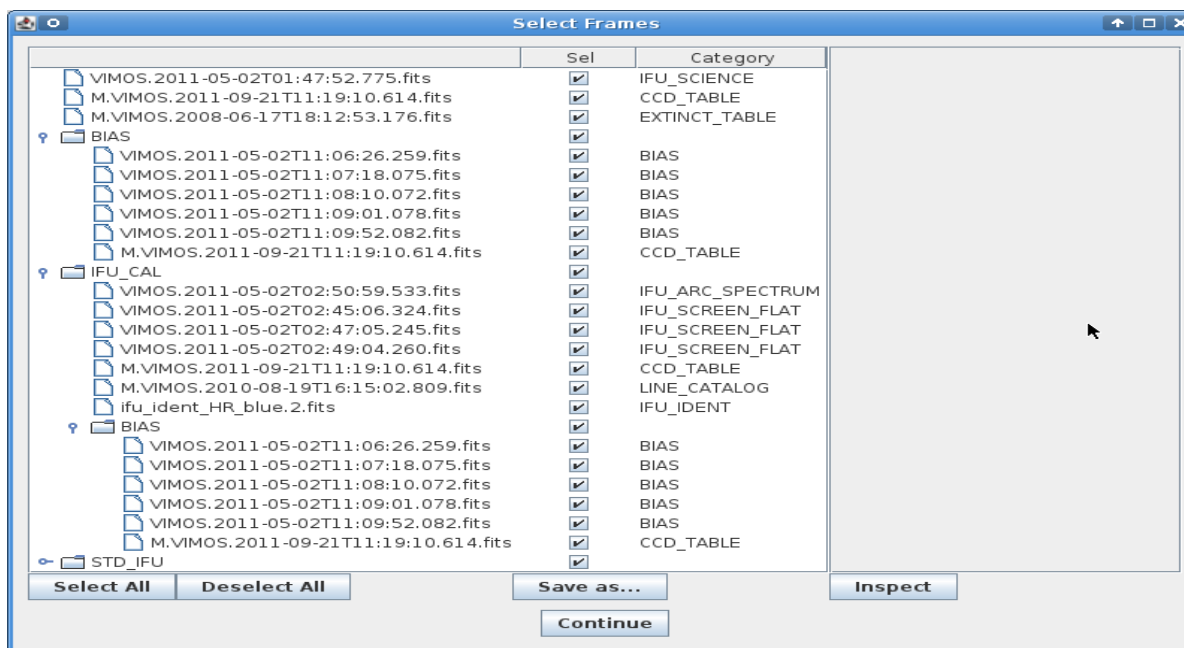
```
recipe rrecipe;
```

```
product IMG_OBJ_COMBINED { PRO.CATG = "IMG_OBJ_COMBINED";PRO.EXT="tpl_0001.fits";}
```

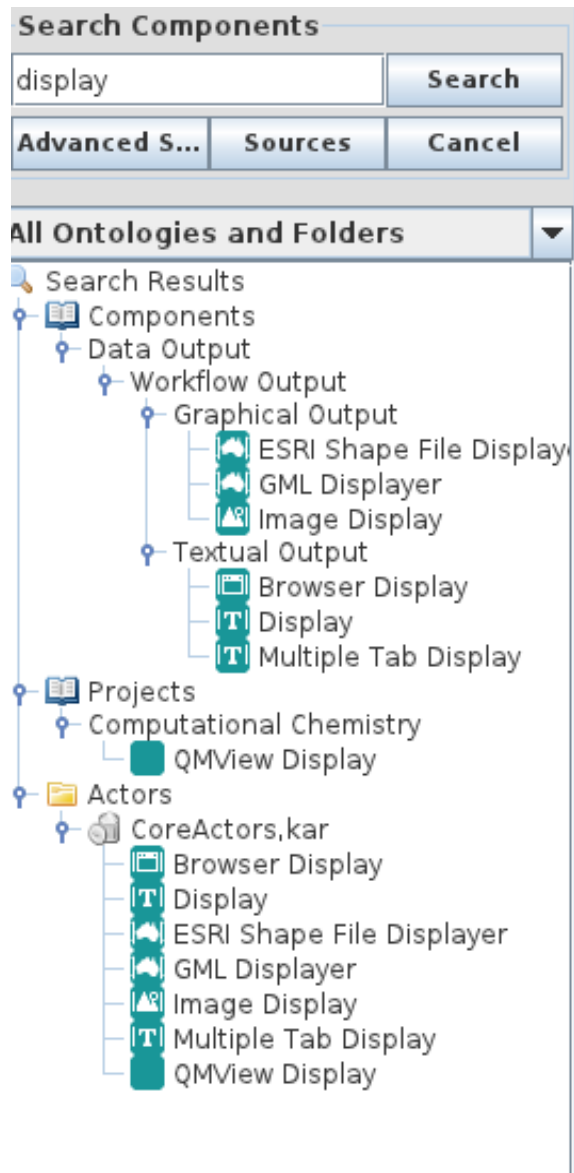
```
}
```

# File Purpose

- Category is not enough to determine the use of a file.
- The Purpose is the **role** a file will have in the **reduction cascade** => Path in the association tree
- A given file can have a list of purposes.
- For example, a raw BIAS might have purpose MASTERBIAS/SCIENCE if it's going to be used to produce the master bias that will be used to reduce the science frames.



# How to search for components



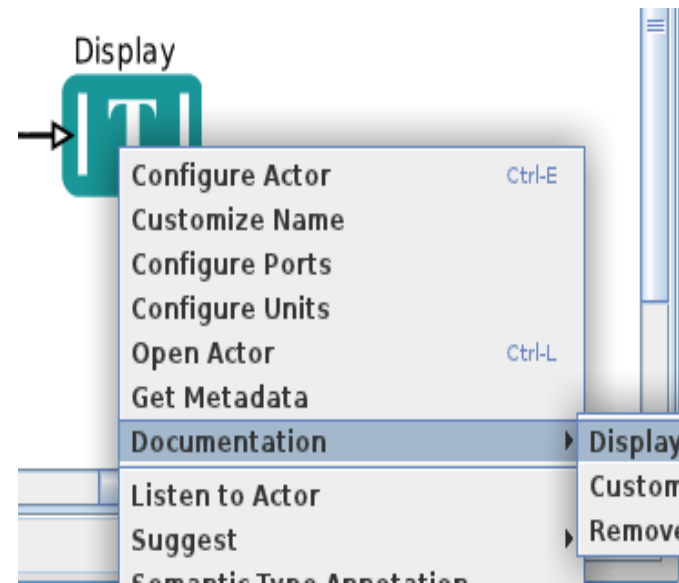
The search components utility allows to filter the available components by name.

The EsoReflex folder contains all the EsoReflex specific actors developed for astronomical reduction.

To use them, simply **drag and drop**.  
(except for recipe executer)



# Each actor comes with documentation



- The documentation explains which should be the functionality of the actor and the format used by the input and output ports.
- Actors documentation accessible already on results from a search
- Documentation is available for most actors

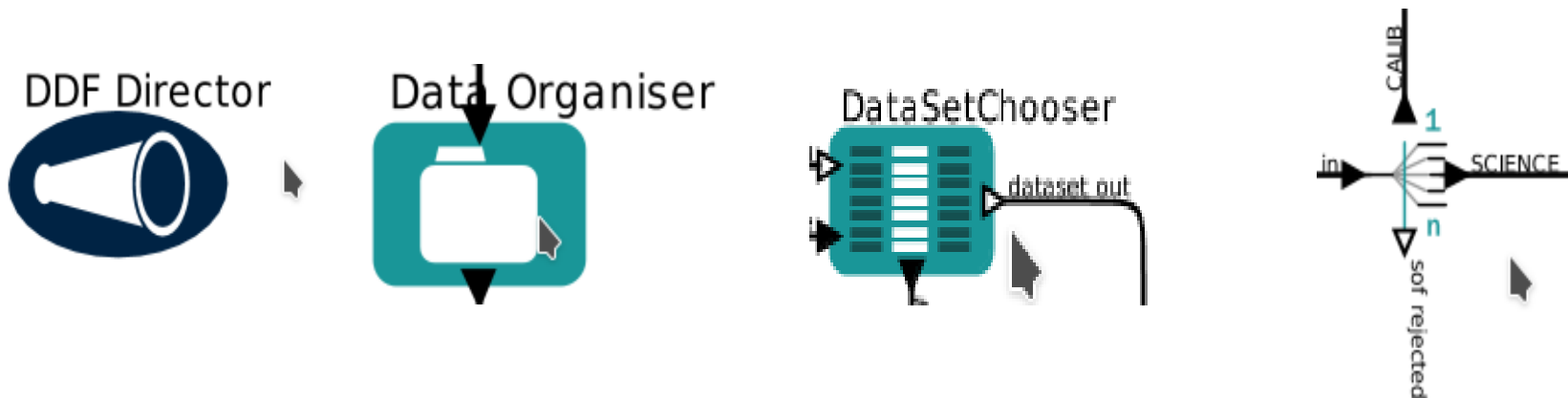
# Main EsoReflex components (1)

**DDF Director:** Drives the workflow engine

**DataOrganiser:** organise, classify, associate data

**DataSetChooser:** allows the user to select data sets

**FitsRouter:** sends data through different paths





# Main EsoReflex components (2)

**SofCombiner**: combines SOFs based on purposes.

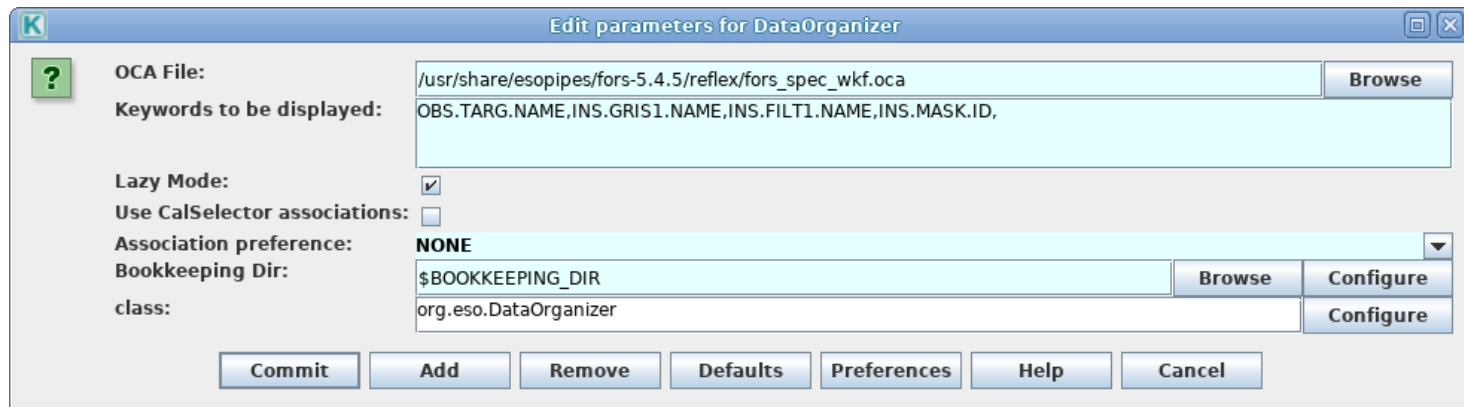
**ProductRenamer**: renames final products (based on relevant FITS keywords)

**DataFilter**: to select and inspect (if FITS\_VIEWER is set) data interactively



# Reflex Actors: DataOrganizer

- The **DataOrganizer** is responsible of giving each input file a category and purpose.
- It is configured by means of a OCA rules file

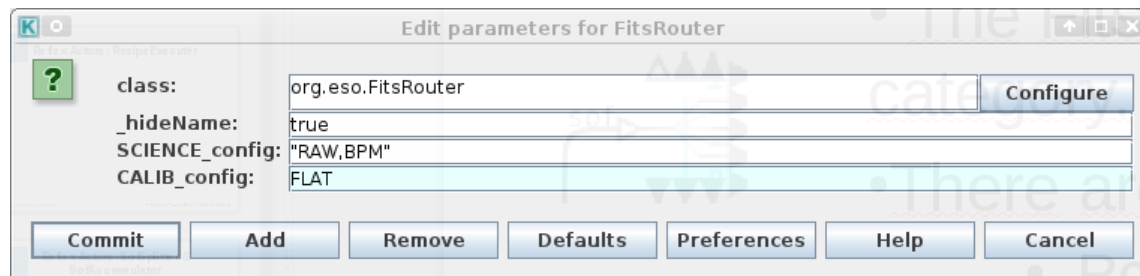
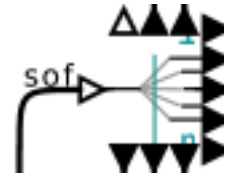


# DataOrganizer Lazy Mode

- **Lazy mode** for **DataOrganizer**.
  - It avoids the organization of all the data in subsequent workflow runs.
  - It checks whether the OCA rules and the input files are the same.

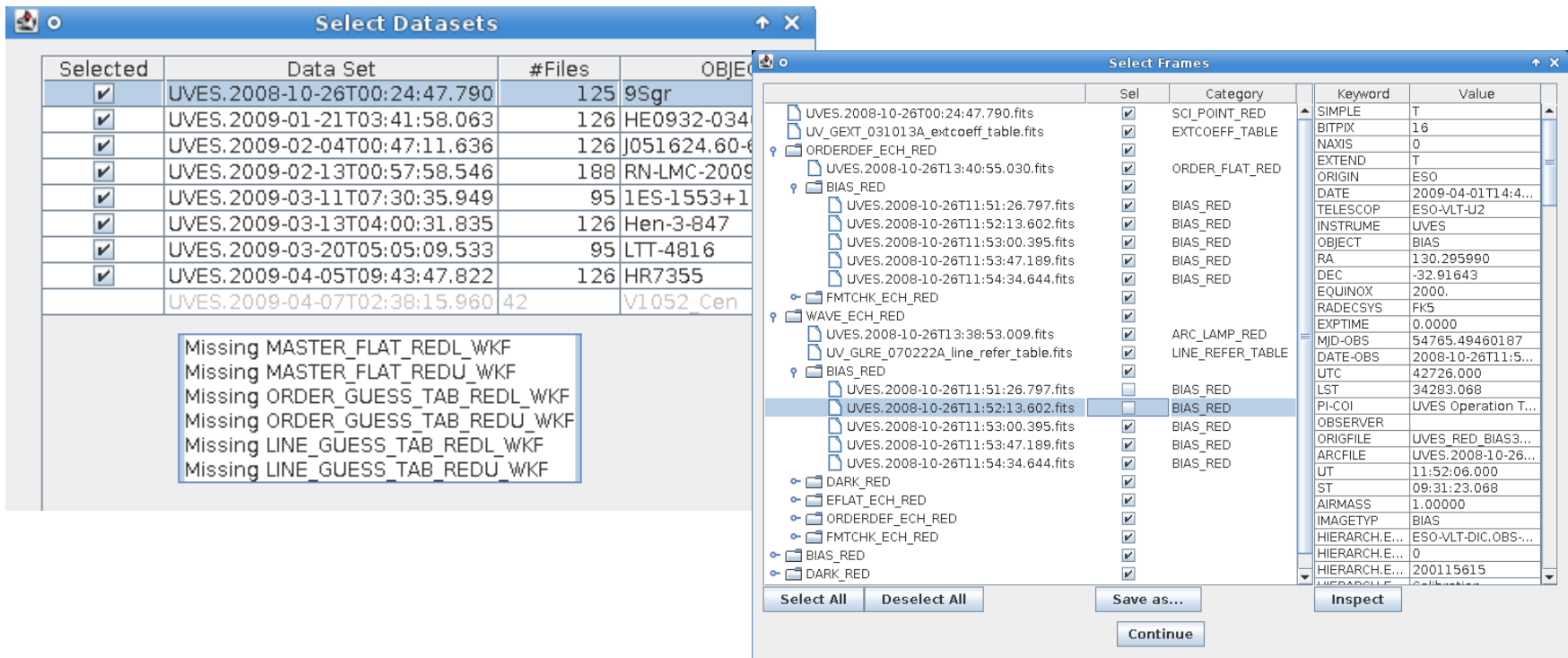
# Reflex Actors: FitsRouter

- The **FitsRouter** will split the input by category. There are two modes:
  - Routing by category explicitly. Just create a port with the name of the category.
  - Configuring a port. Create a configuration parameter with the name PORT\_config and list the desired categories.
- The „sof\_rejected“ ports can be a usefule debug tool



# Reflex Actors: DataSetChooser

- It allows to select the Datasets to reduce
- Datasets can be inspected and the calibration cascade will be shown.
- Purposes can be seen as the path to the leaf in the tree
- If a Dataset is incomplete, it will show which are the missing files.



The screenshot shows two windows from the Reflex Actors software. The 'Select Datasets' window on the left displays a table of datasets with columns for 'Selected', 'Data Set', '#Files', and 'OBJECT'. The 'Select Frames' window on the right shows a tree view of dataset frames with columns for 'Sel', 'Category', 'Keyword', and 'Value'.

**Select Datasets Table:**

Selected	Data Set	#Files	OBJECT
<input checked="" type="checkbox"/>	UVES.2008-10-26T00:24:47.790	125	9Sgr
<input checked="" type="checkbox"/>	UVES.2009-01-21T03:41:58.063	126	HE0932-034
<input checked="" type="checkbox"/>	UVES.2009-02-04T00:47:11.636	126	J051624.60-4
<input checked="" type="checkbox"/>	UVES.2009-02-13T00:57:58.546	188	RN-LMC-2009
<input checked="" type="checkbox"/>	UVES.2009-03-11T07:30:35.949	95	1ES-1553+1
<input checked="" type="checkbox"/>	UVES.2009-03-13T04:00:31.835	126	Hen-3-847
<input checked="" type="checkbox"/>	UVES.2009-03-20T05:05:09.533	95	LTT-4816
<input checked="" type="checkbox"/>	UVES.2009-04-05T09:43:47.822	126	HR7355
<input checked="" type="checkbox"/>	UVES.2009-04-07T02:38:15.960	42	V1052_Cen

Missing files listed below the table:

- Missing MASTER\_FLAT\_REDL\_WKF
- Missing MASTER\_FLAT\_REDUC\_WKF
- Missing ORDER\_GUESS\_TAB\_REDL\_WKF
- Missing ORDER\_GUESS\_TAB\_REDUC\_WKF
- Missing LINE\_GUESS\_TAB\_REDL\_WKF
- Missing LINE\_GUESS\_TAB\_REDUC\_WKF

**Select Frames Table:**

Sel	Category	Keyword	Value
<input checked="" type="checkbox"/>	SCI_POINT_RED	SIMPLE	T
<input checked="" type="checkbox"/>	EXTCOEFF_TABLE	BITPIX	16
<input checked="" type="checkbox"/>		NAXIS	0
<input checked="" type="checkbox"/>		EXTEND	T
<input checked="" type="checkbox"/>	ORDER_FLAT_RED	ORIGIN	ESO
<input checked="" type="checkbox"/>		DATE	2009-04-01T14:4...
<input checked="" type="checkbox"/>		TELESCOP	ESO-VLT-U2
<input checked="" type="checkbox"/>	BIAS_RED	INSTRUME	UVES
<input checked="" type="checkbox"/>		OBJECT	BIAS
<input checked="" type="checkbox"/>		RA	130.295990
<input checked="" type="checkbox"/>		DEC	-32.91643
<input checked="" type="checkbox"/>		EQUINOX	2000.
<input checked="" type="checkbox"/>		RADECSYS	FK5
<input checked="" type="checkbox"/>		EXPTIME	0.0000
<input checked="" type="checkbox"/>		MJD-OBS	54765.49460187
<input checked="" type="checkbox"/>		DATE-OBS	2008-10-26T11:5...
<input checked="" type="checkbox"/>		UTC	42726.000
<input checked="" type="checkbox"/>		LST	34283.068
<input checked="" type="checkbox"/>		PI-COI	UVES Operation T...
<input checked="" type="checkbox"/>		OBSERVER	
<input checked="" type="checkbox"/>		ORIGFILE	UVES_RED_BIAS3...
<input checked="" type="checkbox"/>		ARCFILE	UVES.2008-10-26...
<input checked="" type="checkbox"/>		UT	11:52:06.000
<input checked="" type="checkbox"/>		ST	09:31:23.068
<input checked="" type="checkbox"/>		AIRMASS	1.00000
<input checked="" type="checkbox"/>		IMAGETYP	BIAS
<input checked="" type="checkbox"/>		HIERARCH.E...	ESO-VLT-DIC.OBS...
<input checked="" type="checkbox"/>		HIERARCH.E...	0
<input checked="" type="checkbox"/>		HIERARCH.E...	200115615



# Reflex Actors: RecipeExecutor

- It executes recipes using esorex
- The RecipeExecutor has to be instantiated explicitly:  
*Tools -> Instantiate Component.* It needs to have the recipe installed (i. e. it is shown in *esorex –recipes*).
- For each recipe parameters, an actor parameter is created with the name “*recipe\_param\_nn*”, where *nn* corresponds to the parameter order.
- If recipe parameters change, the RecipeExecutor should be reinstantiated.



# RecipeExecutor: Recipe parameters

recipe_param_1:	debug=FALSE
recipe_param_2:	plotter=no
recipe_param_3:	process_chip=both
recipe_param_4:	mbox_x=PORT
recipe_param_5:	mbox_y=PORT
recipe_param_6:	trans_x=PORT
recipe_param_7:	trans_y=PORT
recipe_param_8:	ech_angle_off=0.0
recipe_param_9:	cd_angle_off=0.0

Some parameters are set to PORT and changed via the sop port. Other parameters however have fixed names and values.



# RecipeExecutor: Other configurations

recipe:	espdr_mdark
mode:	Run
Lazy Mode:	<input checked="" type="checkbox"/>
Recipe Failure Mode:	\$RecipeFailureMode
Input Files Category:	DARK, CCD_GEOM, INST_CONFIG, MASTER_BIAS_RES
Output Files Category:	HOT_PIXEL_MASK, MASTER_DARK

- Using input filters one can select the file categories that are passed to the recipe: DARK, MASTER\_BIAS\_RES, ...
- With output filters one can select the file categories that are broadcast to the sof\_out: MASTER\_DARK, HOT\_PIXEL\_MASK





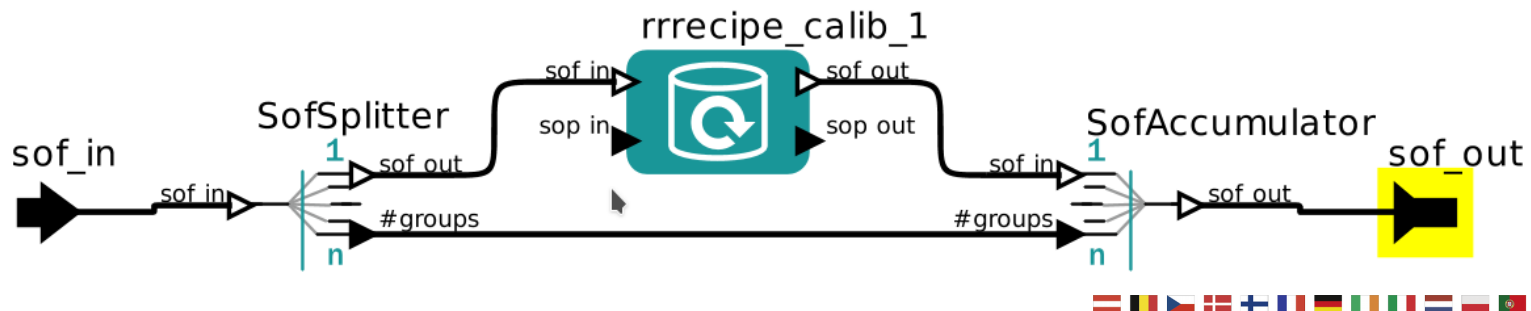
# RecipeExecutor: Lazy Mode

- **Lazy mode for RecipeExecutor**

- It works by comparing the input of the current execution with *all* the previous recipe executions:
  - All files must be the same
  - All files must have the same checksum
  - All files must have the same date
  - All recipe parameters must be the same
- If a recipe at the beginning of the workflow is set to *Not-Lazy mode*, the input of the next recipes will be new and lazy mode will not be triggered.

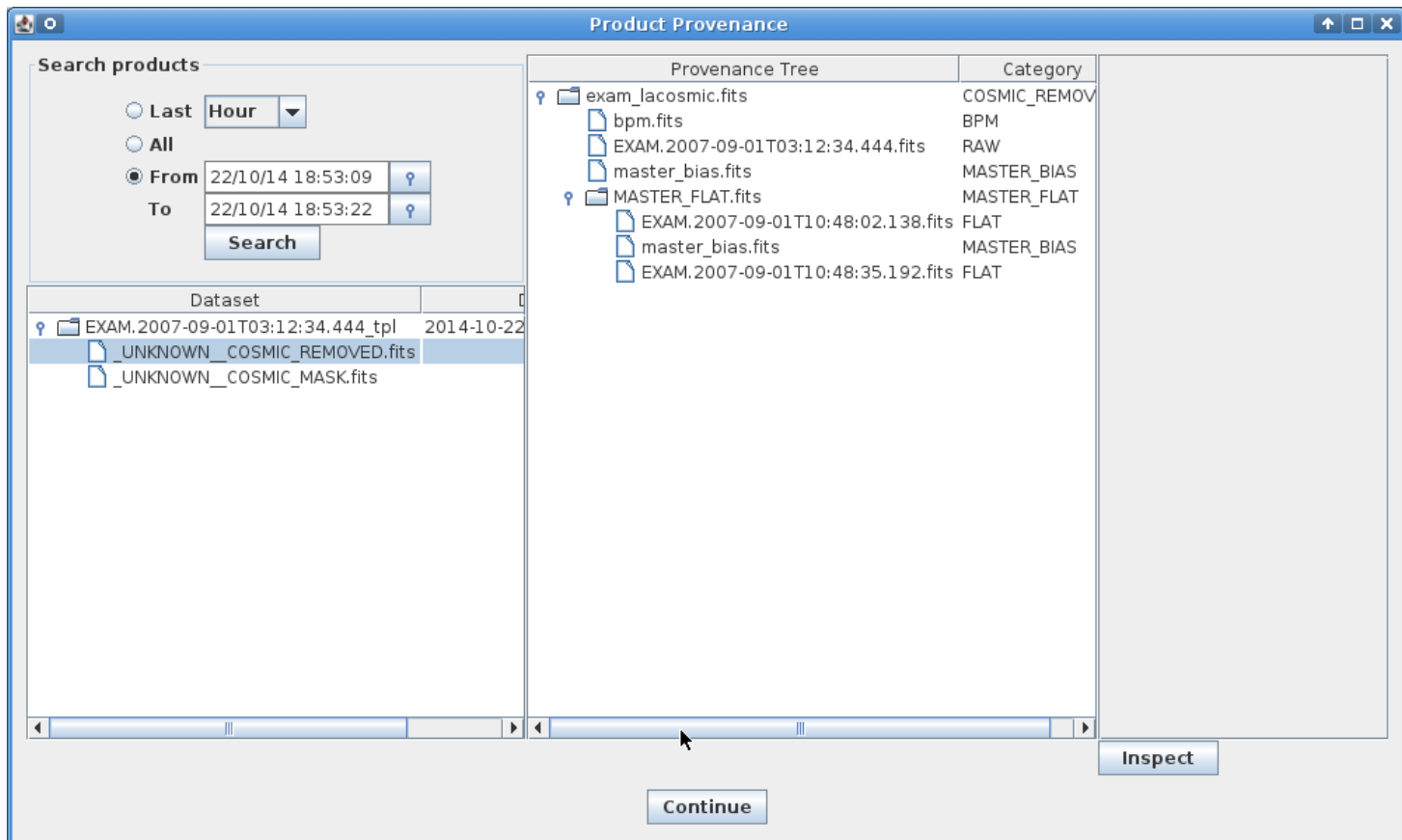
# Recipe actors: SofSplitter & SofAccumulator

- In order to handle properly Purposes, the RecipeExecuter's have to be enclosed between SofSplitter/SofAccumulator.
- SofSplitter will create as many groups as different purposes are in the input and will trigger the RecipeExecuter for each of the purpose groups.
  - For example: Biases that are needed to calibrate the science (purpose=bias/science) will be processed separately from the biases needed to process the standard star (purpose = bias/standard)
- SofAccumulator simply combines back all the products in a single channel.



# Recipe actors: ProductExplorer

- It displays the effective reduction cascade used to create a given product.

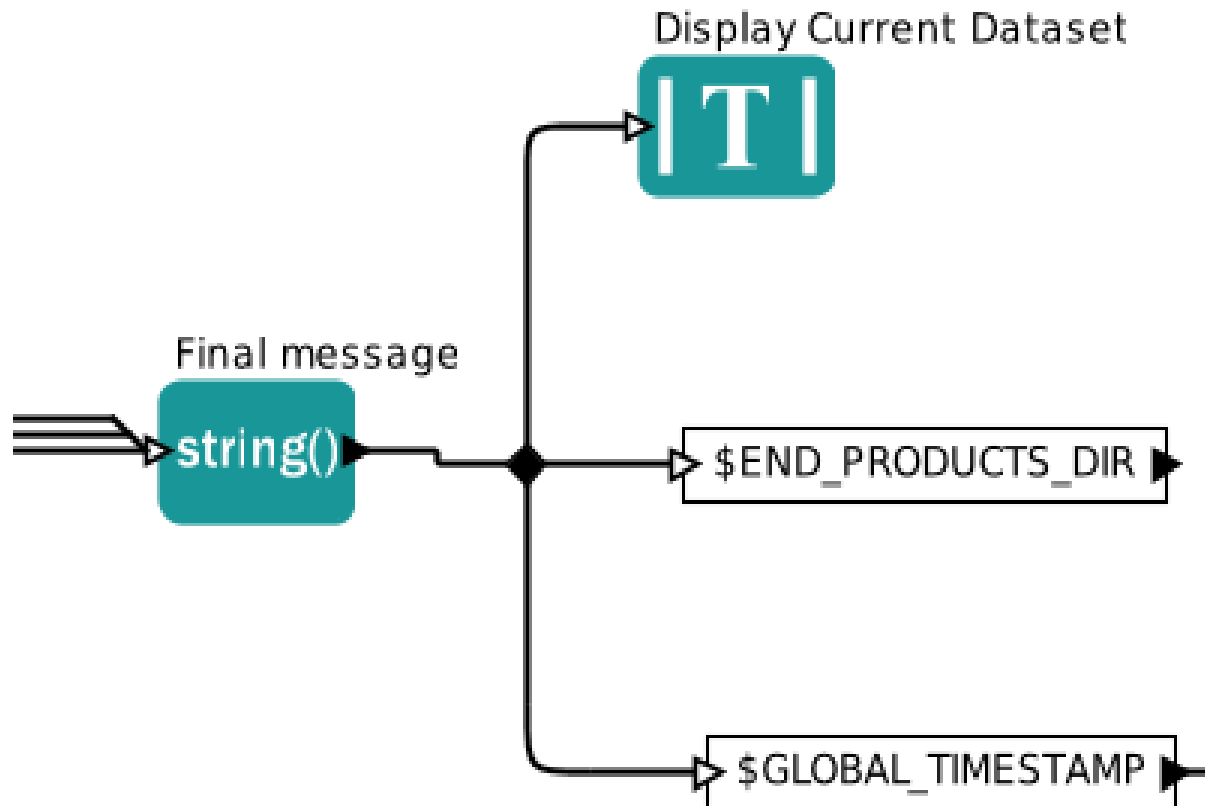


# Reflex actors: SofCombiner


- The SofCombiner takes as input a number of set of files and combines them into one set of files that contains only the files whose purposes are present in all the input tokens.
- With this respect empty tokens are ignored and files with universal purpose are always collected.
- Not to filter out inputs with different purposes, use option “ignore purposes” (or use sof in opt).

# The diamond

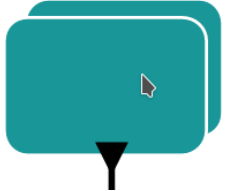
To connect the output of a single port to several actors, the diamond will send the output to several connections



# Reflex actors: CompositeActor

- It is used to create nested workflows.
- In the standard workflows there are quite a number of composite actors
- The inner subworkflow can see the variables from the top level one
- The ports of the CompositeActor in the upper workflow appear as floating sinks or sources: 

Initialise



Inspect previously reduced data



Initialise Current Dataset



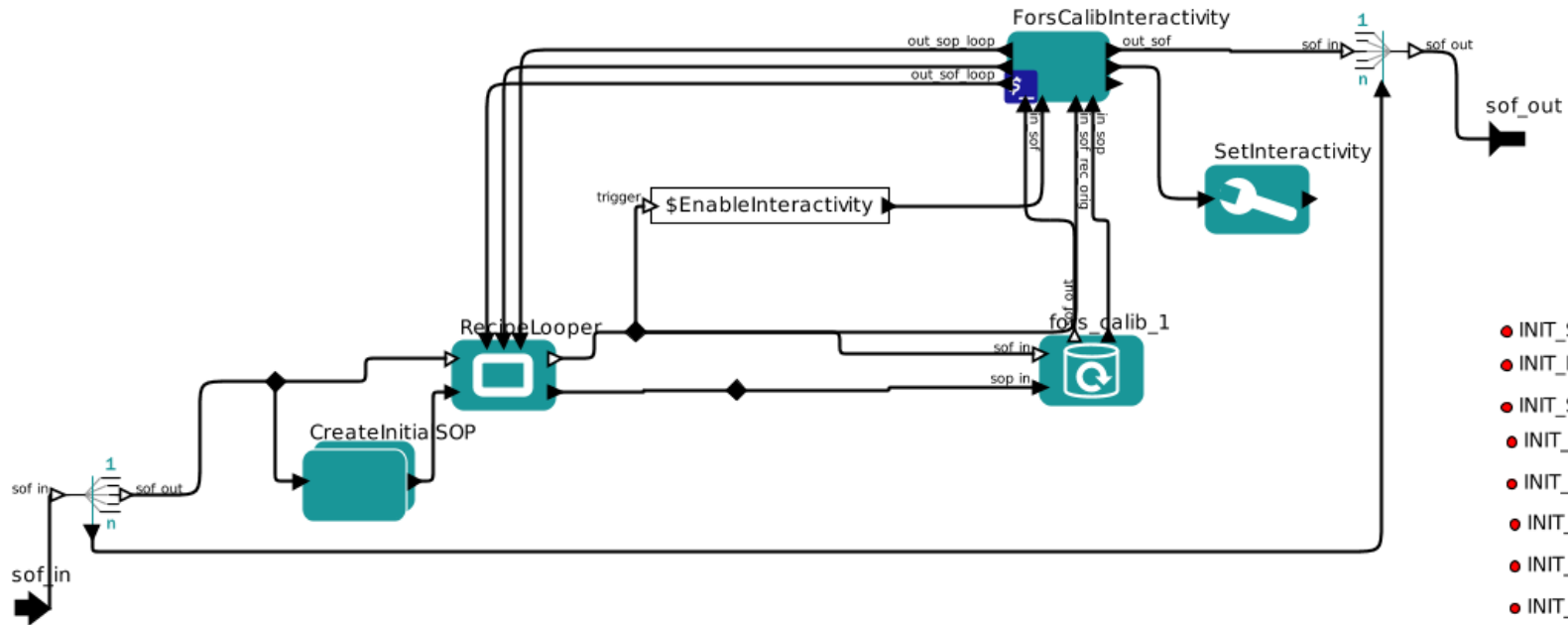
ProductExplorer



Close DataSet



# Reflex actors: RecipeLooper



● EnableInteractivity: \$GlobalPlotInteractivity

- INIT\_SRADIUS: 10.0
- INIT\_DRADIUS: 10.0
- INIT\_S\_NKNOTS: -1
- INIT\_D\_NKNOTS: -1
- INIT\_SPLFIT\_THRESHOLD: 0.01
- INIT\_STACK\_METHOD: sum
- INIT\_MINREJECTION: 1
- INIT\_MAXREJECTION: 1
- INIT\_KITER: 999
- INIT\_KLOW: 3.0
- INIT\_KHIGH: 3.0
- INIT\_WREJECT: 0.7
- INIT\_WMODE: 2
- INIT\_WMOSMODE: 0
- INIT\_WRADIUS: 4



# Reflex actors: PythonActor (1)

- It is able to execute generic python code.
- To translate from/to Reflex ports to/from python script arguments a special syntax is used, with the help of a Python module
- To create a python actor, use the menu *Tools->Instantiate Component* and type *org.eso.PythonActor*.
- The script must **import reflex.py** module
- **Stand-alone execution** is possible (executing cmdline.sh in proper dir)





# Reflex actors: PythonActor (2)

- A python script can be “reflexed” using the following syntax:

```
from reflex import *
```

```
parser = ReflexIOParser()
```

```
#Define inputs
```

```
parser.add_option("-i", "--in_sof", dest="in_sof")
```

```
#Define outputs
```

```
parser.add_output("-o", "--out_sof", dest="out_sof")
```

Importing Reflex

Define Inputs/Outputs

```
(inputs, args) = parser.parse_args()
```

```
outputs = parser.get_outputs()
```

```
#Set the output
```

```
outputs.out_sof = inputs.in_sof
```

Getting inputs

Setting outputs

```
parser.print_outputs()
```

```
sys.exit()
```



# Python Framework

- Reflex is delivered with a python library that helps to create interactive user interface rather easy using matplotlib and wxPython.
- The **reflex\_interactive\_app** module is a framework to create interactive windows.
- The **pipeline\_product** module eases the reading of FITS images, spectra and tables, using pyfits.
- The **pipeline\_display** module produces **scatter plots**, **image displays** and **spectra plots** using matplotlib.
- The **reflex\_plot\_widgets** module allows to add extra interactivity within the matplotlib plots.
- These modules are accessible directly when using a python script inside the PythonActor.

# Python Framework

- Example of using reflex\_interactive\_app framework:

```
def setInteractiveParameters(self):
    return [reflex.RecipeParameter(recipe=rec_name, displayName="par1", group="Limits")]
```

Defining the recipe parameters to show

```
def readFitsData(self, fitsFiles):
    self.frames = dict()
    for f in fitsFiles:
        self.frames[f.category] = PipelineProduct(f)
    pro_raw = self.frames["MY_CAT"]
    pro_raw.readImage()
```

Reading the FITS Data from the input

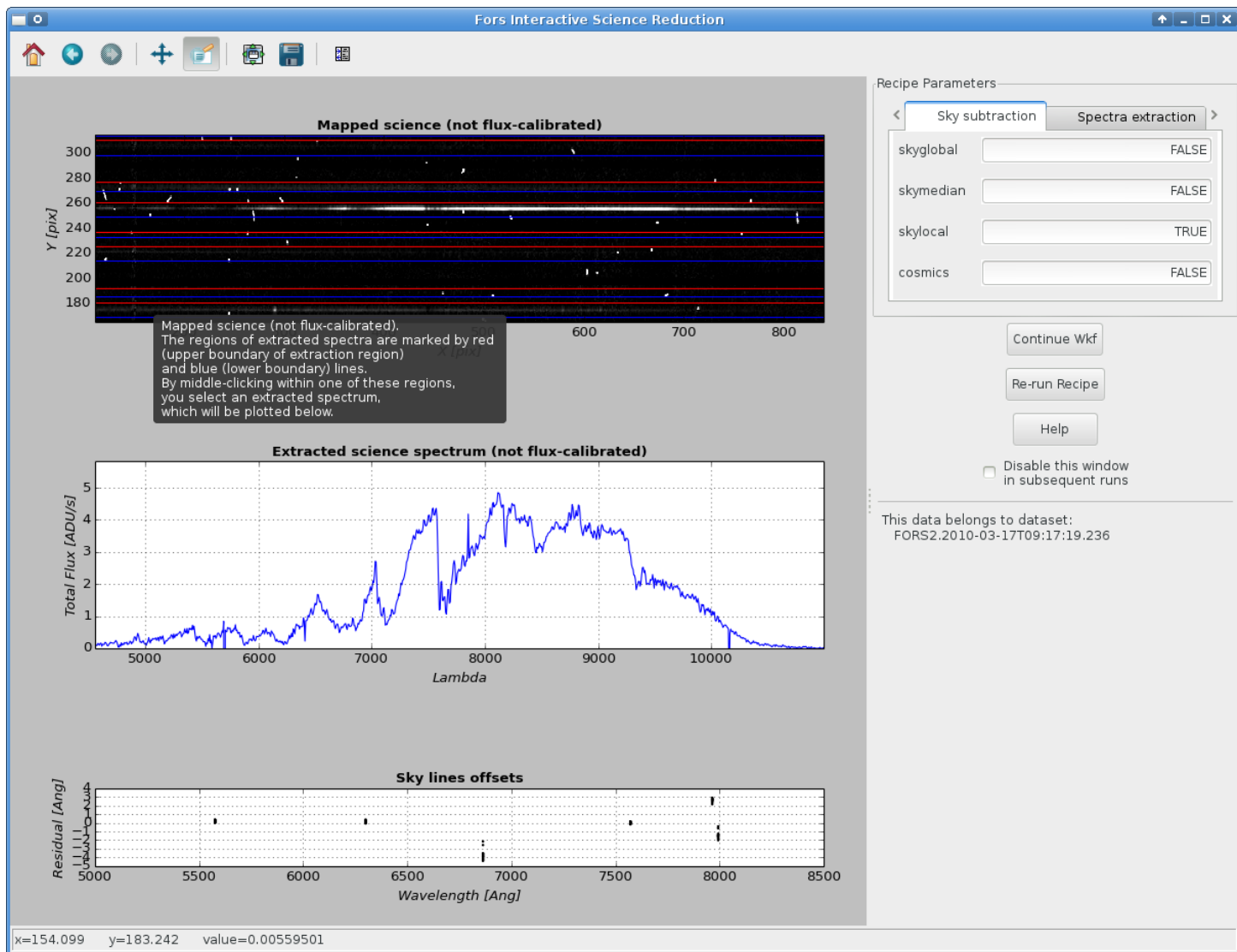
```
def addSubplots(self, figure):
    self.raw_plot = figure.add_subplot(111)
```

Defining the plot layout

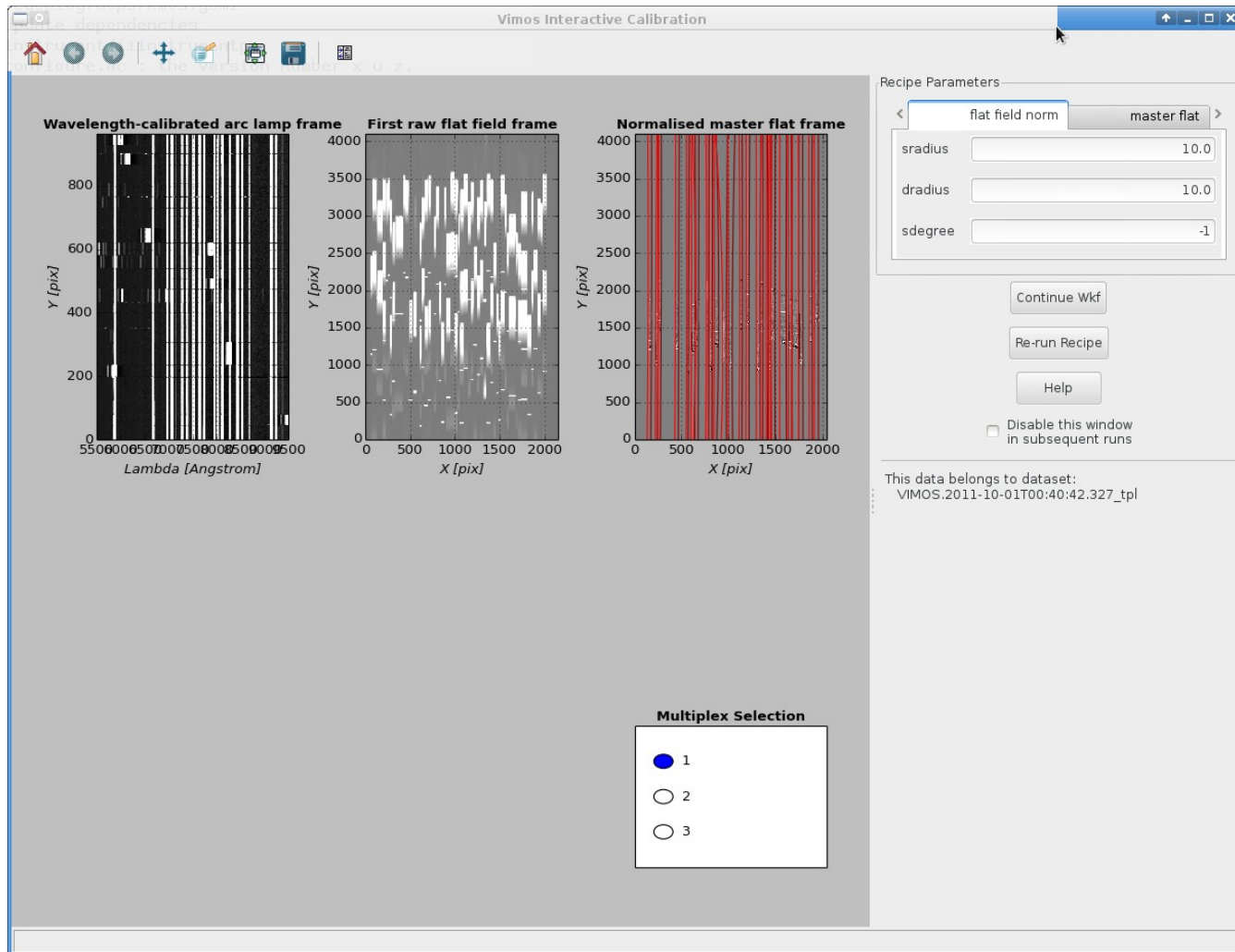
```
def plotProductsGraphics(self):
    img_raw_disp = pipeline_display.ImageDisplay()
    img_raw_disp.setLabels('X', 'Y')
    tooltip = paragraph("Raw image")
    img_raw_disp.display(self.raw_plot, "Raw image", tooltip, pro_raw.image)
```

Plotting the data

# Examples interactive windows

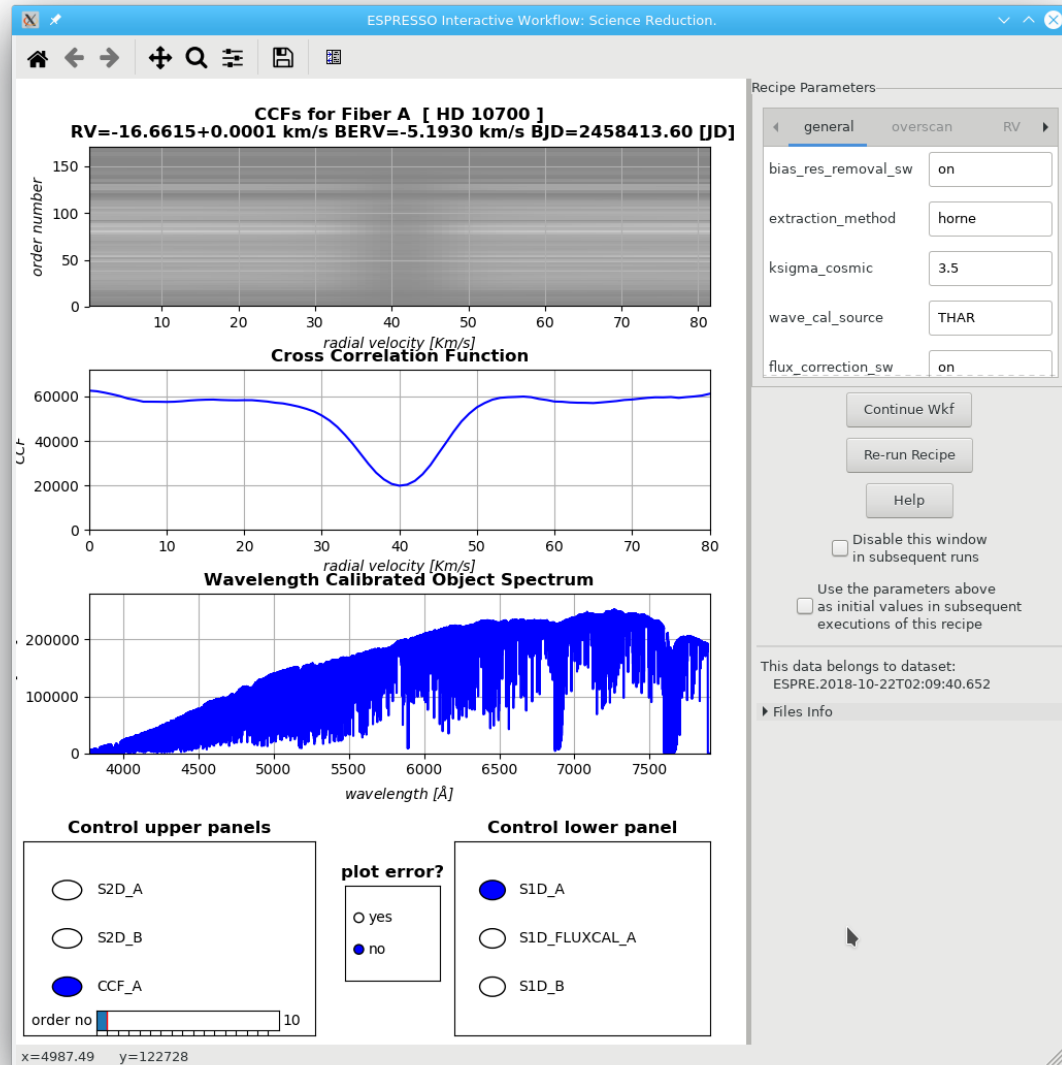


# Examples interactive windows





# Examples interactive windows





# Debugging workflows



# How to verify/debug OCA rules

- Check for typos in FITS keywords used to classify, associate, constrain data.
- Have data for which REFLEX.TARGET is defined
- Grey data sets:
  - 1) Pointing mouse give information (missing PRO.CATG).
  - 2) search in the OCA rules which recipe creates it,
  - 3) Search what raw data triggers that recipe,
  - 4) Add missing data to RAW\_DATA\_DIR/CALIB\_DATA\_DIR.





# How to verify/debug OCA rules

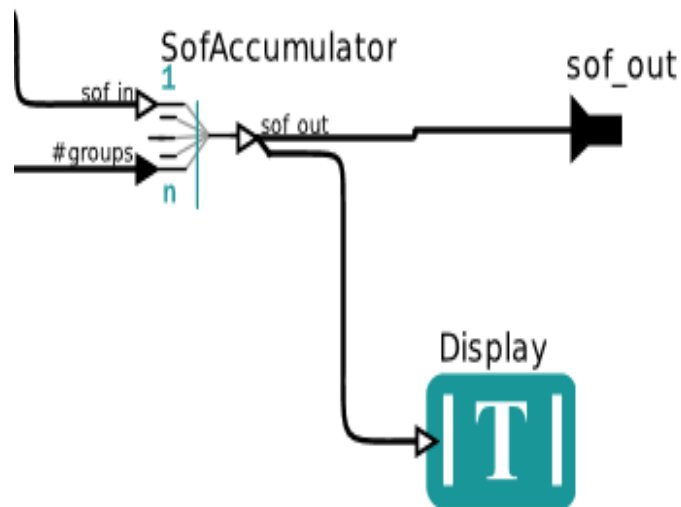
- To inspect the data set can help
- The OCA rules define the reduction chain. They define the workflow layout.
- Make sure that RAW data types and PRO.CATGs are unique.
- You may use minRet/maxRet to constrain trigger of events and associate master calibrations.
- You may define RAW TYPES to group data types together

# Debugging using the text display

Using a text display is the very useful way to inspect a workflow and see the information which travels through it.

The text display will pop up a window the first time it is triggered.

MonitorValue actor may be used to display most recently received token value





# Debugging a recipe (or python script)

- A recipe/python-script may fail due to several cases: wrong input data/parameters, sw implementation bugs...
- To catch a recipe failure be sure to set the recipe failure mode to “Ask” or “Stop”.

Recipe Failure Mode:	<input type="text" value="\$RecipeFailureMode"/>
Input Files Tag:	Continue
Output Files Tag:	Stop
File Purpose Processing:	Ask

- debugging by using command line:
  - **cd reflex\_book\_keeping/<instrume>/recipe\_name**
  - execute: **./cmdline.sh**

You may set “Run on terminal” option to debug a Python actor.



# Debugging: look at EsoReflex directories

Bookkeeping	Input/output SOF. Recipe parameters.DataOrganization Interactive workflows
Logs	Recipe output logs
Tmp_Products	Recipe execution products (FITS files)
End_Products	Final scientific products with meaningful names

## ➤ Directors?