European Organisation for Astronomical Research in the Southern Hemisphere

**Programme:** VLT

**Project/WP:** Science Data Quality group

# ESO Data Processing System (EDPS) Tutorial

**Document Number:** ESO-XXX

**Document Version:** 0.9.7

**Document Type:** Manual (MAN)

**Released on:** 2024-12-20

**Document Classification:** Public

Authors: L. Coccato, W. Freudling, S. Zampieri

Name

This page was intentionally left blank

**Change record**

| Issue/Rev. | Date | Section/Parag. affected | Reason/Initiation/Documents/Remarks |
|---|---|---|---|
| 0.9 | 20/10/2023 | All | First official release |
| 0.9.2 | 22/01/2024 | 2, 7, 8 | Installation, Update F.A.Q and Question and Feedback. |
| 0.9.3 | 01/08/2024 | 2.4, 5.1, 4.3.1 | Aligned with EDPS 1.3.3., ADARI 0.6.0 Better example of loading workflow and recipe parameter |
| 0.9.4 | 01/10/2024 | 2.2, 2.4 6.6 | Suggestion for shutdown and application.properties setup |
| 0.9.5 | 10/02/2025 | 2.2, 2.4 | Updated installation instructions |
| 0.9.7 | 15/10/2025 | Few typos corrected | |

This page was intentionally left blank

# Contents

# 1 Introduction

## 1.1 Scope

This document describes how to reduce data with ESO Data Processing System (EDPS). It is meant for first time user of EDPS, and guides through the installation procedure and data reduction using the ESPRESSO pipeline and its demo data as an example. After working through this tutorial, the reader should be able to use other supported EDPS workflows for the reduction of user provided data.

## 1.2 What is EDPS?

The ESO Data Processing System (EDPS) is a framework to run ESO's data processing pipelines and it is meant to eventually replace the previous EsoReflex environment (https://www.eso.org/sci/software/esoreflex/). The general principles of EDPS have been described by Freudling, Zampieri, Coccato et al. (2024, A&A, 681, A93). Please refer to that paper if you have used EDPS for research resulting in a scientific publication.

Each of ESO's data processing pipeline consist of a series of standalone programs called "recipes". Each recipe is designed to process certain type(s) of input data. The processing of these input data typically requires a range of auxiliary files such as calibration files. EDPS is designed to select appropriate input data for the different recipes of a pipeline, and execute them in sequence. This is done by specifying for each pipeline the workflow for organising data and executing the recipes. This workflow can the used to process a set of data fully automatically.

# 2 Installation.

## 2.1 Prerequisites

The installation of EDPS requires Python 3.9 (recommended) or higher. Linux and MacOS operating systems are supported. You can check the installed version of Python with the command

```
python --version
```

EDPS is pre-configured to run workflows for ESO data reduction pipelines. Only pipelines that include an EDPS workflow can be used by EDPS. As of this writing, the ESPRESSO, UVES, and KMOS pipelines include such a workflow. The instrument pipeline has to be installed following the instructions at https://www.eso.org/sci/software/pipelines/. That page also include a full up-to-date list of pipelines with EDPS workflow.

The pipeline recipe executer `esorex`, which is automatically installed with the pipeline, must be in the path so that EDPS can be find the pipelines. The command

```
which esorex
```

should return the full path to the installed `esorex` binary. EDPS will see only the workflows of the pipelines associated to that esorex binary.

## 2.2 Installation Procedure

Installing EDPS is quite simple and requires only few steps. There are 2 main ways to install EPDS, the first with Homebrew (recommended), the second via a Python virtual environment.

### 2.2.1 Installation via Homebrew

1. Install Homebrew. You can refer to the official documentation and Homebrew on Linux, but the installation is as simple as running the following command in your macOS Terminal or Linux shell prompt (single line):

   ```
   /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
   ```

   Then follow the instructions on the terminal about how to add Homebrew to your path.

2. Setup the ESO repository for Homebrew. The following command adds a custom Homebrew repository (Tap) which contains the pipeline packages.

   ```
   brew tap eso/pipelines
   ```

3. Installing EDPS. Type:

```
brew install edps
```

4. Uninstalling EDPS. Type:

```
brew uninstall edps
```

5. Installing a pipeline (e.g., ESPRESSO)

```
brew install esopipe-espdr
```

### 2.2.2 Installation via Python virtual environment.

From a bash shell:

- Define a new environment `edps`:
  `python3 -m venv <path_to_environment>/edps`
  where `<path_to_environment>` is a directory where the environment named `edps` is to be saved[1].

- Activate the environment named `edps`[2]:
  `. <path_to_environment>/edps/bin/activate`

- Upgrade `pip`:
  `pip install --upgrade pip`

- Install EDPS (single line command):
  `pip install --extra-index-url https://ftp.eso.org/pub/dfs/pipelines/repositories/stable/src edps adari_core`

  Note: if this is a re-installation, make sure that the EDPS server is not running in the background. To close the EDPS server, type `edps -shutdown` Alternatively, it can be killed with usual Linux commands.

Alternative, one can use a *conda environment; instructions are provided in A.

[1] In this example, we named the environment `edps`, but any other name can be used.

[2] For csh or tcsh shells use `source <path_to_environment>/edps/bin/activate.csh`, and for fish shell use `source <path_to_environment>/edps/bin/activate.fish` instead

## 2.3 First Run: EDPS configuration

Once EDPS has been installed, it can be run with the commands

- If EDPS was not installed via Homebrew: activate the previously defined edps environment[2]:
  `. <path_to_environment>/edps/bin/activate` (Section 2.2).
- Run edps:
  `edps`

The first time EDPS is executed, it will ask for a location where intermediate data products, bookkeeping information and logs will be stored. This should be a location with sufficient disk space to store the output data for several executions of the pipeline and needs full write permission. This location will be stored in '~/.edps/application.properties' and used for further calls of EDPS. EDPS will exit after this initial setup.

## 2.4 Update EDPS

Once EDPS is installed, it can be updated with the following command:

- If the installation was not done via Homwbrew: activate the previously defined edps environment:
  `. <path_to_environment>/edps/bin/activate`
- type (one line command):
  ```
  pip install --upgrade --extra-index-url
    https://ftp.eso.org/pub/dfs/pipelines/repositories/stable/src
    edps adari_core
  ```

Note: if this is a re-installation, make sure that the EDPS server is not running in the background. To close the EDPS server, type `edps -shutdown` Alternatively, it can be killed with usual Linux commands.

# 3 Data reduction with EDPS

In this Section we guide the user to the reduction of science data with an EDPS workflow. As example, we refer mostly to the ESPRESSO workflow. We assume that EDPS, the ESPRESSO pipeline with `esorex` and the workflow has been successfully installed and configured in your system (see Section 2), the `esorex` is in the system path. If the installation was done via Python virtual environment, then it needs to be activated as described in Section 2.2:

```
. <path_to_environment>/edps/bin/activate
```

We provide examples for other instrument workflows whenever needed to describe features that might not be available in the ESPRESSO workflow.

## 3.1 My first data reduction with EDPS

A simple reduction of all of the ESPRESSO data in a directory `input_directory` can be achieved with a single line command:

```
edps -w espresso.espresso_wkf
     -i <input_directory> -o <output_directory>
```

For example, the default location of ESPRESSO demo data when installing the ESPRESSO pipeline with RPMs is `/usr/share/esopipes/datademo/espdr`. EDPS scans the input directory recursively, i.e. the directory and all sub-directories are scanned for data. In addition, the static calibration directory delivered with the pipeline is used. Any FITS file found is classified, the science files are identified, and the best calibrations to be used are associated. All science exposures with a complete set of calibrations are then processed, and the final data products are copied in the `output_directory` (if specified). Intermediate files, calibration results, logs and book-keeping information are saved in the general EDPS directory specified during installation (see Section 2.3).

It is recommended to try this command on the demo data provided with the pipeline.

## 3.2 Closing EDPS

After the completion of the first `edps` (section 3.1) command, subsequent calls will start processing with reduced overhead. The reason for this is that EDPS starts a server component that is persistent and remains running in the background even when the processing is completed. Subsequent `edps` commands connect to this server. This mechanism improves the efficiency of the processing. Once no further processing is desired, this server can be closed with the following command:

```
edps --shutdown
```

One can also pass the option `-shutdown` at every command to close the server after the command execution.

## 3.3 Workflows for different pipelines

Depending on the instrument, the workflow reduces science data in different ways. The workflows are similar to the ones documented in detail in the EsoReflex tutorials for each pipeline, which are available at `http://eso.org/pipelines/`. Section 5.1 describes how to visualise the individual steps of the actually implemented EDPS workflows.

In the cases of ESPRESSO and UVES, individual science observations are processed but not combined. In the case of KMOS, the EDPS worflow first reduces and combines together all the science data that belongs to the same Observing Block, then it combines the results that refer to the same target name and instrument setup. In the case of MUSE, the EDPS workflow first reduces exposures separately and then it combines them according to preference expressed by the user: data from the same Observing Block, or data of the same target name, or data that fall within a certain distance on the sky (default).

EDPS processes the data following a default reduction strategy and using certain values for the recipe parameters (whose defaults can vary depending on the type of input data). It is possible to customise the data reduction strategy by changing the values of the recipe parameters, and eventually by activating or de-activating some reduction steps according to the science needs. All these options are instrument-dependent, see Section 4 for further details.

## 3.4 Location of reduced data and quality control plots.

EDPS saves the products into two directories. The first contains all recipe products, logs and plots (Section 3.4.1), whereas the second one, which is optional, contains only the results of the last reduction steps (Section 3.4.2).

### 3.4.1 Reduced data, logs, quality plots and book-keeping

All recipe products, including all logs, book-keeping and intermediate files are saved into the directory specified in Section 2.3 (default: `~/EDPS_data)`. This directory should not be deleted, even after the execution of EDPS. It is used for bookkeeping purposes. EDPS will use the files stored there to run efficiently. Among other things, EDPS will reuse products from that directory instead of re-executing a recipe whenever possible ("smart re-run"). The location of this directory can be changed by changing the value of `base_dir` parameter in the configuration file `~/.edps/application.properties`, or deleting the `~/.edps` directory and repeat the configuration step.

The files in this directory are organised according to the following tree structure. The first level contains directories named after the instrument used (e.g., ESPRESSO). The second level contains directories named after the various reduction steps, e.g., "dark", "flat", "bias", "object" and so-forth. The exact list depends on the instrument, the data present on disk, and the reduction strategy. The third level

includes each individual execution of a given task. For example, if the bias task was executed N times because there were N sets of bias to reduce, the bias directory will contain N subdirectories (a.k.a. job-directory), each of them containing the products and the book-keeping files (such as list of inputs, list of parameters used, recipe log and so-forth). The fourth and final level contains quality control plots that can be used to inspect the recipe products. Each job directory can have one or more sub-directories containing the quality control plots for that specific recipe execution, depending on the workflow.

### 3.4.2 Final products

If the output directory has been specified in the edps request via the `-o` option (see Section 3.1), then the products of the last reduction steps are copied in the output directory (see Section 6.4 for different methods). In the case of ESPRESSO, the last reduction step corresponds to the reduction of science target with the recipe `espdr_sci_red`. For other instruments, like MUSE and KMOS, also the combination of multiple exposures taken on different nights are included among the final products. To specify different final task(s) (also known as "target task"), see Section 5.3.

Only the fits files are stored in the output directory, all other information is still available in the general EDPS data directory described in Section 3.4.1. Data are organised first by dataset name. The dataset name is defined as the name of the first fits file that triggers the recipe. Inside each dataset directory, there are the results of different EDPS executions, identified by the time stamp of the EDPS request. If two reductions are identical (i.e. same inputs, same parameters), EDPS does not create a different time stamp directory.

The user can specify a different location for some products by setting the EDPS configuration file accordingly (see Section 6.4).

# 4 Customise the data reduction

Each step of the reduction cascade is organised in the so-called "tasks". In other words, a task represent a step in the data reduction. Each task is defined by input files and the recipe to execute. It is possible to define the reduction cascade up to a certain step, by selecting the "target task", or a "meta-target" (i.e. a defined group of tasks), so that the data reduction stops at a certain point.

In order to display the tasks for a certain workflow type:

```
edps -w espresso.espresso_wkf -lt
```

The output list shows the tasks grouped by meta-target. While a task can run only one recipe, a recipe can be associated to many tasks.

In order to execute the reduction up to a certain step, one has to indicate the "target task", or a meta-target (this will select multiple "target tasks", i.e. those that are within the same meta-target group). Type:

```
edps -w espresso.espresso_wkf -t <TASK>
```

for a single "target task", or:

```
edps -w espresso.espresso_wkf -m <meta-target>
```

for multiple target tasks, within meta-target group. More information on Section 5.4.

## 4.1 Recipe parameters

To process data with specific value for a parameter, which is executed by a recipe of a given task, type on a terminal where the edps Python environment is active (single line command):

```
edps -w espresso.espresso_wkf -i <input_directories>
     -rp <TASK> <PARAMETER> <VALUE>
     -o <output_directory>
```

Where TASK is the task name that runs the recipe we want to change the parameter for; PARAMETER is the parameter name, and VALUE is the value we want to use. The PARAMETER name must give the full name, which includes the instrument name and the recipe name (see Section 4.3 on how to display the available recipe parameters).

For example, the command (single line):

```
edps -w  espresso.espresso_wkf -i <input_directories>
     -rp object espdr.espdr_sci_red.cosmic_detection_sw 1
     -o <output_directory>
```

instructs the task `object` to activate the Laplacian Cosmic Ray detection algorithm when running the recipe `espdr_sci_red`.

To change more than one parameters, just add other `-rp` lines to the example above. If many parameters have to be configured, it might be convenient to load them from a configuration file (see Section 4.4)

For a given instrument workflow, the full list of options is described in the corresponding pipeline manuals and data reduction tutorials.

*Note:* The recipe parameters defined through the command line as above override:

- default recipe values;
- configuration files with specified values;
- values that are hard-coded or configured automatically by the workflow (e.g., that depend on input data).

## 4.2 Workflow parameters: select the most appropriate data reduction strategy

Some instrument workflows can reduce the data in different ways, depending on the science needs. Certain steps of the reduction can be avoided or executed instead of others, or some calibrations can be ignored despite being present in the input data directory.

Some of the data reduction strategies are hard-coded in the workflow, in the sense that the data reduction cascade depends on the properties of the data we want to process. On the other hand, some other strategies can be controlled by so called "workflow parameters." In other words, workflow parameters are not directly associated to the recipes but they define the strategy of the data reduction and the reduction chain. In the case of ESPRESSO, there is only one reduction strategy and no workflows parameters are available. In the case of the KMOS workflow, the following workflow parameters are available:

- molecfit: 'standard' (default), 'science', 'false'.
- use_sky_flats: 'false' (default), 'true'.
- qc0: 'false' (default), 'true'.

For example, to use the sky flats for illumination correction instead of using the lamp flats as default strategy, type (single line command):

```
edps -w kmos.kmos_wkf -i <input_directories>
     -wp use_sky_flats 'true'
     -o <output_directory>
```

For a given instrument workflow, the full list of options is described in the corresponding pipeline manuals and data reduction tutorials.

To specify more workflow parameters, add as many `-wp` lines as needed. If a large number of workflow parameters have to be specified, it might be convenient to load them from a configuration file (see Section 4.4)

*Note:* Workflow parameters defined through the command line as above, override:

- values defined in configuration files;
- values that are hard-coded or configured automatically by the workflow (e.g., that depends on input data).

## 4.3 Display default parameter values

To display what are the recipe parameters full names and their values for the task `object` used in the default parameter set (note: the ESPRESSO pipeline must be installed):

```
edps -w espresso.espresso_wkf -p object
```

In the case there are several parameter sets, add the name of the set for which you would like to the parameters. In the ESPRESSO workflow there is only a parameter set. But in the case of KMOS, one can show the information for the parameter set: `qc0_parameters`, which is not the default one:

```
edps -w kmos.kmos_wkf -p object qc0_parameters
```

In general, to display simultaneously the information for all the parameter sets, type:

```
edps -w espresso.espresso_wkf -ps
```

### 4.3.1 Use certain workflow and recipe parameter sets

Each workflow can have different sets of parameters. There are two types of parameters: workflow parameters and recipe parameters. In the case of KMOS, there is a non-default parameter set named `qc0_parameters`. In order to use the workflow parameters from that set, type:

```
edps -w kmos.kmos_wkf -wps qc0_parameters
```

In order to use the recipe parameters from that set, type:

```
edps -w kmos.kmos_wkf -rps qc0_parameters
```

In order to use both the workflow and recipe parameters from that set, type:

```
edps -w kmos.kmos_wkf -wps qc0_parameters -rps qc0_parameters
```

### 4.4 Configuration file for recipe and workflow parameters.

Each workflow comes with a configuration file that contains recipe and workflow parameters. The recipe parameters are organised by task (indeed, several tasks can run the same recipe and might require different parameter values). If a recipe parameter is not listed, then the pipeline default is used. Each parameter file can contain more than one parameter set, therefore the users can define the set with the parameters that are most suited for their reduction.

The configuration file is in `yaml` format, therefore it must follow some conventions. Please use the default configuration file as starting point to create new ones. General rules that one has to keep in mind are:

- Booleans has to be indicated as strings (e.g. "FALSE", "TRUE").
- Values has to be specified via column, not via equality. E.g.:
  `espresso.espdr_sci_red.cosmic_detection_sw: 2`
- Recipe parameters have to be specified by their full name, that follows the convention
  `<instrument>.<recipe>.<alias>`

Note that the direct specification of a parameter value in the `edps` command overrides the values defined in the parameter file (see Section 4).

# 5 EDPS most useful options

To get the EDPS full list of options, type on the terminal where the EDPS environment is active:

```
edps -h
```

In the following Sections, we describe the most useful options.

## 5.1 Graphic representation of a workflow.

One convenient way to understand a workflow is to look at a graphical representation. EDPS supports 2 levels of graphic display. The first shows datasets, tasks, and subworkflow (general graph). The second shows tasks, subworkflows, the input categories and the recipes (detailed graph). These graphs can be produced with the `-g` and `-g2` options of EDPS. The graphs are produced in the `dot` format. This format can be converted into a large range of formats by the `dot` program. This program needs to be installed separately and is available in all commonly used Linux and MacOs versions.

For example, to produce a workflow graph for the ESPRESSO workflow in pdf format, type from a terminal where the edps environment is active:

```
edps -w espresso.espresso_wkf -g | dot  -Tpdf > espresso.png
```

For `.ps` or `.png` format, use `-Tps` or `-Tpng`, respectively.

The program `dot` [3] has many option that can be used to produce presentations of the workflow depending on the preferences of the user. For example, to produce a detailed workflow graph for the ESPRESSO workflow, type from a terminal where the edps environment is active (single line command):

```
edps -w espresso.espresso_wkf -g2 | dot  -Tpdf -O
```

This generally produces several output files, one with the workflow and the other ones with details of each sub-workflow[4]. A convenient way to to merge the various created files into a unique .pdf file is:

```
gs -sDEVICE=pdfwrite -sOutputFile=espresso.pdf -dNOPAUSE -dBATCH noname*pdf

rm noname*pdf
```

[3] https://graphviz.org/

[4] If a workflow does not contain any sub-workflows, then only one file is generated, i.e. noname.gv.pdf. In this case, the file can be simply renamed without the need to merge multiple .pdf files.

### 5.2 Classification of input data.

To show only the classification of the input data, type: from a terminal where the edps environment is active:

```
edps -w espresso.espresso_wkf -i <input_directory> -c
```

EDPS inspects the `<input_directory>` recursively, and prints a list of the fits files and their classifications. In general, a file can have more than one classification.

### 5.3 Reduce the data until a certain step

EDPS can perform the reduction up to a certain processing step, which we call the "target task". Only input data related to that task (and needed calibrations) will be processed.

To reduce data until the task `flat`, i.e., reduction of flat field raw calibrations (single line command):

```
edps -w espresso.espresso_wkf -i <input_directory>
     -t flat -o <output_directory>
```

To see the list of processing tasks, type on the terminal where the EDPS environment is active:

```
edps -w espresso.espresso_wkf -lt
```

A list of tasks grouped by the so-called "metatargets" is shown.

To process data of the tasks that belong to the same "metatarget" (single line command)

```
edps -w espresso.espresso_wkf -i <input_directory>
     -m <METATARGET> -o <output_directory>
```

For example, :

```
edps -w espresso.espresso_wkf -i <input_directory>
     -m qc1calib -o <output_directory>
```

executes only calibration tasks. By default, if no target tasks or metatargets are specified, EDPS assumes the default option `-m science`, i.e. all science tasks are considered targets of the reduction.

Note:, EDPS organises the data and reduces them (assuming `-m science` as meta-target). It is possible to perform only the data classification in order to check the content of datasets/

Note: when the output directory is specified with the `-o` option, only the products of the (meta)target tasks are saved into the final directory. All other products, including logs and book-keeping, are stored in the general EDPS data directory, as specified during the configuration (Section 2.3).

### 5.4 Inspect the reduction cascade and datasets content (without starting the data reduction).

By default, if a workflow and the input data directories are specified, EDPS organises the data into datasets and processes them, according to the (meta)target tasks. It is possible to stop at the data organisation, and visualise the content of the datasets with the option `-f`. The data organisation is done up to the (meta) target tasks (default is `-m science`, i.e. all the science tasks).

To perform the data organisation only: EDPS up to a certain task type (for example, up to the flat fielding):

```
edps -w espresso.espresso_wkf -i <input_directory>  -t flat  -f
```

One can direct the output (json format) to a file and open it with a browser

```
edps -w espresso.espresso_wkf -i <input_directory>
     -t flat -f  > flat_field_datasets.json

firefox flat_field_datasets.json
```

Notes:

- With the option `-f`, the datasets are organised in a tree-like structure. If one replaces `-f` with `-od`, then each job is displayed independently with more information, but the associations between the various calibrations are not highlighted.
- If the `-od` and `-f` options are omitted in the previous example, EDPS processes all the flat fields (and needed calibrations) in the input directory. The steps after the reduction of flat fields are not executed.

# 6 How to configure EDPS: the `application.properties` file.

Several options in EDPS can be specified in a configuration file, named `application.properties`. This file is located in the `.edps/` directory in your HOME directory (see Section 2.3). If a feature can be specified both in the command-line request and in the configuration file, the command line request has priority. Note: to make effective the changes done in application.properties, one must first restart the EDPS server. To close the server, type (from a terminal with the EDPS environment active):

```
edps -shutdown
```

## 6.1 Association preference: `RAW` vs `MASTER` calibrations.

If the input directory contain both `MASTER` (e.g., pre-reduced calibrations) and `RAW` calibrations, it could happen that both of them fulfil the matching criteria and quality level (see Section **??**) for a certain task. In this case, one can specify to which type of calibration to give priority by setting the variable `association_preference` in the `application.properties` configuration file.

Possible values of `association_preference` are:

- `raw`. First, EDPS checks if there are raw calibrations ensuring the first quality level of the products. If found, they are associated. If not found, raw calibrations ensuring the second quality level of the products are searched. If not found, the next level is searched until the last quality level permitted by the workflow parameter `quality_threshold` is reached (see Section **??**). If no raw calibrations are found for none of the quality levels, then EDPS searches for master calibrations, starting from those ensuring the first quality level. If none are found, the second level is searched, and so forth. If no calibrations are found, the association is not done.
- `master`. Same as `raw`, but first master calibrations are looked for all the products quality levels permitted by the workflow parameter `quality_threshold` (see Section **??**). Then, if master calibrations are not found, the system looks for raw calibrations.
- `raw_per_quality_level` (default). First, the system will check if there are raw calibrations ensuring the first quality level of the products. If not found, MASTER calibrations ensuring this level are searched for. If not found, RAW calibrations ensuring the second quality level are searched for, if not found MASTER calibrations matching the second quality level are searched for. The sequence goes on until the last level permitted by the workflow parameter `quality_threshold`.
- `master_per_quality_level`. Same as `raw_per_quality_level`, but with inverted roles for MASTER and RAW calibrations.

If a combination of `RAW` and `MASTER` calibrations are present, the value of `association_preference` might have an impact on the performances and the quality of the results. Typically. `association_preference = raw_per_quality_level` delivers the best quality products, at the price of speed. On the other hand, `association_preference = master` ensures faster performances, at cost

of quality (e.g., a very old master calibration could be used instead a more recent raw calibration). If only `RAW` or `MASTER` calibrations are present in the input directories, then the value of `association_preference` has no impact.

## 6.2 Running recipes in parallel

One of the advantages of EDPS is that it can exploit of powerful hardware. The following variables in the `application.properties` file determine the pararellization of EDPS reduction.

- `processes` (default: 1). It specifies the maximum number of jobs to run in parallel (e.g. `esorex` parallel executions) .
- `cores` (default: 1). It specifies the maximum numbers of computers cores to use, considering all the parallel jobs.
- `default_omp_threads` (default: 1). The number of cores to use for each job. This can be overridden by specifying a recipe parameter `OMP_NUM_THREAD` for a given task. (e.g. `-rp object OMP_NUM_THREAD 3` to assign a desired number of threads to the `object` task, whereas all the others use the default value.

## 6.3 Order of executions.

The variable `ordering` in the `application.properties` file specifies the priority to give to the reduction jobs. The most important values are:

- `dfs`. depth-first, gives preference to reaching final reduction target quicker. In other words, it finish the reduction of a dataset before moving to the next dataset. This choice is less efficient in time but it gives priority to the reduction of individual datasets.
- `type`. It gives preference to following the reduction cascade level by level making sure to process same type of data together (eg. first all Biases).
- `dynamic`. Immediately runs whichever job is ready (has all needed inputs), no stalling but the order is unpredictable. This is the most time efficient execution order.

## 6.4 Renaming products file names

If the user specified an output directory via the `-o` option in the EDPS request (see Section 3), the products of the target task(s) (i.e. the final steps in the processing cascade) are saved in the specified directory. Their names are given by the `pattern` variable in application.properties.

The users can decide to copy or hard link certain product categories into a different location and with a different naming convention. This can be done by setting the following variables in the application.properties configuration file:

- `mode`. (Values: `copy`/`link`) Specifies if the products have to be copied or (hard) linked into the output directory. Default: copy.
- `categories`. List of categories (i.e. the `HIERARCH ESO PRO CATG` header keywords of the recipe products) that have to be copied or linked.
- `pattern`. Pattern to follow for the saving and naming convention. The default value is:
  `pattern = $DATASET/$TIMESTAMP/$object$_$pro.catg$.$EXT`.

  The following predefined variables can be used:
  `$TASK`: name of the task generating the product.
  `$DATASET`: name of the dataset.
  `$TIMESTAMP`: time of the request to EDPS.
  `$EXT`: file extension (.fits).
  `object` and `pro.catg` in the default values represents the `OBJECT` and `HIERARCH ESO PRO CATG` header keywords of the file.

  Values from header keywords and general text can be added, bracketed by `$`. For example, the following setup:

```
package_base_dir=~/my_reduction/
mode=link
pattern=$TASK/$DATASET/$TIMESTAMP/\$object\$\_\$pro.catg\$_reduced.$EXT
categories=S1D_FINAL_A
```

  will (hard) link the category `S1D_FINAL_A` produced by the pipeline recipe `espdr_sci_red` into the directory `~/my_reduction/`. Files are organised into subdirectories that specify task name, the dataset name, the creation date. The names of the files contain the value of the `OBJECT` and `HIERARCH ESO PRO CATG` keywords, as specified in the product header.

## 6.5 Reprocessing a given set of datasets

If you want to test which combination of options in `${HOME}/.edps/application.properties` config file works best for you by processing a given data set several times with different values of one or more parameters in the config file, you need to set the config parameter `truncate` to `True` and restart the server each time, or more precisely, shutdown the server after each invocation of edps. i.e.:

1. edit `${HOME}/.edps/application.properties`
2. run `edps`, e.g.
   ```
   edps -w espresso.espresso_wkf -i <input_directory> \
     -o <output_directory> \
   ```
3. stop the edps server,
   ```
   edps --shutdown
   ```
4. repeat from step 1 for each combination of config parameter values you want to test

### 6.6 Changing the pipeline

EDPS runs the workflows associated to the `esorex` command that is in the system path. In order to change the workflow and or the pipeline, edit the following variables in the application.properties file so that they point to the location of the workflow and esorex command associated to the desired pipeline.

```
workflow_dir=          #insert path to the new workflow directory
esorex_path=esorex     #insert path of the new executable
```

In order to these changes to have effect, the edps server has to be shutdown.

```
edps -shutdown
```

# 7 Frequently asked questions

- **Q:** I get an error message "workflow not found", and/or nothing happens after I submitted the `edps` command with my workflow. How do I fix it?
  **A:** This should not happen if the recommended installation procedures for edps and pipelines have been followed. If the installation was done in a non-standard way, the "workflow not found" message could indicate that:
  - the `esorex` command is not associated to the instrument pipeline you want to use.
  - The installed pipeline does not have an EDPS workflow yet.
  - The workflow name is mispelled.

  Type command `esorex --recipes` to check which pipeline is seen by `esorex`. If there are no recipes for your instrument, it means that path in `application.properties` points to the wrong `esorex` installation. Fix it and restart the edps server. Type the command `edps -lw` to list the installed workflows. If the workflow is not present, please check the spelling or the workflow directory in the `application.properties` file. See Section 6.6
- **Q:** The association reveals that my datasets are not complete, but I think to have all the needed data. How do I fix it?
  **A:** It could be that EDPS could not find the location of static calibrations for that instrument pipeline (this could happen, for example, if the instrument pipeline was not installed following the recommended procedures). Try to add the static calibration directory to the list of input data via the `-i` option.
- **Q:** I do not remember the name of the workflow I want to run. How do I know the exact workflow names?
  **A:** EDPS workflows are installed together with the instrument pipeline installation. To see which are the workflows installed in your system and their names, type: `edps -lw`.
- **Q:** I edited the configuration file application.properties, but this seems to have no effect. Why?
  **A:** In order for the changes to take effect, first close the EDPS server by typing the command `edps -shutdown`, and then relaunch the request.
- **B:** How do I know which recipe parameters were or will be used?
  **A:**. The book-keeping directory have the file parameters.rc, which indicates only the values of the recipe parameters that are different from the recipe default. The products have record of the recipe parameters values in their header. For inspecting the parameters values that are going to be used in the reduction *before* the reduction stars, consult Section 4.3.

# 8 Questions and feedback.

For suggestions, questions, or feedback in general, please open a ticket with the EDPS Support team. This link should take you directly to a webpage for creating and EDPS feedback ticket, but incase you want to navigate there 'manually', go to https://support.eso.org, login, click on "Submit Helpdesk Ticket", and specify the Help topic: "Post Observations", "ESO Data Processing System [EDPS]".

# A Installing in a `conda` environment

Assuming that `conda` is installed and activated...

- define a new `conda` environment named `edps`:
  `conda create -n edps python=3.10`
- activate the `edps` environment:
  `conda activate edps`
- upgrade `pip`:
  `python -m pip install --upgrade pip`
- install EDPS (single line command):
  ```
  python -m pip install --extra-index-url \
     https://ftp.eso.org/pub/dfs/pipelines/libraries edps adari_core
  ```
- to remove the `edps conda` environment at anytime:
  `conda env remove --name edps`

Then to run `edps` at any time in the future, activate the `edps` environment, if it is not already activated, and then simply invoke `edps`, e.g. the first time to create and setup the `applications.properties` config file:

```
edps
```