European Organisation for Astronomical Research in the Southern Hemisphere

Programme: VLT

Project/WP: Science Data Products Group

EsoReflex ZAP tutorial

Document Number: ESO-287180

Document Version: 2.0

Document Type: Manual (MAN)

Released on: 2017-03-01

Document Classification: Public

Prepared by: L. Coccato

Validated by:

Approved by:

Name

This page was intentionally left blank

Change record

Issue/Rev.	Date	Section/Parag. affected	Reason/Initiation/Documents/Remarks
1.0	01-07-2016	All	First official release
2.0	01-03-2017	All	Association of dedicated sky exposures.

This page was intentionally left blank

Contents

1 Introduction and scope

Reflex is the ESO Recipe Flexible Execution Workbench, an environment to run ESO VLT pipelines which employs a workflow engine to provide a real-time visual representation of a data reduction cascade, called a workflow, which can be easily understood by most astronomers. The basic philosophy and concepts of Reflex have been discussed by Freudling et al. (2013A&A...559A..96F). Please reference this article if you use Reflex in a scientific publication.

The current MUSE reflex distribution contains a dedicated workflow, `muse_zap.xml`, which makes use of the Zurich Atmosphere Purge code (ZAP, Soto et al. 2016, MNRAS, 458, 3210). It is a Python code that has been created to remove residual sky contamination from the MUSE datacubes by performing principal components analysis.

Typically, the datacubes produced by the MUSE pipeline contain a residual sky contamination of the order of $\approx$ 5% of the sky signal. This residual contamination can be higher if the sky has been evaluated on dedicated exposures, because of the time difference between the sky and target observations. The ZAP code helps to remove this residual contamination.

The purpose of this document is to instruct the user on the use of the `muse_zap.xml` workflow to efficiently remove residual sky contamination from MUSE datacubes.

In the following, we assume that the user is familiar with:

- the basic concepts of esoreflex;
- the basic steps of the reduction with the MUSE pipeline, in particular with the products category names such as `DATACUBE_FINAL` and `IMAGE_FOV`;
- the ZAP code.

Reflex and the data reduction workflows have been developed at ESO and they are fully supported. If you have any issue with this particular workflow, please contact sdp_muse@eso.org for further support.

2 Installation

Reflex and the `muse_zap` workflows can be installed in different ways: via package repositories, via the `install_esoreflex` script or manually installing the software tar files.

The recommended way is to use the package repositories if you operating system is supported. The `macports` repositories support OS X (section 2.1), while the `rpm/yum` repositories support Fedora 20/21/22/23 (section 2.2). For any other operating system it is recommended to use the `install_esoreflex` script (section 2.3).

2.1 Installing Reflex workflows via `macports`

This method is supported for OS X operating system. It is assumed that macports (www.macports.org) and java are installed. If you have any problem with this installation method, please read the full documentation at www.eso.org/sci/software/pipelines/installation/macports.html.

For a quick installation, the following steps will install the ESO pipeline macports repository, the MUSE pipeline, including the Reflex workflows support and Reflex itself:

- Set up the repository:

```
# curl ftp://ftp.eso.org/pub/dfs/pipelines/repositories/macports/setup/Portfile -o Portfile
# sudo port install
# sudo port sync
```

- Install Reflex, the MUSE pipeline, demo data, and the workflows:
  `# sudo port install esopipe-muse-all`

Other useful installation options are:

- To show the list of available top level packages for different instruments is given by:

  `port list esopipe-*`

- To install only the ZAP workflow, type:

  `sudo port install esopipe-muse-contrib-wkf`

2.2 Installing Reflex workflows via `rpm/yum`

This method is supported for Fedora 20/21/22/23 operating systems. If you have any problem with this installation method, please read the full documentation at www.eso.org/sci/software/pipelines/installation/rpm.html.

For a quick installation, the following steps will install the ESO pipeline rpm repository, the MUSE pipeline, including the Reflex workflows support and Reflex itself:

- Set up the repository for Fedora 20/21:

```
# sudo yum install yum-utils
# sudo yum-config-manager \
  --add-repo=ftp://ftp.eso.org/pub/dfs/pipelines/repositories/fedora/esorepo.repo
```

- Set up the repository for Fedora 22/23:

```
# sudo dnf install dnf-plugins-core
# sudo dnf config-manager \
  --add-repo=ftp://ftp.eso.org/pub/dfs/pipelines/repositories/fedora/esorepo.repo
```

- Install the MUSE pipeline (Fedora 20/21):
  `# sudo yum install esopipe-muse-all`
- Install the MUSE pipeline (Fedora 22/23):
  `# sudo dnf install esopipe-muse-all`

To install the muse_zap workflow only, replace `esopipe-muse-all` with `esopipe-muse-contrib-wkf`. To see the list of all available packages to install, type
`yum list esopipe-*-all` # (Fedora 20/21)
`dnf list esopipe-*-all` # (Fedora 22 or newer)

2.3 Installation with the `install_esoreflex` script

Download the `install_esoreflex` script

`ftp://ftp.eso.org/pub/dfs/reflex/install_esoreflex`

and execute it by typing:

```
chmod u+x install_esoreflex
./install_esoreflex
```

On the top of the esoreflex requirements common for the other workflows that are specified at:

`http://www.eso.org/sci/software/pipelines/reflex_workflows/`

the `muse_zap` workflow requires the following Python packages: astropy and scipy. If they are not present in your system, please install them by executing:

```
sudo yum install python-astropy scipy
```

for Fedora systems; or

```
sudo apt-get install python-astropy python-scipy
```

for Ubuntu systems.

For Mac OS X type use:

```
sudo port install astropy py-scipy
```

2.3.1 Using your preferred Python environment

If you have your Python environment (e.g., installed via miniconda) that contains all the prerequisites including astropy and scipy, you can instruct esoreflex to use it instead of the default Python. To do so, proceed as follows:

1. Creation of a esoreflex configuration file. From your terminal type:

   ```
   esoreflex -create-config zap.rc
   ```

   This will create a default configuration file, named zap.rc. Any other name is ok.

2. Edit the newly created zap.rc and change the Python command call. The line to modify is:

   ```
   esoreflex.python-command=python
   ```

   which is located around line 27. The value you have to enter is the full path of Python executable of the environment you'd like to use. For example, if you environment is saved in:

   ```
   /home/user/miniconda/envs/my_environment/
   ```

   the line to insert in the configuration file is:

   ```
   esoreflex.python-command=/home/user/miniconda/
          envs/my_environment/bin/python
   ```

3. Instruct esoreflex to use the newly edited zap.rc file. Start esoreflex with the command line:

   ```
   esoreflex -config <full_path>/zap.rc
   ```

   In this way, esoreflex will use the desired Python environment. Please, specify the full path to the zap.rc configuration file.

3 System requirements

Handling MUSE datacubes with nominal spatial sampling and wavelength coverage is highly demanding in terms of memory. We recommend to use at least 20 Gb RAM for single pointing datacubes. For computers with limited resources one way to process the data is to use a small number of principal components. See Sections 4.1 and 7.2 for further information.

The ZAP code is parallelized, therefore it takes advantage of all the available processors to speed up the entire process.

The demo dataset was created by setting the spatial sampling of the cube to 0.8 arcsec/pixel (full-size datacubes have 0.2 arcsec/pixel), and can be processed with a 8 Gb RAM computer using default parameters.

4 Quick start: reducing the demo data

The current distribution of the `muse_zap` workflow contains two demo datasets.

The first dataset contains a fully reduced and sky subtracted datacube (name: `CUBE1.fits`, category: `DATACUBE_FINAL`) and its corresponding white-light image (name: `FOV1.fits`, category: `IMAGE_FOV`).

The second dataset contains a fully reduced and sky subtracted datacube (name: `CUBE2.fits`, category: `DATACUBE_FINAL`), its corresponding white-light image (name: `FOV2.fits`, category: `IMAGE_FOV`), a cube of residual sky background from a dedicated sky exposure (name: `DATACUBE_SkyRes.fits`, category: `DATACUBE_FINAL`), and its corresponding white-light image (name: `IMAGE_SkyRes.fits` category: `IMAGE_FOV`).

Note: a cube of residual sky background from a dedicated sky exposures can be obtained from the main muse reflex workflow (see the main muse reflex tutorial, Section 8.3).

For the user who is keen on starting reductions without being distracted by detailed documentation, we describe the steps to be performed to use the ZAP code to clean residual sky background contamination from the demo data set supplied with the current release. By following these steps, the user should have enough information to attempt a reduction of his/her own data without any further reading.

1. Type:

   ```
   <install_dir>/bin/esoreflex -l&
   ```

   at the terminal command line. A list of available workflows and their short-cut names will appear.

2. Start the `muse_zap` workflow by typing:

   ```
   <install_dir>/bin/esoreflex muse\_zap&
   ```

   The main `muse_zap.xml` workflow will appear (Figure 4.1). Alternatively, you can open an empty reflex canvas by typing:

   ```
   <install_dir>/bin/esoreflex &
   ```

   The empty `esoreflex` canvas will appear. You can then open the `muse_zap` workflow by clicking on `File -> Open`, selecting first `MUSE-<version>/contrib_wkf/` and then the file `muse_zap.xml` in the file browser.

   Note: `<install_dir>` is the reflex installation directory specified at the reflex installation (e.g., /home/username/reflex/install). If the command esoreflex is present in your path, or an alias has been defined, there is no need to prepend the `<install_dir>/bin/` path to it.

3. To aid the visual tracking of the reduction cascade, it is advisable to use component (or actor) highlighting. Click on `Tools -> Animate at Runtime`, enter the number of milliseconds representing the animation interval (100 ms is recommended), and click `OK`.

4. Under "Setup Directories" in the workflow canvas there are some parameters that specify important directories (green dots).Setting the value of `ROOT_DATA_DIR` is the only necessary modification if you want to process data other than the demo data[1], since the value of this parameter specifies the working directory within which the other directories are organized. Double-click on the parameter `ROOT_DATA_DIR` and a pop-up window will appear allowing you to modify the directory string, which you may either edit directly, or use the `Browse` button to select the directory from a file browser. When you have finished, click `OK` to save your changes.

   Set the `INPUT_DATA_DIR` to the directory that contains the datacubes to be cleaned (with category `DATACUBE_FINAL`) and their corresponding collapsed images (with category `IMAGE_FOV`)[2]. Again, for the demo dataset, this directory is automatically set to the right path. The `INPUT_DATA_DIR` can also contain user supplied mask (with the category `MASK_FINAL_CUBE`) that separates spaxels that contain only sky (value 0) from spaxels that contain objects (value 1). A user supplied mask, if present, will be associated to the datacube with the same observation time (header keyword `MJD-OBS`).

5. Press  to start the workflow. The following steps will be executed:

   - If `EraseDirs=true` was set, the workflow will erase the content of the log, bookkeeping, and temporary directories.
   - The workflow will highlight the `Data Organiser` actor which recursively scans the raw data directory (specified by the parameter `INPUT_DATA_DIR` under "Setup Directories" in the workflow canvas) and constructs the DataSets.

     Datasets are created on the basis of header information and a set of `OCA`[3]. A dataset for each datacube in the `INPUT_DATA_DIR` will be created. Each dataset has one `IMAGE_FOV` and one sky mask (if present) with the same `MJD-OBS` as the datacube. Datasets without a `IMAGE_FOV` are considered incomplete.
   - The `Data Set Chooser` actor will be highlighted next and will display a "Select Datasets" window (see Figure 4.2, top panel) that lists the DataSets along with the values of a selection of useful header keywords[4]. The first column consists of a set of tick boxes which allow the user to select the DataSets to be processed. By default all complete DataSets which have not yet been reduced will be selected. Incomplete datasets are grayed out in the window. If you put the mouse cursor on the top of an incomplete dataset, you will get information on the missing file categories. Each dataset can be inspected by pressing the "Inspect Highlighted" button; the Select Frames window will appear (Figure 4.2, bottom panel).

6. Click the `Continue` button and watch the progress of the workflow by following the red highlighting of the actors. A window will show which DataSet is currently being processed. If interactivity is enabled, the mask creation can be done interactively and a dedicated window will appear. This will be discussed in Section 7.1. Click the `Continue` button to proceed with the next steps of the analysis.

[1] If you used the install script `install_esoreflex`, then the value of the parameter `ROOT_DATA_DIR` will already be set correctly to the directory where the demo data was downloaded.

[2] All categories of the MUSE internal data products are supported.

[3] `OCA` stands for Organization, Classification, Association and refers to rules, which allow to classify the raw data according to the contents of the header keywords, organize them in appropriate groups for processing, and associate the required calibration data for processing.

[4] The keywords listed can be changed by right-clicking on the `DataOrganiser` Actor, selecting `Configure Actor`, and then changing the list of keywords in the second line of the pop-up window.

7. The workflow processes the selected datasets separately. Only when the reduction of one dataset is completed, the reduction of the next dataset starts.
8. The workflow will continue with the remaining DataSets following the same steps described above.
9. After the workflow has finished, all the products from all the DataSets can be found in a directory under `END_PRODUCTS_DIR` with the named with the workflow start timestamp. Further sub-directories will be found with the name of each DataSet.
10. When the reduction of the last DataSet is finishes, a pop-up window called *Product Explorer* will appear showing the datasets which have been so far reduced together with the list of final products. This actor allows the user to inspect the final data products, as well as to search and inspect the input data used to create any of the products of the workflow. Figure 4.3 shows the Product Explorer window.

Well done! You have successfully completed the quick start section and you should be able to use this knowledge to reduce your own data. However, there are many interesting features of `Reflex` and the ZAP workflow that merit a look at the rest of this tutorial.

4.1 Processing your own datacubes

To process your own MUSE datacubes, simply change the path to the input data directory. This is defined at the top of the workflow window in the area labeled Setup Directories . Simply double click on the `INPUT_DATA_DIR`, enter the path to your raw science directory and then re-run the workflow in the same way as was done for the tutorial demo data.

If the reduction of your dataset requires high memory, you can limit the memory usage by decreasing the number of components to fit. This can be achieved by setting **Eigenspectra_detection_type (optimizeType)**=*'none'* and specifying the components to use via the **Percentage of eigenspectra(pevals)** or **number of eigenspectra(nvals)** parameters in the main workflow canvas.

Also, you can save time when experimenting with **pevals** and **nevals** by passing the file with the components via the **External Sky components (full path)** parameter in the main workflow canvas. In this way, the ZAP code does not recompute the components and is much faster.

See section 7.2 for further information.

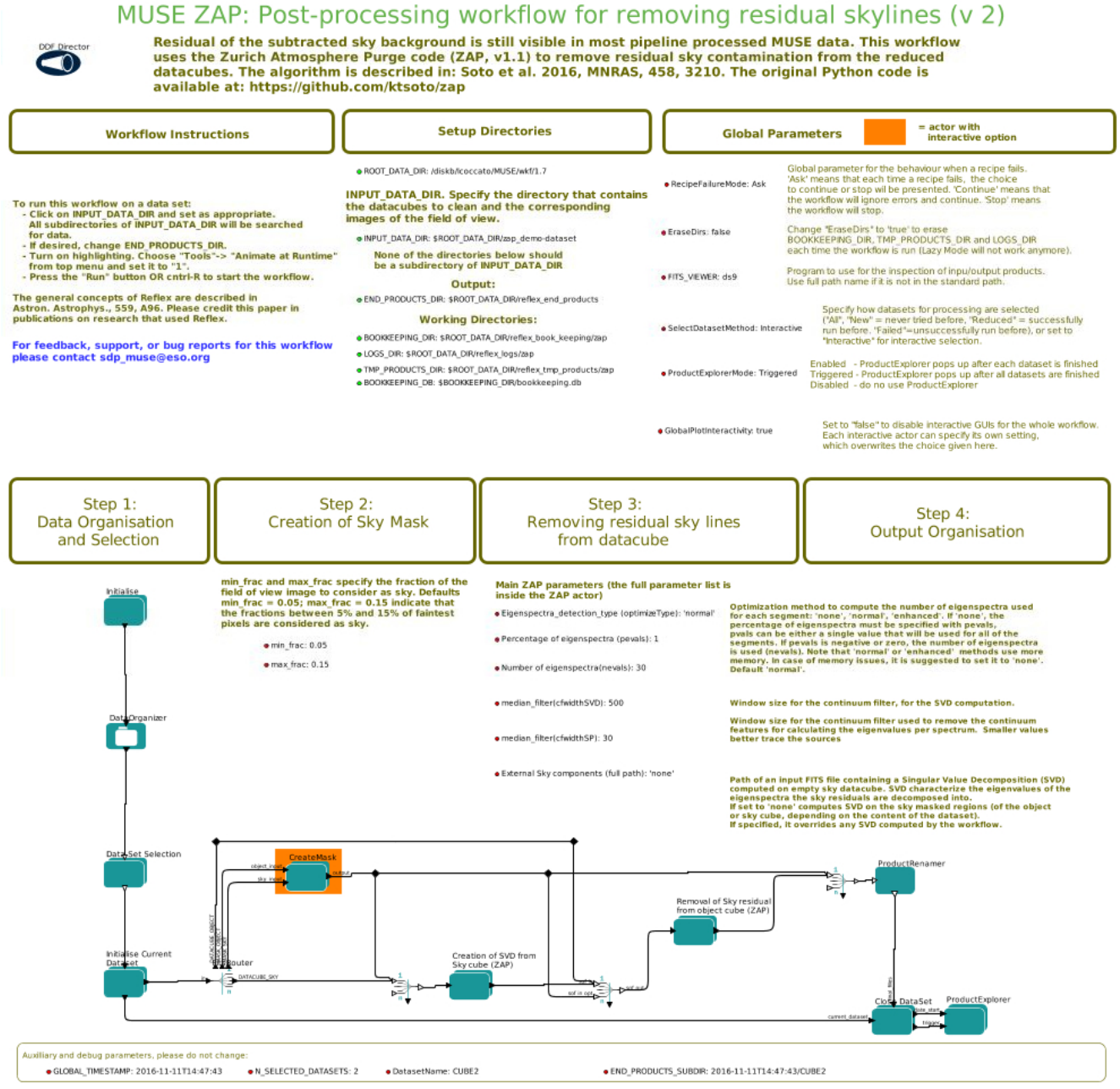


Figure 4.1: The `muse_zap.xml Reflex` workflow; it executes the Zurich Atmosphere Purge code (Soto et al. 2016) to remove residual sky contamination from the datacubes.

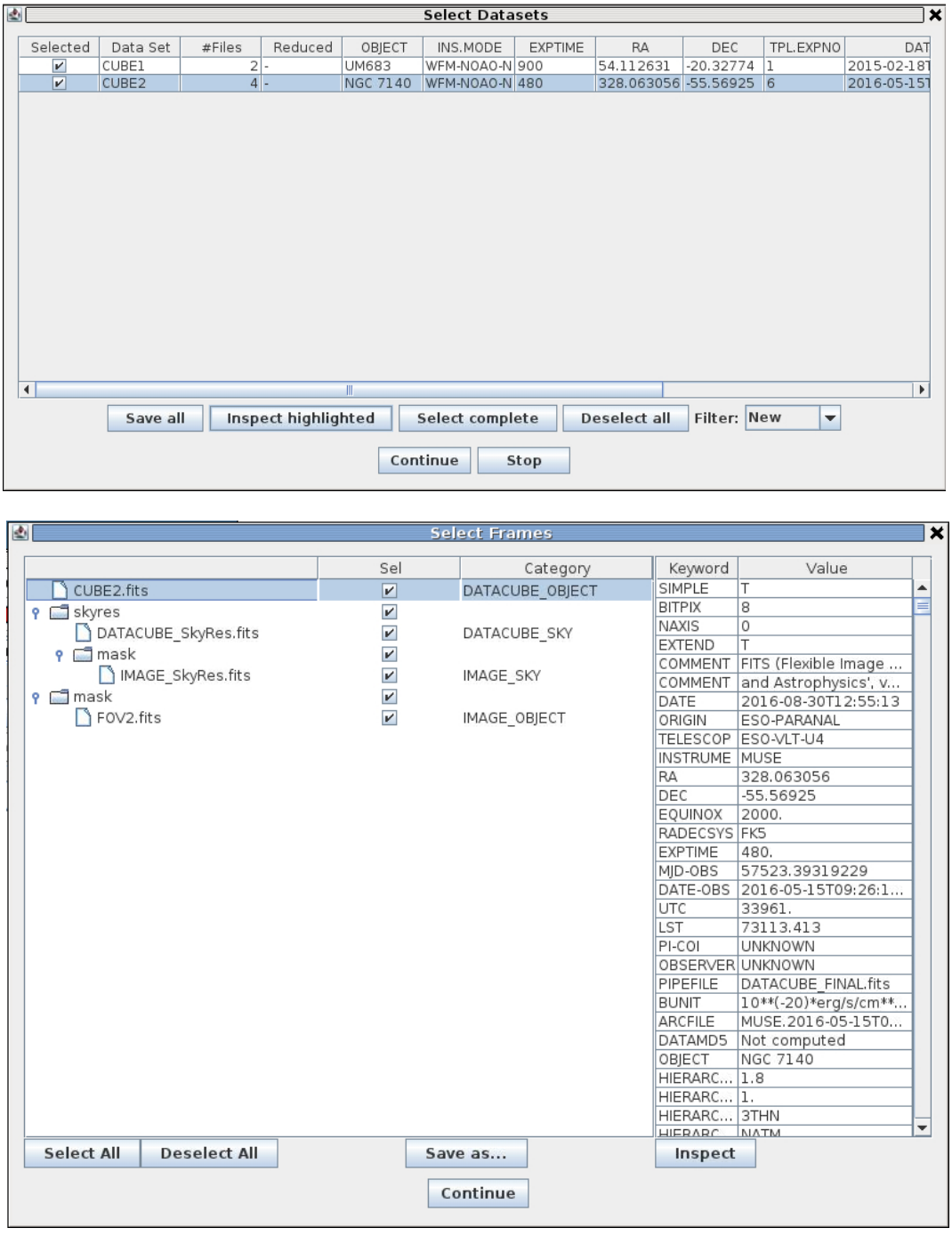


Figure 4.2: The Select Datasets window (top) and the Select Frames window (bottom).

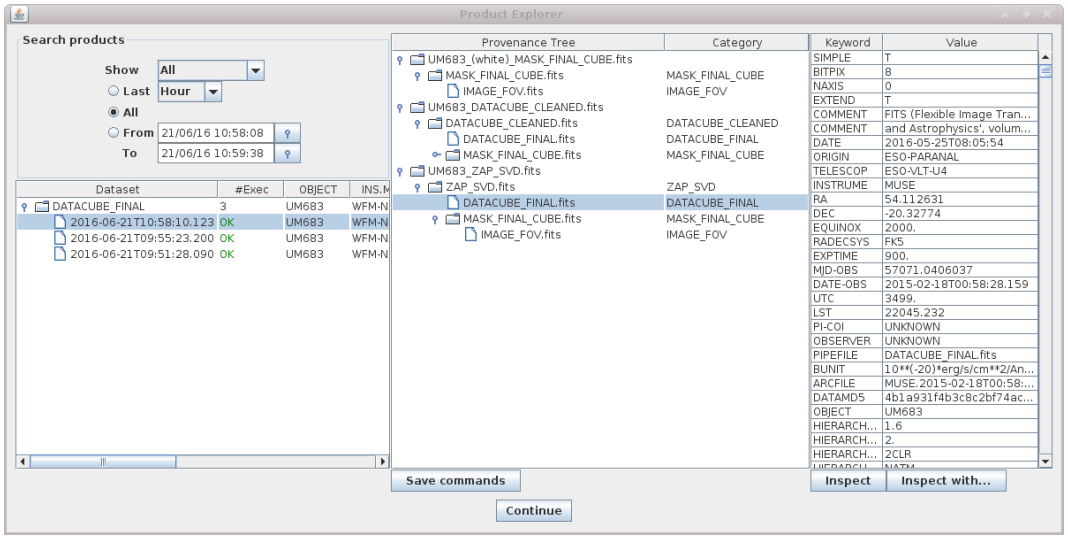


Figure 4.3: The Product Explorer window.

5 About The Reflex Canvas

5.1 Saving And Loading Workflows

In the course of your data reductions, it is likely that you will customise the workflow for various data sets, even if this simply consists of editing the `ROOT_DATA_DIR` to a different value for each data set. Whenever you modify a workflow in any way, you have the option of saving the modified version to an `XML` file using `File -> Export As` (which will also open a new workflow canvas corresponding to the saved file). The saved workflow may be opened in subsequent `Reflex` sessions using `File -> Open`. Saving the workflow in the default Kepler format (.kar) is only advised if you do not plan to use the workflow with another computer.

5.2 Buttons

At the top of the `Reflex` canvas are a set of buttons which have the following functions:

-  - Zoom in.
-  - Reset the zoom to 100%.
-  - Zoom the workflow to fit the current window size (Recommended).
-  - Zoom out.
-  - Run (or resume) the workflow.
-  - Pause the workflow execution.
-  - Stop the workflow execution.

The remainder of the buttons (not shown here) are not relevant to the workflow execution.

5.3 Workflow States

A workflow may only be in one of three states: executing, paused, or stopped. These states are indicated by the yellow highlighting of the  ,  , and  buttons, respectively. A workflow is executed by clicking the  button. Subsequently the workflow and any running pipeline recipe may be stopped immediately by clicking the  button, or the workflow may be paused by clicking the  button which will allow the current actor/recipe to finish execution before the workflow is actually paused. After pausing, the workflow may be resumed by clicking the  button again.

6 The muze_zap workflow

In this section we describe in more details the various elements of the muse_zap workflow, the steps it executes, and its interactive features.

The muse_zap workflow supports the two main strategies of the ZAP code. The first evaluates the principal components that characterize the sky residual directly on the field of view of the object to clean. This is the case of the first demo dataset. The second evaluates the principal components on a sky residual cube (from a dedicated sky observation) and then apply the results to the object cube. This is the case of the second demo dataset.

In both strategies, the principal components are evaluated only on a portion of the field of view as defined by a sky mask.

6.1 Creation of datasets

The first thing to know for understanding how the muze_zap workflow works and the adopted strategy, is understand how the datasets are generated.

The creation of datasets is done by the actor DatasetOrganizer, which follows a series of rules, as in all other workflows. The workflow handles these kind of files, which are identified by the category written in the PRO CATG header keyword:

- **Datacubes**. Any category that contains the string DATACUBE is considered a datacube. These files could be either the datacubes of the objects we have to clean (classified as DATACUBE_OBJECT by the workflow), or the datacube of a residual sky background (classified as DATACUBE_SKY by the workflow). This distinction is done on the basis of the PRO REC1 RAW1 CATG header keyword, which can be either OBJECT or SKY. A DATACUBE_SKY is associated to a DATACUBE_OBJECT if they have the same value of the OBJECT header keyword.

- **Images**. Any category that contains the string IMAGE_FOV is considered an image, which will be used to create a sky mask. The images can be associated to the object (in this case they will be classified as IMAGE_OBJECT by the workflow) or to a sky residual cube (in this case they will be classified as IMAGE_SKY by the workflow). As for the datacubes, this distinction is done on the basis of the PRO REC1 RAW1 CATG header keyword, which can be either OBJECT or SKY. An image is associated to a datacube if they have the same value of the MJD-OBS header keyword.

- **Sky masks**. Sky masks have the category MASK_FINAL_CUBE. They can be classified as MASK_OBJECT or MASK_SKY depending on the value of the PRO REC1 RAW1 CATG header keyword, as for the previous cases. A sky mask is associated to a cube if they have the same MJD-OBS header keyword.

Each dataset contains a DATACUBE_OBJECT and an IMAGE_OBJECT. Eventually, they can have a MASK_OBJECT assigned to it. The user can decide whether to use the provided mask or create a new one from the image. These datasets reflect the first analysis strategy, where the principal components are evaluated on the object datacube and then fitted.

A DATACUBE_SKY with the corresponding IMAGE_SKY (and eventually a MASK_SKY) can be associated to the dataset, if they target the same object. These datasets reflect the second analysis strategy, where the principal components are evaluated on the sky cube and then fitted to the object datacube.

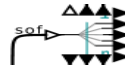
6.2 Workflow actors

6.2.1 Simple Actors

Simple actors have workflow symbols that consist of a single (rather than multiple) green-blue rectangle. They may also have a logo within the rectangle to aid in their identification. The following actors are simple actors:

- 

- The Data Organiser actor.
- 

- The Data Set Chooser actor (inside a composite actor).
- 

- The Fits Router actor
- 

- The Product Renamer actor.
- 

- The Product Explorer actor (inside a composite actor).

Access to the parameters for a simple actor is achieved by right-clicking on the actor and selecting Configure Actor. This will open an “Edit parameters” window. Note that the Product Renamer actor is a Jython script (Java implementation of the Python interpreter) meant to be customized by the user (by double-clicking on it).

6.2.2 Lazy Mode

By default, all recipe executer actors in a pipeline workflow are “Lazy Mode” enabled. This means that when the workflow attempts to execute such an actor, the actor will check whether the relevant pipeline recipe has already been executed with the same input files and with the same recipe parameters. If this is the case, then the actor will not execute the pipeline recipe, and instead it will simply broadcast the previously generated products to the output port. The purpose of the Lazy mode is therefore to minimise any reprocessing of data by avoiding data rereduction where it is not necessary.

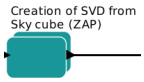
One should note that the actor’s Lazy mode depends on the contents of the directory specified by BOOKKEEPING_DIR and the relevant FITS file checksums. Any modification to the directory contents and/or the file checksums will cause the corresponding actor when executed to run the pipeline recipe again, thereby rereducing the input data.

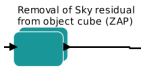
The forced rereduction of data at each execution may of course be desirable. To force a rereduction of all data for all `RecipeExecuter` actors in the workflow (i.e. to disable Lazy mode for the whole workflow), set the `EraseDirs` parameter under the "Global Parameters" area of the workflow canvas to `true`. This will then remove all previous results as well. To force a rereduction of data for any single `RecipeExecuter` actor in the workflow (which will be inside the relevant composite actor), right-click the `RecipeExecuter` actor, select `Configure Actor`, and uncheck the Lazy mode parameter tick-box in the "Edit parameters" window that is displayed. If the Lazy mode is switched off for an actor, all subsequent actors that use products from that one will also reprocess the data, as they see new products.

6.3 Workflow composite actors

There are two main composite actors in the `muse_zap` workflow, which are dedicated to the 2 main steps of the analysis:

- 

`CreateMask`. This actor uses the `IMAGE_OBJECT` or `IMAGE_SKY` to identify the regions in the field of view that are dominated by the sky. Those regions will be used to evaluate the components that describe the sky residuals. It also allows to edit a user-provided mask if present in the dataset. It is an interactive actor, and it will be described in Section 7.1
- 

`Creation of SVD from Sky cube (ZAP)`. It runs the ZAP code to create the sky principal components and save them in the so-called SVD file. It is run only on `DATACUBE_SKY` frames. It uses the output of `CreateMask`.
- 

`Removal of Sky residual from object cube (ZAP)`. It runs the ZAP code to remove residual sky contamination from the object cube. If the principal components were created from the sky cube, they will be used. Otherwise, it will determine the sky principal components from the object cube, using an appropriate mask defined in `CreateMask`.

7 Removing the residuals of the sky background

In this section we describe the three main steps performed by the `muse_zap` workflow aimed at removing the residual contamination of the sky background from a datacube.

7.1 Creation of Sky Mask

The `CreateMask` actor creates a sky mask, which is a 2D image file that covers the same field of view as the datacube and with the same spatial resolution. Its values are 0 (for sky pixel, on which the residuals need to be analyzed) and 1 (for object pixel, not to be used for the sky residual analysis). The category of the mask fits file is `MASK_FINAL_CUBE`, and it is stored in the header. The mask has the same `MJD-OBS` as the associated datacube.

In the case you are evaluating the principal components directly on the object cube, it is recommended to define the sky regions on areas surrounding the object you are interested in, making sure not to include bright sources.

7.1.1 Description of the interactive window

An interactive window will be displayed (Figure 7.1), allowing the user to inspect the current sky mask (either created from the `IMAGE_FOV` or provided in the `INPUT_DATA_DIR`) and, eventually, to change parameters to change the mask definition.

The `CreateMask` interactive window is divided into 2 parts. On the left side, there is the plotting area with two panels; on the right side there is the list of recipe parameters.

Both panels in the plotting area display the image of the field of view (`IMAGE_FOV`) and the current sky mask (`MASK_FINAL_CUBE`) with a reversed color scheme.

In the top panel (named “Sky”) the region dominated by the sky are in gray-scale, whereas the regions dominated by objects are in red. With this window, the user has the possibility to check whether there are bright objects inside the empty sky regions. If so, the sky regions need to be redefined not to include bright objects.

In the bottom panel (named “Object”) the region dominated by the sky are in red, whereas the regions dominated by objects are in gray-scale. With this window, the user has the chance to inspect the coverage of the sky region, and decide whether there are other areas around the object, free of bright sources, that can be included in the mask.

The red-colored pixels in the “Object” panel corresponds to the pixels that have value 0 in the created mask and identify empty sky regions.

If present, the user supplied mask is used by default. Otherwise, the one created with default parameters is used.

7.1.2 Description of the parameters

The mask can be create either from the image of the field of view (`IMAGE_OBJECT` or `IMAGE_SKY`) or from an user-provided mask (`MASK_OBJECT` or `MASK_SKY`, optional) present in the dataset.

The creation of a sky mask is regulated by a series of parameters that can be modified by the user through the interactive window. When a parameter has been modified, press the button "Re-run Recipe" to calculate and display the new mask. If you are happy with the results, press the button "Continue Wkf".

If a mask is provided with the dataset, it will be displayed in the interactive window for inspection. The user has the possibility to modify it, use it as it is, or discard it and create a new one from the `IMAGE_OBJECT/SKY`. These 3 choices are determined by the value of the parameter **edit input mask?** in the interactive window (Figure 7.1). The editing of the user-provided mask is regulated by the parameters **n_grow** and **skybox**.

If a mask is not provided with the dataset, the actor generates a new one from the `IMAGE_OBJECT/SKY` (using default parameters) and displays it. The user has the possibility to modify it by changing the parameters **max_frac**, **min_frac**, **n_grow**, and **skybox**.

The parameters that regulate the creation of the sky mask are the following:

- Tab **Mask**. Note: this Tab and parameters are available only if a user-provided mask is present in the dataset.
  - **edit input mask?**. It determines whether a user-provided mask has to be edited or not. The allowed values are the following:
    - *use*. The provided mask is the one displayed in the left panels and will be used as it is without the creation of a new file.
    - *add*. The provided mask will be edited using **n_grow** and **skybox** to add regions to it.
    - *discard*. The input mask will not be used and a new one will be created using the `IMAGE_FOV` and all the recipe parameters.

    Values need to be entered without quotation marks. Default = *use*.
- Tab **Flux Ranges**
  - **min_frac** and **max_frac**. The mask is created by rejecting the fraction **min_frac** of faintest spaxels in the image, and retain the fraction **max_frac** of faintest spaxels for sky evaluation. Defaults are **min_frac**=*0.05* and **max_frac** = *0.15*. This means that only the 5%-15% faintest pixels in `IMAGE_OBJECT/SKY` are used to define the sky regions in the sky mask. These parameters have an effect only if the mask is created from the `IMAGE_OBJECT/SKY`. The default values can also be changed by double clicking on the corresponding parameter in the workflow main canvas before starting the workflow. *Warning:* These parameters must follow the rule 0 $\leq$**min_frac**$\leq$**max_frac**.
  - **n_grow**. It enlarges the existing map spaxels containing sky by the specified number of pixels. It does not modify regions defined with the **skybox** parameter. It **n_grow** is useful to enlarge isolated sky regions. Large values might increase the risk to include bright objects among the sky regions. Default= *3*.
- Tab **Sky Regions**

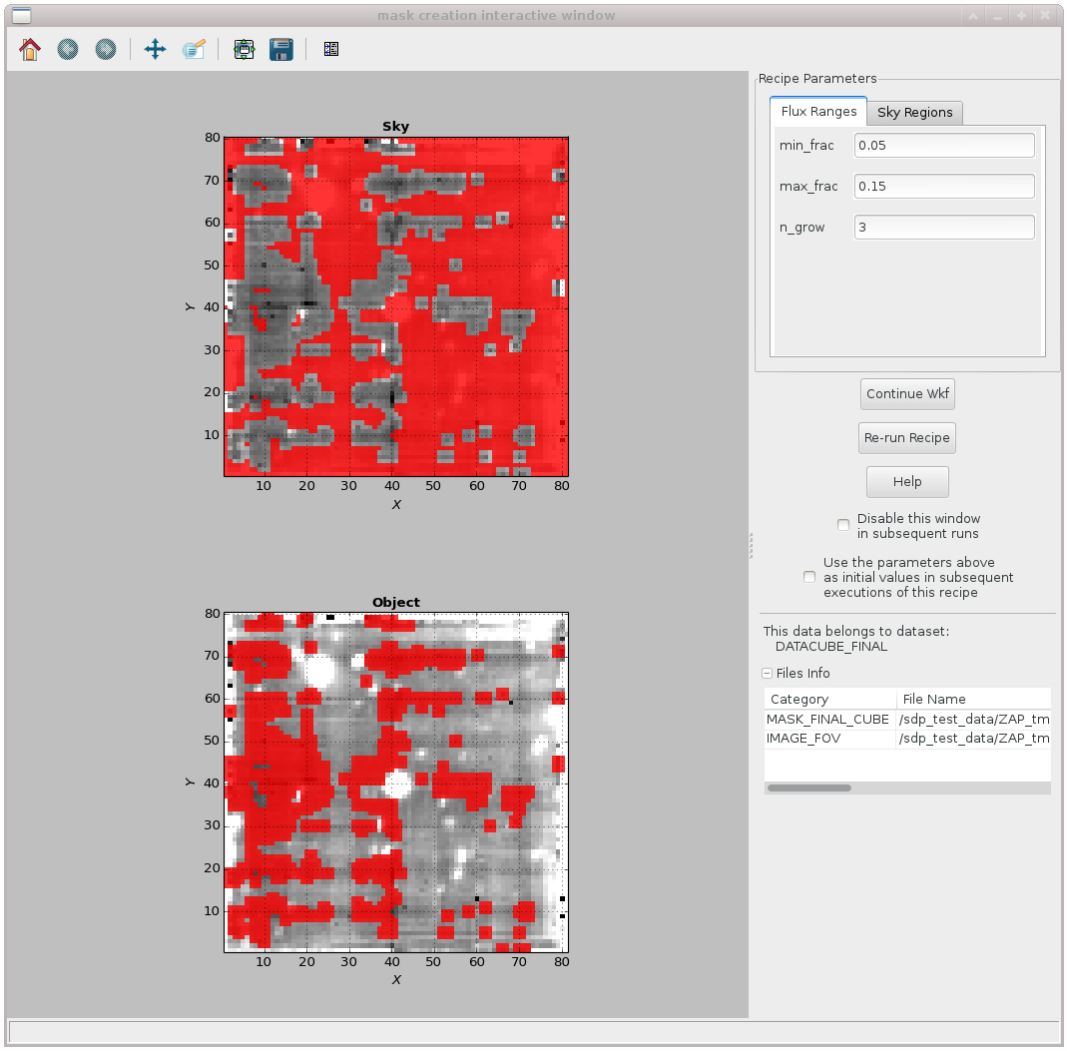


Figure 7.1: The interactive window associated to the CreateMask actor.

– **skybox***N* Up to *N* = 10 sky regions that can be added to the existing mask to include areas with only sky. Each region will be specified by 4 numbers indicating the xmin, xmax, ymin, and ymax pixel position in the field of view. An entry of -1 -1 -1 -1 will not add a sky region to the existing mask. Values needs to be entered without quotation marks and need be space separated. *Warning*:**n_grow** has no impact on the size of skyboxes. Default: *-1 -1 -1 -1*.

Tip: The suggested way to generate a custom sky mask is first to use the workflow to create one from the `IMAGE_OBJECT/SKY`. The mask will be saved into the reflex end products directory with the string `MASK_FINAL_CUBE` in its name. The user can either edit it (e.g., with the `imedit` task in `IRAF`) or copy it directly into the `INPUT_DATA_DIR` and then re-run the workflow to have the possibility to further modify it. It is important to remember that, when editing a mask via external tools, to maintain the same file structure, header content, spatial coverage and sampling as the original mask.

Tip: If you want to design your mask only using skyboxes, without any flux-based spaxel selection, you have to specify **min_frac**=**max_frac**=*0* and at least one **skybox**. It is not allowed to set **min_frac**=**max_frac**=*0* without specifying any skybox.

Warning: if you specify an external SVD file via the **External Sky components (full path)** parameter, the mask interactive window will not pop up because the mask is not needed.

7.2 Removing sky residuals via ZAP

The `Post Processing (ZAP)` actor executes the Zurich Atmosphere Purge (Soto et al. 2016) version 1.1 on the selected datacube to remove the residual sky contamination. The process is divided into two parts.

7.2.1 Calculation of the sky principal components (SVD)

The basic idea is to describe the sky residuals (onto areas that contain only sky) in terms of eigenspectra via Single Value Decomposition (SVD). The components specified in the SVD will be then fitted on the spectra containing the object in order to isolate the contribution of the sky residuals and subtract it.

This creation of a SVD file (category `ZAP_SVD` containing the sky principal components is performed directly either on the object datacube (in the actor `Removal of Sky residuals`) or on the sky datacube if present in the dataset (in the actor `Creation of SVD from Sky cube`).

The analysis is done on 11 wavelength segments (Table 1 in Soto et al. 2016), i.e. the `ZAP_SVD` contains the components for each of the segments.

In the process, only the portion of the sky selected by the input mask (created in the actor `CreateMask`) will be used.

7.2.2 Fit the sky principal components to the object

In this step, the components specified in the `ZAP_SVD` file are fitted to each spaxel in datacube to evaluate and remove the residual sky contamination. As explained in the previous section, the `ZAP_SVD` file can be computed either on the object datacube or on the sky datacube if present in the dataset.

Alternatively, a pre-calculated `ZAP_SVD` can be passed via the parameter **External Sky components (full path)**, which is editable from the main reflex canvas. The specification of this parameter will override any other `ZAP_SVD` eventually by the workflow.

7.2.3 Main parameters of the ZAP code

The parameters that regulate the ZAP code and that can be specified by the user (either from the main reflex canvas, or inside the ZAP composite actor) are the following:

- **Eigenspectra_detection_type (optimizeType)** Optimization method to compute the number of eigenspectra used for each wavelength segment. Accepted values are: *none*, *normal*, *enhanced*. If *none*, the number of eigenspectra must be specified with **nevals** or **pevals**. *normal* and *enhanced* refers to different algorithms to automatically compute the number of eigenspectra. Default: *normal*.
- **Percentage of eigenspectra(pevals)**.Allow to specify the percentage of eigenspectra used for each segment. Provide a single value that will be used for all of the segments. If **pevals** ≤ 0, the parameter **nvals** will be used instead (see below). Default: *1.*
- **number of eigenspectra(nvals)**. It specifies the number of eigenspectra to use when **optimizeType** = *none*. It is a single value that will be used for all of the segments. It has an effect only if **pevals** ≤ 0. Default: *30*.
- **External Sky components (full path)**. Path of an input FITS file containing a Singular Value Decomposition (SVD) computed on empty sky regions. *none* computes SVD on the sky regions specified in the sky mask. Specifying the SVD via a file makes the ZAP code much faster, this is useful when you need to optimize other parameters. Default: *'none'*.
- **median_filter(cfwidthSVD)**. Window size for the continuum filter, for the SVD computation. See section 2 of Soto et al. (2016). Default: *500*.
- **median_filter(cfwidthSP)**. Window size for the continuum filter used to remove the continuum features for calculating the eigenvalues per spectrum. Smaller values better trace the sources. An optimal range of is typically 20 – 50 pixels. Default: *30*.
- **zeroth sky removal (zlevel)**. Method for the zeroth order sky removal: *'none'*, *'sigclip'* or *'median'*. Default: *'median'*.
- **continuum filter method (ctype)**. Method for the continuum filter: *'median'* or *'weight'*. For the *'weight'* method, the definition of a zeroth order sky is required (see **zlevel**). Default: *'weight'*.
- **CleanNaN**. If True, the NaN values are cleaned. Spaxels with more then 25% of NaN values are removed, the others are replaced with an interpolation from the neighbors. The NaN values are reinserted into the final datacube. If set to False, any spaxel with a NaN value will be ignored. Default: *'True'*.

For a proper description of the ZAP code and all its parameters, we refer the interested reader to the its documentation (http://muse-vlt.eu/science/tools/) and publication (Soto et al. MNRAS, 458, 3210).

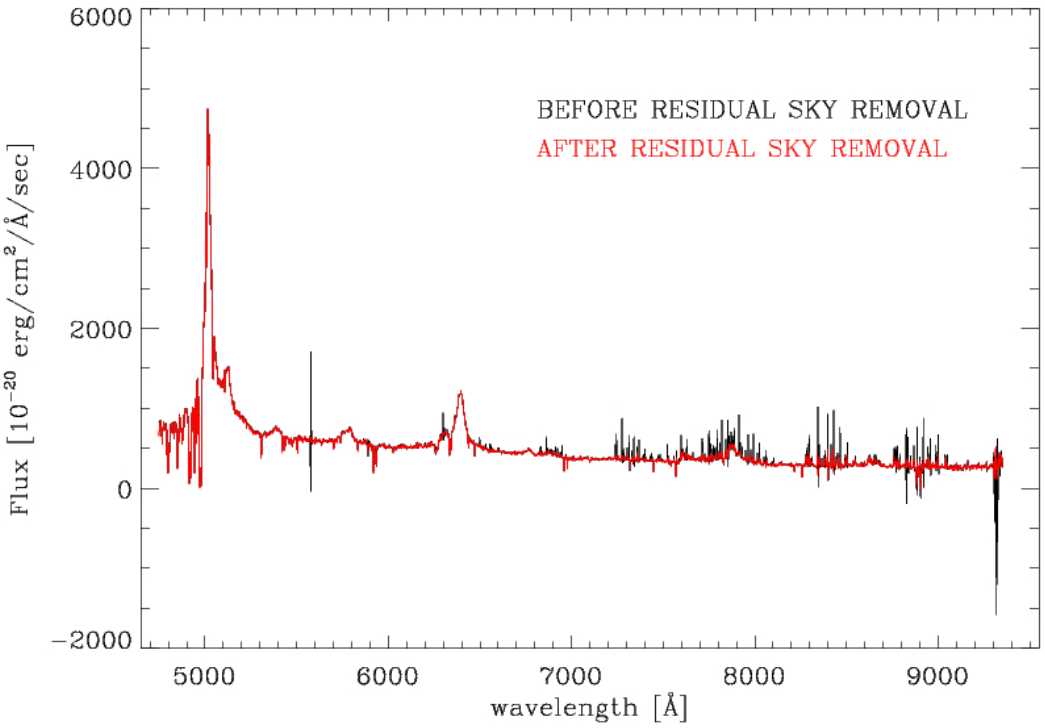


Figure 7.2: spectrum the central regions of the demo datacube (radius 5 pixels) before (black) and after (red) the post processing with ZAP. The used mask was obtained with **n_grow**=*1* **min_max**=*0.05*, **max_frac**=*0.15*, **skybox1**=*10 80 10 20*, and **skybox2**=*60 70 10 80*, respectively.

The user can use any datacube viewer (e.g., QFitsView; `www.mpe.mpg.de/~ott/QFitsView/`) to inspect the results.

Figure 7.2 shows the spectrum the central regions of the first demo datacube (radius 5 pixels) before and after the post processing with ZAP.

Warning: The user will be notified by a pop-up window in the case the code crashes for memory issues. See also Section 4.1.