



EUROPEAN SOUTHERN OBSERVATORY

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

VERY LARGE TELESCOPE

How to include Python scripts into esoreflex workflows.

VLT-MAN-ESO-xxxx

Issue 3.0

Date 2022 06 12

Prepared: L. Coccato 2022 12 06
Name Date Signature

Approved: -
Name Date Signature

Released: -
Name Date Signature

This page was intentionally left blank

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	3 of 25

Change record

Issue/Rev.	Date	Section/Parag. affected	Reason/Initiation/Documents/Remarks
1.0	01-04-2020	All	First official release
2.0	23-10-2020	All	Minor changes and different colors adopted
3.0	06-12-2022	1	Reference to instructions for Python recipes added

This page was intentionally left blank

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	5 of 25

Contents

1	Introduction and scope	7
2	How to insert a PythonActor in an esoreflex workflow	8
3	How to execute Python scripts in EsoReflex: the PythonActor	10
3.1	The PythonActor	10
3.2	E1. Process files independently	12
3.3	E2. Process files simultaneously, replace a pipeline recipe	15
3.4	E3. Set Python script parameters via the workflow.	18
3.5	Updating the Provenance Database	21

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	6 of 25

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	7 of 25

1 Introduction and scope

This document explains how to include and execute Python scripts within `esoreflex` workflows, the environment to run instrument pipelines. The described procedures apply to any Python script that reads and writes FITS files. The steps described here provide a quick and easy way to establish the communication between `esoreflex` and such a script. This tutorial does not cover the development of full fledged recipes with Python, which is described in the `Esoflex` user manual (see <https://www.eso.org/sci/software/esoreflex/>).

The advantage of a Python script is that it requires less effort for the interface with `Esoflex`, but it does not easily offer the full flexibility of a recipe (e.g., expose parameters to the user and keep record in the provenance explorer.)

We assume that the reader is familiar with the basic concepts of pipelines, pipeline workflows, `esoreflex`, and Python. Here we simply summarize some concepts. For comprehensive information we refer the reader to the user manuals and tutorials that are available at: <http://www.eso.org/sci/software/pipelines/>

Each VLT instrument comes with a “pipeline”, a set of CPL-based programs designed to reduce the data coming from that instrument. Each pipeline is composed of individual “recipes”, which are in charge of executing a precise step of the data reduction. There are 3 ways to run a pipeline:

- `esorex`, a command-line utility for running pipeline recipes. It may be embedded by users in their own computers into data reduction scripts for the automation of processing tasks.
- `esoreflex` is the recommended environment to reduce ESO data. It automatically organizes input files according to their category and runs the entire reduction chain at the push of a button. It supports break points in the reduction sequence in order to inspect and interact with intermediate and final products and rerun the corresponding step if necessary.
- `Gasgano` is a Java-based data file organizer developed and maintained by ESO. It can be used to manage and organize in a systematic way the astronomical data observed and produced by all VLT compliant instruments.

The majority of instrument pipelines come with a pre-defined “workflow”, which is a pre-defined set of execution blocks that runs the individual pipeline recipes with the correct input/outputs. Workflows are meant to be executed within the `esoreflex` environment, each workflow is designed to execute one or more data reduction strategies, that can be selected by the user. Workflows can be modified and users have the possibility to include external programs to optimise their data reduction.

The most convenient way to modify a workflow is to include Python scripts. Starting from version 2.10 of `esoreflex`, both Python 2.7 and Python 3 can be used. The purpose of this document is to provide instructions and examples on how to execute Python scripts within an `esoreflex` workflow. For question about this document, please contact sdp@eso.org or usd-help@eso.org.

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	8 of 25

2 How to insert a PythonActor in an esoreflex workflow

One can add Python scripts to the workflow in any location of the data reduction cascade. The execution of the Python script is regulated in reflex by an apposite actor, named “PythonActor”, that has to be linked to the script itself. The procedure to include a PythonActor and to link it to the Python script is described here and shown in Figure 2.1. The instructions on how to write a Python script compatible with esoreflex is described in Section 3.

1. Type `Python` in the field "Search Components", which is located on the left side of the workflow canvas, in the "Components" Tab. Press search; a list of Python-related actors is shown.
2. Select with the mouse the PythonActor under the Esoreflex → Scripting.kar folder and drag it inside the workflow canvas. It is a green box with a blue square on the bottom-left side.
3. Once you've dragged 'n' dropped the PythonActor actor, it will appear into the canvas. Double click on it to configure it; the configuration dialog window will appear.
4. Configure the fields:
 - Python script. Full path of the python script that has to be included. The syntax requirements of this script is outlined in Section 3.
 - If desired, enable the Lazy Mode (recommended).
 - Set the Recipe Failure Mode to `$RecipeFailureMode`, so that this specific behaviour is regulated by the global parameter `RecipeFailureMode` in the main reflex canvas.
 - Press, Commit to enable the changes.
5. The PythonActor is now linked to the Python script. The black triangles representing the input and output ports will appear at the sides of the Python Actor within the main workflow canvas and need to be connected to the appropriate actors. The products of the PythonActor will be in `$TMP_PRODUCTS_DIR/PythonActor/`. Note that you can also change the name of the PythonActor (right click on the PythonActor with the mouse and select “Customize Name”). In this case the products will be located in `$TMP_PRODUCTS_ DIR/<new_name>/`, where `<new_name>` is the name you selected.
6. If desired, the edited workflow can be saved. The suggested procedure is to go under the File menu and select Export as → xml. If you save the workflow with a new name in a location recognized by esoreflex (installation directory or `/KeplerData/MyWorkflows`), then the new workflow appears in the list of available workflows when typing `esoreflex -l` on the terminal.

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	9 of 25

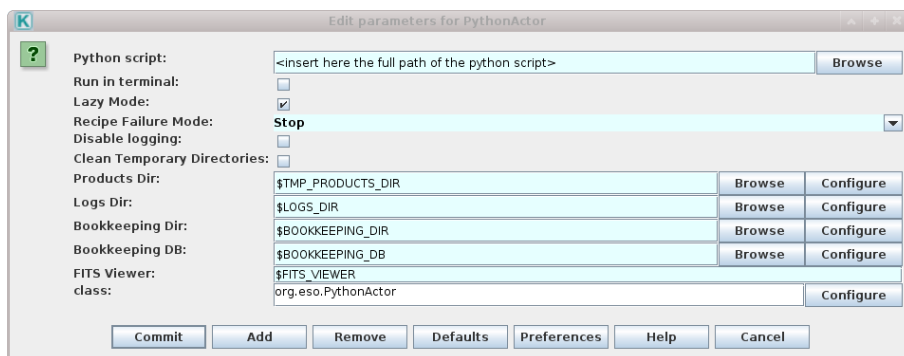
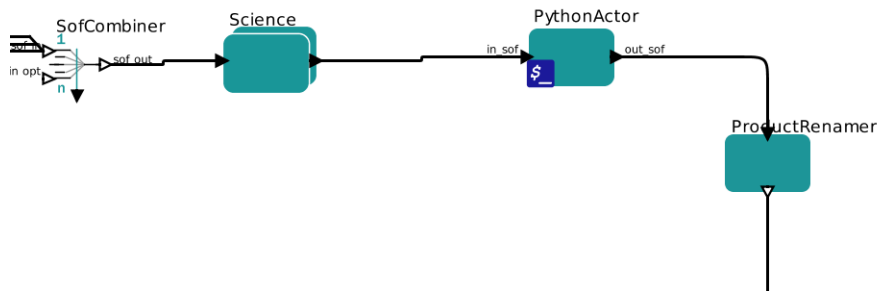
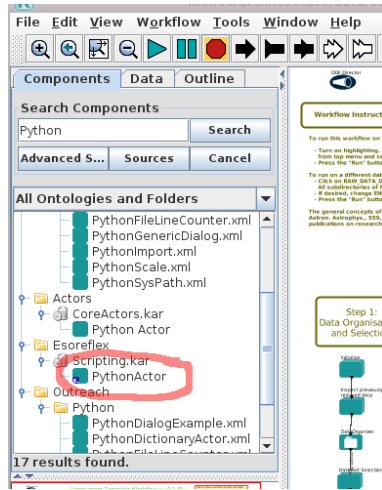


Figure 2.1: How to include the Python actor in a esoreflex workflow. Top panel: location of the Python Actor in the Component Tab. Middle panel: location of a PythonActor in the data reduction chain, as per Section 3.2. Bottom panel: Python Actor configuration window. The full path of the Python script to be executed has to be entered in the first field.

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	10 of 25

3 How to execute Python scripts in EsoReflex: the PythonActor

There are two main ways of executing a Python script, the first via the PythonActor, the second via the Recipe-Executer. In the last case, we refer to the script as a "Python Recipe". PythonActors are the fastest way to include a Python script within a workflow, although with some limitations (e.g., the control of input parameters is not straightforward). A Python recipe is a bit more complicated to prepare, but it has the advantage that it has all the functionality as the regular recipes of the instrument pipeline, and that it can even be called via the `esorex` command line.

In the following we describe the procedure to execute Python scripts via the PythonActor. The definition of how to create and execute a Python recipe will be done in a separate document.

3.1 The PythonActor

The general concept is that `esoreflex` can execute any Python script, but this needs to be included in an appropriate Python interface that allows communication between the PythonActor (i.e. `esoreflex`) and the Python script itself. In the following, we illustrate how to prepare the Python script and how to interface it with `esoreflex`. The instructions of how to include a PythonActor in the workflow and how to link it to the Python script is done in Section 2.

In general, a Python script is divided into 3 areas.

- A1 The `import` Python statements.
- A2 The Python algorithm to be executed.
- A3 The interface between `esoreflex` and the Python algorithm.

The interface (A3) of the Python script is composed into 3 parts:

- P1 A general part that defines the input/output ports and that it is the same for all the cases.
- P2 A customized part that calls the scripts to be executed.
- P3 A general part that broadcasts the products to the output ports and that is the same for all the cases.

An graphical layout of these areas and parts is given in Figure 3.1.

We will consider the following 3 examples of Python scripts.

- E1 Execute a Python script on N input files, process them independently, and produce N outputs. In this example, any script parameter must be hard-coded in the Python script itself.
- E2 Execute a Python script on N input files, process them simultaneously, and produce 1 output. In this example, the Python script will replace a recipe in the workflow. In this example, any script parameter must be hard-coded in the Python script itself.

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	11 of 25

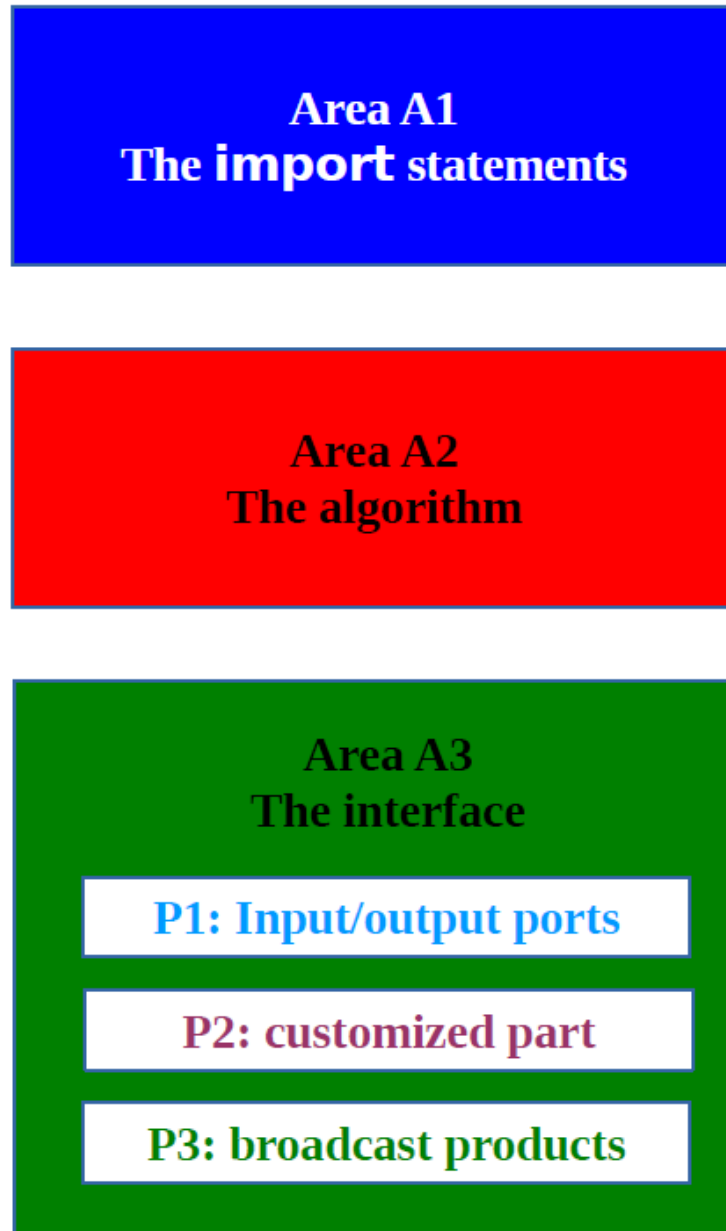


Figure 3.1: Schematic representation of the Python script called by a PythonActor. Definitions of Areas and Parts is as in Section 3.1.

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	12 of 25

E3 As example E1 above, but the script parameters are read from the workflow canvas and not hard-coded in the script.

In all these examples we set the customized part to call 1 function; however the user is free to call as many functions as needed. Each function can call other functions defined in the script. If the function is designed to print a log, it goes into the `reflex_logs` sub-dir, not in the workflow progress windows or in the terminal where the workflow is started from.

The text color in the 3 examples reflects the convention used in Figure 3.1.

3.2 E1. Process files independently

In this example, we prepare a script to include in the `fors_imaging` workflow. The script processes the `SCIENCE_REDUCED_IMG` produced by the `fors` pipeline and divides a box in the center of the frame by 2. We assume the user has the `fors_imaging` workflow installed.

The Python code of the full example can be found here: <https://www.eso.org/sci/data-processing/example1.py>

As mentioned above, the Python script is divided into 3 areas. The first area contains the import statements that are needed for `esoreflex` and the algorithm to be executed.

```
# AREA A1: the import statements
# import statements needed by esoreflex
import reflex
import sys
from optparse import OptionParser
import json
import os

# Import statements needed by the
# algorithm to execute
from astropy.io import fits as fits
import numpy as np
# END OF AREA A1
```

The second area contains the code one has to execute. It can be anything, with its own inputs/outputs. In the following example, the function is called "dostuff". It accepts two inputs that are the strings specifying the name of the fits file to process (i.e., `input_`) and to create (i.e., `output_`).

```
# AREA A2: the algorithm(s).
# This is the function that is called by the python script. The
# function reads the input, edits it. It divides a box region in the
# data by 2 and saves the changes into a new file.
```

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	13 of 25

```
def dostuff(input_, output_) :

    hdu=fits.open(input_, mode='readonly')
    data = hdu[0].data

    ny,nx=data.shape[:]
    xc=np.int (nx/2)
    yc=np.int (ny/2)
    xr=np.int (nx/10)
    yr=np.int (ny/10)
    data[yc-yr:yc+yr,xc-xr:xc+xr] = data[yc-yr:yc+yr,xc-xr:xc+xr]/2

    hdu.writeto(output_, checksum=True,output_verify='ignore')
    hdu.close()
# END OF AREA A2
```

The third and last area is the interface, that ensures the correct communication between Python and esoreflex. It has 3 parts.

The first part is always the same, and it is responsible for generating the input and output ports of the Python-Actor.

```
# AREA A3: the interface.
# This is the main Python script. It contains 2 general parts and a
# structural part, that is responsible of calling the desired
# reduction function.

if __name__ == '__main__':

    # *** PART P1: Input/output ports ***

    #Define inputs/outputs
    parser = reflex.ReflexIOParser()
    parser.add_option("-i", "--in_sof", dest="in_sof")
    parser.add_output("-o", "--out_sof", dest="out_sof")
    # If you need extra input and output ports add these
    # two lines
    #parser.add_input("-k", "--extra_input_port",
    #                dest="extra_input_port")
    #parser.add_output("-q", "--extra_output_port",
    #                 dest="extra_output_port")

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

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	14 of 25

```

in_sof = inputs.in_sof
files = in_sof.files
# Get the files from the extra input port
#in_sof2 = inputs.extra_input_port
#files2 = in_sof2.files

#Define the list of outputs
output_files=list()
output_datasetname=in_sof.datasetName

#Get the name of the output directory
pattern = '--products-dir'
for arg in sys.argv:
    if arg.split("=")[0] == pattern:
        output_dir = arg.split("=")[1]

# *** END OF PART P1 ***

```

The second part of the third area is called the “customized part”, and it specifies the way the algorithm is launched. It has therefore to be modified according to the needs. In the following example, the script does a loop on the input files, and selects only those with a particular category (SCIENCE_REDUCED_IMG), which will be passed to the function “dostuff” for processing. Files that are not of this category will be either broadcast to the output port of the PythonActor, or blocked.

```

# *** PART P2: customized part***
# THE SCRIPT DOES A LOOP ON THE INPUT FILES AND
# PROCESSES ONLY SPECIFIED CATEGORIES INDIVIDUALLY
for file in files:
    if file.category == 'SCIENCE_REDUCED_IMG':
        #The output file has the suffix _new
        infile = file.name.rsplit('/',1)[1]
        output_filename = output_dir+'/'+infile+'_new.fits'
        # This is the call to the reduction function. It can be any script.
        dostuff(file.name,output_filename)

        #Append the newly produced file to the list of outputs
        output_files.append(reflex.FitsFile(output_filename,file.category,\
                                           None,file.purposes))

    else:
        # IF FILE IS NOT PROCESSED, IT IS INCLUDED IN THE OUTPUT
        # SOF AS IT IS.
        # IF YOU DO NOT WANT THE FILE TO BE BROADCASTED, COMMENT
        # THE LINE BELOW.

```

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	15 of 25

```

output_files.append(reflex.FitsFile(file.name, file.category,\
                                   None,file.purposes))

# *** END OF PART P2***

```

The third and last part of the third area is always the same, and it is responsible for broadcasting the output sof to the output port.

```

# *** PART P3: broadcast products ***

#After the script is completed, the list of products is
#broadcasted to the output sof.
new_sof = reflex.SetOfFiles(output_datasetname,output_files)
outputs.out_sof = new_sof

# Broadcast outputs to the extra output port.
#outputs.extra_output_port = new_sof

# broadcast outputs:
parser.write_outputs()
sys.exit()
# *** END OF PART P3 ***
# *** END OF AREA A3***

```

3.3 E2. Process files simultaneously, replace a pipeline recipe

In this example we show how to modify the customized part (P2) of the Python interface (A3) to process N files simultaneously and produce 1 single output (the mean of the input files). In our example we also show how to replace the `fors_bias` pipeline recipe with our Python script. Please note that this is just an example, the adopted algorithm is by no means better than the pipeline recipe.

The full Python script of this example can be found here: <https://www.eso.org/sci/data-processing/example2.py>.

Below, we show the customized part only (P2). The difference with the previous example is that the input files (category BIAS) will be stored in a list and then the list will be passed to the algorithm (called `do_average`). The general parts (before and after the structural part) remain unchanged.

```

# *** P2: CUSTOMIZED PART ***
files_to_process=list() #list containing the full path names.
infile=list()          #list containing the file names.
output_filename = output_dir+'/master_bias.fits'

```

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	16 of 25

```

# THE SCRIPT DOES A LOOP ON THE INPUT FILES AND ONLY
# PROCESSES SPECIFIED CATEGORIES ALL TOGETHER.
for file in files:
    if file.category == 'BIAS':
        #Add the raw bias to the list of files to process
        infile = file.name.rsplit('/',1)[1]
        files_to_process.append(file.name)
        infiles.append(infile)
        purpose=file.purposes
        catg=file.category
    else:
        # IF FILE IS NOT A BIAS, IT IS NOT INCLUDED IN THE OUTPUT SOF.
        # IF YOU DO WANT THE FILE TO BE BROADCASTED, THEN
        # UNCOMMENT THE LINE BELOW.
        # output_files.append(reflex.FitsFile(file.name, file.category, \
            None,file.purposes))

    ok=1

# This is the call to the reduction function. It can be any script.
do_average(files_to_process,infiles,catg,output_filename)

# Add the file output_filename to the list of outputs that Esoreflex
# has to broadcast.
# In this example, only one product is generated.
# NOTE: the purpose of the product is equal to the purpose of the
# inputs. Therefore this PythonActor has to be followed by a
# "ModifyPurpose" set to "trim last" in the workflow.
output_files.append(reflex.FitsFile(output_filename,'MASTER_BIAS', \
    None,purpose))

# *** END OF PART P2 ***

```

The algorithm in area A2 is as follows. It does a average of the input frames, plus it adds some header keywords that are mandatory for recipe products and that are needed for the next recipes.

```

def do_average(input_, infiles, catg, output_) :
    #definition of output array
    tmp=fits.open(input_[0], mode='readonly')
    ny0,nx0=tmp[0].data.shape[:]
    tmp2=tmp[0].data
    a=tmp[0].header['HIERARCH ESO DET OUT1 PRSCX']
    b=tmp[0].header['HIERARCH ESO DET OUT1 OVSCX']
    c=tmp[0].header['HIERARCH ESO DET OUT1 PRSCY']

```


ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	17 of 25

```

d=tmp[0].header['HIERARCH ESO DET OUT1 OVSCY']
# cutting over/pre scan regions
tmp2=tmp[c:ny0-d,a:nx0-b]
ny,nx=tmp2.shape[:]
img=np.zeros((ny,nx,len(input_)))

for i in range(0,len(input_)):
    tmp=fits.open(input_[i], mode='readonly')
    data_tmp=tmp[0].data
    #cutting over/pre scan regions
    img[:, :, i]=data_tmp[c:ny0-d,a:nx0-b]
    if i == 0:
        hdr0=tmp[0].header
        si=str(i)
        hdr0['HIERARCH ESO PRO REC1 RAW'+si+' NAME']=infile[i]
        hdr0['HIERARCH ESO PRO REC1 RAW'+si+' CATG']=catg

mean =np.mean(img,axis=2)
ml = np.mean(mean)
mean = mean-ml
sd=np.std(img,axis=2)
stddev=fits.PrimaryHDU(sd)
ron = np.mean(sd)
#definition of output hdu
hdu = fits.PrimaryHDU(mean,header=hdr0)
hdul = fits.HDUList(hdu)
hdul.append(stddev)
# some header keywords that are mandatory for the fors_bias products
hdul[0].header['HIERARCH ESO PRO TYPE'] = 'REDUCED'
hdul[0].header['HIERARCH ESO PRO CATG'] = 'MASTER_BIAS'
hdul[0].header['HIERARCH ESO PRO TECH'] = 'IMAGE'
hdul[0].header['HIERARCH ESO PRO SCIENCE'] = False
hdul[0].header['HIERARCH ESO PRO DATANCOM']=len(input_)
hdul[0].header['HIERARCH ESO QC BIAS LEVEL'] = ml
hdul[0].header['HIERARCH ESO QC RON'] = ron
hdul[0].header['HIERARCH ESO QC BIAS FPN'] = 0
hdul[0].header['HIERARCH ESO QC BIAS STRUCT'] = 0
hdul[0].header['HIERARCH ESO QC MBIAS LEVEL'] = 0
hdul[0].header['HIERARCH ESO QC MBIAS RONEXP'] = 0
hdul[0].header['HIERARCH ESO QC MBIAS NOISE'] = 0
hdul[0].header['HIERARCH ESO QC MBIAS NRATIO'] = 0
hdul[0].header['HIERARCH ESO QC MBIAS STRUCT'] = 0
hdul[0].header['HIERARCH ESO QC DET OUT1 RON'] = ron
hdul[0].header['HIERARCH ESO QC BIAS OVSC_SUB'] = True
hdul[1].header['EXTNAME'] = 'IMAGE.ERR'

```

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	18 of 25

```
hdul.writeto(output_, checksum=True, output_verify='ignore', \
             clobber=True)
hdul.close()
```

The list of header keywords might be complicated and differ from recipe to recipe. It might be advisable to i) keep the original recipe, ii) define the Python script to also accept the recipe product, iii) define the Python script to replace the header of the Python output with the one of the recipe product.

We now provide the instructions on how to replace the `fors_bias` recipe in the workflow with this Python script.

1. Identify the Composite Actor in the `fors` imaging workflow named `MasterBias`, click on it with the mouse left button and press the delete button on the keyboard.
2. From the left panel of the reflex window, under Search Component, identify and “drag ’n’ drop” in the workflow the following components: `PythonActor`, `ModifyPurpose`, `SofSplitter`, and `SofAccumulator`. Associate the `PythonActor` to the python script (follow Section 2). It is recommended to change the name of the `PythonActor`.
3. Connect with the mouse the Actors that you’ve dragged into the workflow as show in Figure 3.2.
4. Optionally, highlight with the mouse the newly connected actors (they will appear yellow as in Figure 3.2). Click on the main `esoreflex` menu Tools → Create Composite Actor to include these elements in a unique composite actor. You can now change the name of the actor and/or relocate it in the workflow with the mouse. This process might have replicated some workflow variable inside the new composite actor. It is recommended to delete them. The final outfit looks like figure 3.3.

3.4 E3. Set Python script parameters via the workflow.

In all the previous examples, the Python scripts have no input parameters. In the following, however, we instruct the user how to modify the workflow so that the `PythonActor` can accept input parameters. However, the best way to include parameters is via “Python Recipes” (executed with the actor `RecipeExecuter`), and not with the `PythonActor`.

The Algorithm we want to apply in this example is a modification of the algorithm given in Section 3.2. Here, the central box of the science frames is multiplied by a factor (which we’ll call `PAR_1`) and added to a constant value (which we’ll call `PAR_2`). These 2 parameters are inputs of the Python script and can be set directly in the main `esoreflex` canvas.

The Python code of the full example can be found here <https://www.eso.org/sci/data-processing/example3.py>. In the following, we show only relevant portions.

The algorithm in area A2 looks:

```
def dostuff(input_, output_, PAR_1, PAR_2):
```

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	19 of 25

```

hdu=fits.open(input_, mode='readonly')
data = hdu[0].data
ny,nx=data.shape[:]
xc=np.int(nx/2)
yc=np.int(ny/2)
xr=np.int(nx/10)
yr=np.int(ny/10)
data[yc-yr:yc+yr,xc-xr:xc+xr] = \
    data[yc-yr:yc+yr,xc-xr:xc+xr]*PAR_1+PAR_2
hdu.writeto(output_,checksum=True,output_verify='ignore',\
            clobber=True)
hdu.close()

```

The first part (P1) of the interface (area A3) has been modified to add an extra port that will receive the parameters PAR_1 and PAR_2 from the main workflow canvas. It looks like:

```

if __name__ == '__main__':

    # *** GENERAL PART 1 ***

    #Define inputs/outputs
    parser = reflex.ReflexIOParser()
    parser.add_option("-i", "--in_sof", dest="in_sof")
    parser.add_output("-o", "--out_sof", dest="out_sof")
    #extra port for input parameters
    parser.add_input("-p", "--in_sop", dest="in_sop", type='string')

    inputs = parser.get_inputs()
    outputs = parser.get_outputs()
    in_sof = inputs.in_sof
    files = in_sof.files

    #define script input parameters
    param = inputs.in_sop
    params = inputs.in_sop
    values= {}
    for param in params:
        values[param.name]= param.value
    PAR_1 = values['fors_myscript.PAR_1']
    PAR_2 = values['fors_myscript.PAR_2']

    #Define the list of outputs

```

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	20 of 25

```

output_files=list()
output_datasetname=in_sof.datasetName

#Get the name of the output directory
pattern = '--products-dir'
for arg in sys.argv:
    if arg.split("=")[0] == pattern:
        output_dir = arg.split("=")[1]

# *** END OF GENERAL PART 1 ***

```

The call to the dostuff script inside the customized part P2 is as in example E1. Obviously, it has now to include the parameters PAR_1 and PAR_2.

```

[...] (as in E1)
dostuff(file.name,output_filename,PAR_1,PAR_2)
[...] (as in E1)

```

Now the workflow has to be modified to include 3 additional fundamental components. The final configuration is shown in Figure 3.4

1. Two StringParameters. From the left panel, under “Search Components” type “StringParameter”. Select and “drag ’n’ plug” two of them in the main canvas. They are identifiable by the red bullet. Now, Change the name of the StringParameters that you have added to the workflow (click with the right mouse button and select Customize Name. Set their name to PAR_1 and PAR_2 respectively.
2. One String Constant. From the left panel, under “Search Components” type “String Constant”. Select and “drag ’n’ plug” one of it. Configure the string constant by double clicking on it. The configuration window will appear, press “Preferences” to modify the text appearance. The Edit preferences window will appear, change the field “Value” from “Line” to “Text”. Press OK. You can also change the name of the Actor (as example we’ve used “Definition of parameters for script fors_myscript”). Include the following text and press “Commit”.

```

[
  {
    "class": "org.eso.domain.RecipeParameter",
    "description": "multiplicative factor",
    "display_name": "PAR_1",
    "name": "fors_myscript.PAR_1",
    "partype": "value",
    "recipe": "fors_myscript",
    "valtype": "double",
    "value": $PAR_1
  }
]

```

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	21 of 25

```

    },
    {
      "class": "org.eso.domain.RecipeParameter",
      "description": "additive bias",
      "display_name": "PAR_2",
      "name": "fors_myscript.PAR_2",
      "partype": "value",
      "recipe": "fors_myscript",
      "valtype": "double",
      "value": $PAR_2
    }
  ]

```

3. The PythonActor associated to the script. Note that this time, the actor will have an extra output port that will receive the parameters from the workflow.

PAR_1 and PAR_2 can be set by double clicking on them and specifying the desired values (e.g., -1.5 for PAR_1 and -10 for PAR_2).

Optionally, one can add some decorations and text that explains what these parameters are.

You can now connect the newly inserted actors as in Figure 3.4 and save the workflow.

3.5 Updating the Provenance Database

In contrast to the RecipeExecuter (executing either Python or CPL recipes), the PythonActor does not update the `esoreflex` database used by the ProductExplorer. This will limit the capabilities of exploring various products along the data reduction tree and of comparing different executions.

To overcome this issue, a new Actor can be introduced in the `esoreflex` workflow: the ProvenanceUpdater. It is available from `esoreflex` version 2.10.

To include it, type ProvenanceUpdater in the SearchComponent area on the left side of the main workflow window, and “drag ’n’ plug’ it in the workflow.

With respect to the E1 example provided in Section 3.2, the ProvenanceUpdater has to be connected as shown in Figure 3.5.

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	22 of 25

● BOOKKEEPING_DB: \$BOOKKEEPING_D

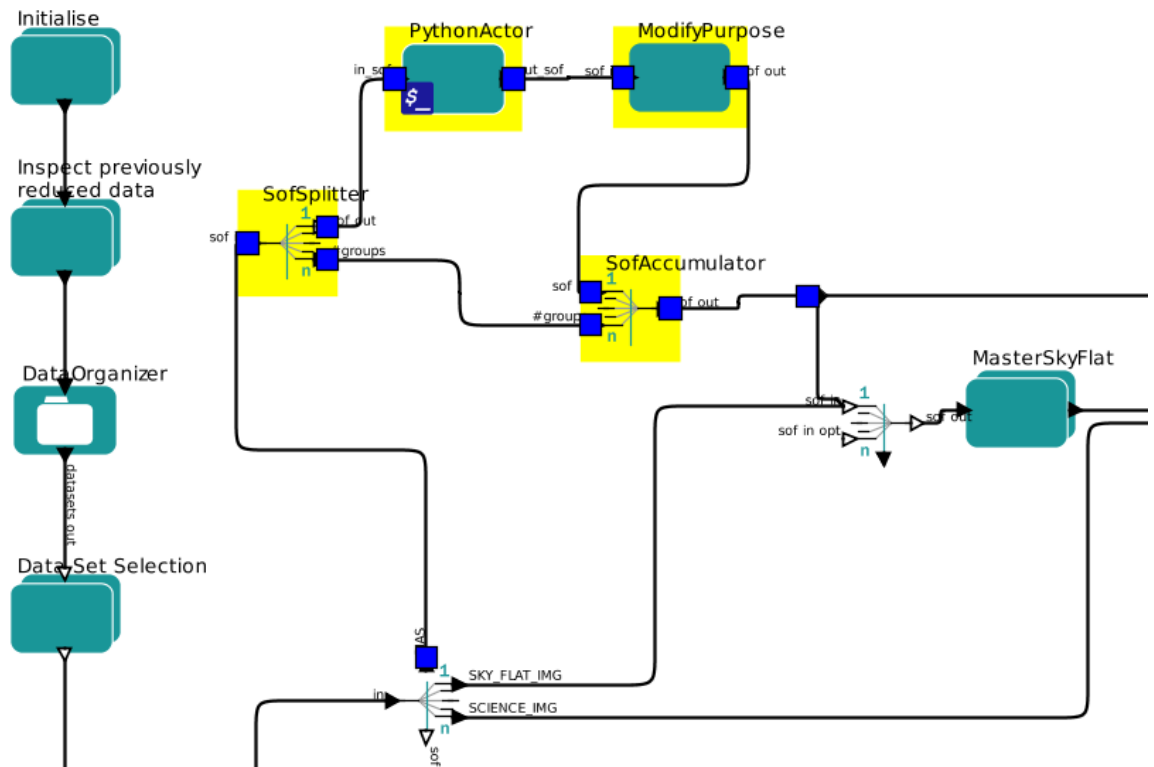



Figure 3.2: The new Actors that need to be included in the workflow in order to replace the fors_bias recipe, as per example E2. The structure SofSplitter and SofAccumulator is a loop that execute the job for each input dataset. The PythonActor has to be linked to the PythonScript and configured as described in Section 2. The ModifyPurpose actor ensures that the products that will be broadcast to the next workflow steps have the right purpose. Open the configuration menu by double-clicking on the Actor with the mouse and set “File Purpose Processing” field to “Strip Last”.

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	23 of 25



FORS Workflow For Imaging Data (v. 5.4.3)

This is a basic workflow to help with data organisation and execution of the pipeline. The workflow was generated without a review of the quality of the science products.

Workflow Instructions

To run this workflow on the demo data:

- Turn on highlighting. Choose "Tools"-> "Animate at Runtime" from top menu and set it to "1".
- Press the "Run" button OR cntrl-R to start the workflow.

To run on a different data set:

- Click on RAW_DATA_DIR and set as appropriate. All subdirectories of RAW_DATA_DIR will be searched for data.
- If desired, change END_PRODUCTS_DIR. IMPORTANT: END_PRODUCTS_DIR should not be a subdirectory of the RAW_DATA_DIR, otherwise it will be searched for raw data!
- Press the "Run" button OR cntrl-R to start the workflow.

The general concepts of Reflex are described in Astron. Astrophys., 559, A96. Please credit this paper in publications on research that used Reflex.

Workflow tutorial and Fors pipeline manual can be found here: http://www.eso.org/sci/software/pipelines/#reflex_workflows

Setup Directories

- ROOT_DATA_DIR: /scratch/loccato/data/esoreflex2.9/fors/data_wkf
- RAW_DATA_DIR: \$ROOT_DATA_DIR/reflex_inputfors
- CALIB_DATA_DIR: /None of the di
- END_PRODUCTS_DIR: /None of the di
- BOOKKEEPING_DIR: /None of the di
- LOGS_DIR: \$ROOT
- TMP_PRODUCTS_DIR: /None of the di
- BOOKKEEPING_DB: /None of the di

Input:

- Only change CALIB_DATA_DIR if you do NOT want to use the calibration data

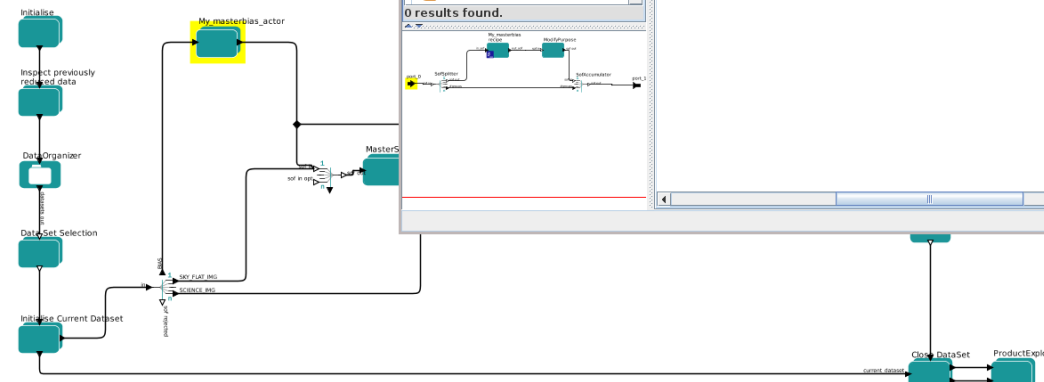
Global Parameters

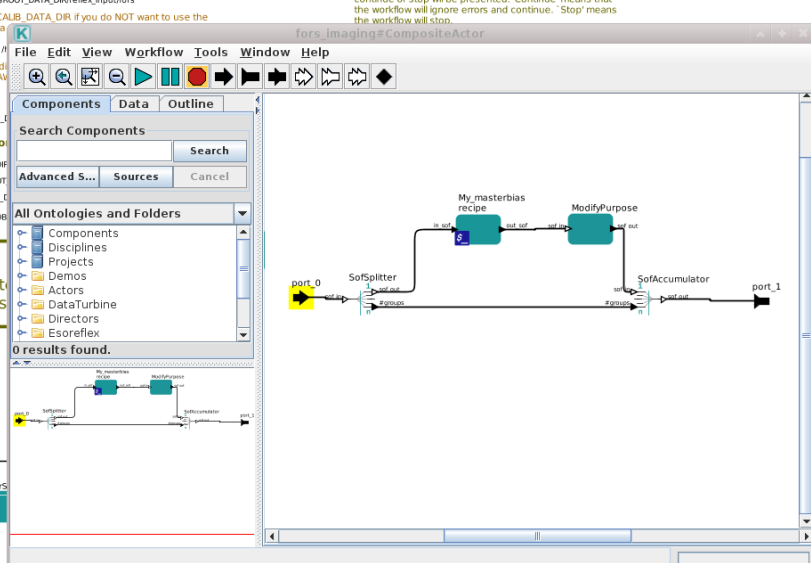
- RecipeFailureMode: Ask

Global parameter for the behaviour when a recipe fails. 'Ask' means that each time a recipe fails, the choice to continue or stop will be presented. 'Continue' means that the workflow will ignore errors and continue. 'Stop' means the workflow will stop.

Step 1:
Data Organisation
and Selection

Step 2:
Creation of Mast
Calibration Files





Auxiliary and debug parameters, please do not change: ● GLOBAL_TIMESTAMP: 2019-09-16T11:47:44 ● ESOREArgs: --suppress-prefix=TRUE ● END_PRODUCTS_SUBDIR: 2019-09-16T11:47:44/FORS2.2007-08-02T07:31:29.167 ● N_SELECTED_DATASETS: 1

Figure 3.3: Final configuration after having replaced the fors_bias recipe with our Python script, as per example E2 in Section 3.3. The panel on the right illustrates the content of the CompositeActor that replace the original Master Bias composite actor. In the figure, a new name has been given to the composite actor (My_masterbias_actor).

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	24 of 25

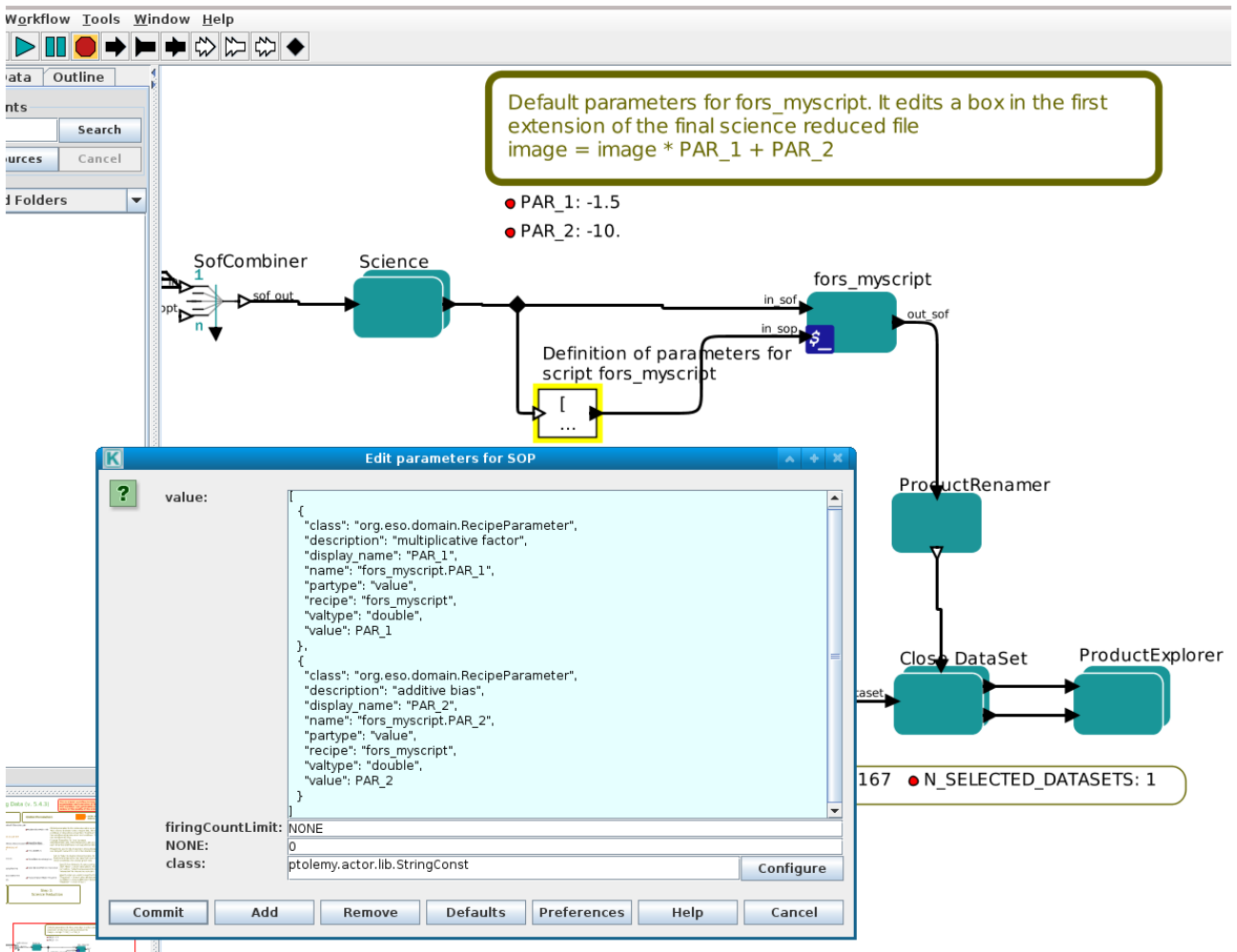


Figure 3.4: Final configuration of example E3. The workflow now contains 2 new parameters, PAR_1 and PAR_2 that will be passed to the Python script associated to the PythonActor fors_mycript. The values defined in PAR_1 and PAR_2 will be converted into a reflex SOP (Set Of Parameters) by the String Constant which we named Definition of parameters for script fors_mycript. The structure of the SOP is embedded in the String Constant and shown in the figure box.

ESO	How to include Python scripts into esoreflex workflows.	Doc:	VLT-MAN-ESO-xxxx
		Issue:	Issue 3.0
		Date:	Date 2022 06 12
		Page:	25 of 25

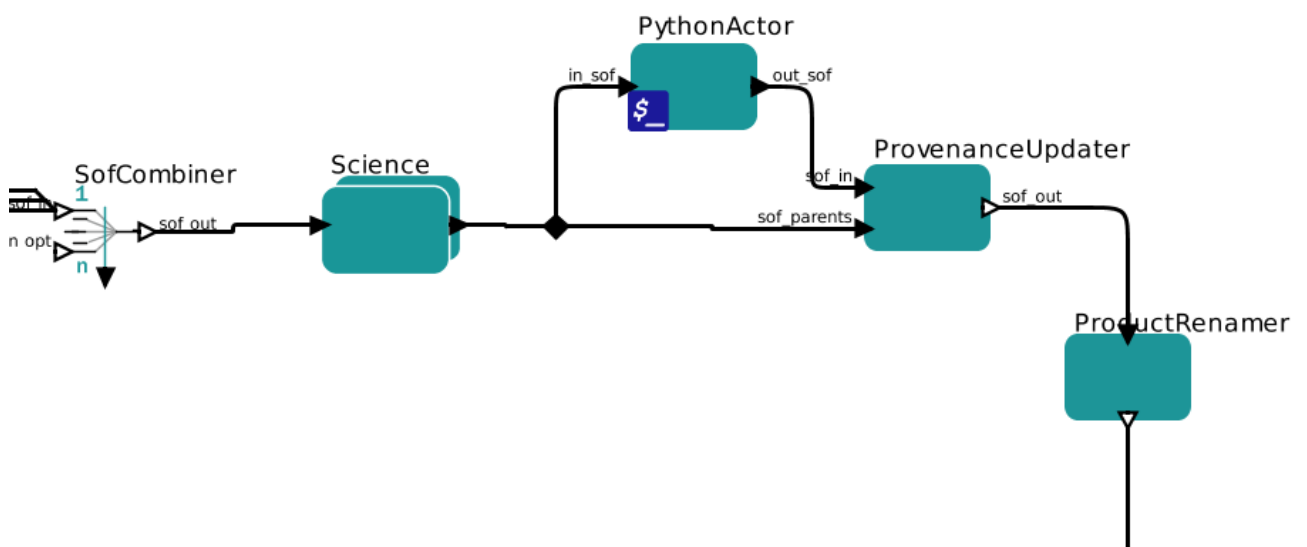


Figure 3.5: How to connect ProvenanceUpdater and PythonActor in a workflow.