Small EDPS Tutorial

Benjamin Eisele

September 21, 2025

Contents

1 Introduction

This small tutorial aims to give a concrete list of steps one can follow to produce a simple data reduction workflow set up with EDPS. It does NOT show how to install/use existing ESO EDPS workflows. It's intention is to be practical example without showing/explaining all of the features of EDPS, as they are already documented in the official resources by ESO.
Disclaimer: Fits files required to run the created workflow are not provided.

2 Installation

Disclaimer: The installation section only covers the procedure for systems running Ubuntu/Debian. Fedora and MacOS based systems have their own pre-build binaries. Other Unix systems may require manual compilation of the dependencies.
This section aims to give a straight-forward list of instructions to install PyCPL and PyESOREX (needed for running the data processing), as well as EDPS itself. Detailed information with examples and troubleshooting advice can be found at `https://www.eso.org/sci/software/pycpl/`

pycpl-site/.
Note: *For users that prefer containers/reproducible environments or simply want to skip the installation procedure, there is a Dockerfile available at https://github.com/AstarVienna/LFOA_Pipeline/blob/main/podman_container/Containerfile. . Users can then skip to section 3.*

2.1 Setting up the environment

In a terminal of your choice run the following commands in order. These update the system and install all the needed dependencies for the installation:

```
sudo apt update && sudo apt upgrade %Getting the system up to date
sudo apt install python==3.10 gcc libcpl-dev
sudo apt install python3-dev python3-venv
```

The next set of commands creates a virtual python environment in your home directory and installs the python packages with pip. If you prefer another program (like anaconda) to manage environments you can do so here.

```
% In case you get an error with the python command not working, try
    replacing it with python3.
python -m venv ~/edps_environment
. ~/edps_environment/bin/activate
python -m pip install --upgrade pip setuptools build wheel
python -m pip install pybind11 astropy
% The following two lines seem to not be necessary anymore, but we've
    included it for the sake of completion.
export CPLDIR=/usr/lib
export LD_LIBRARY_PATH=$CPLDIR/lib64:$CPLDIR/lib:$LD_LIBRARY_PATH
```

2.2 Installing PyCPL & PyEsorex

Next we create a directory where you will store your recipes. We need to direct PyCPL to this directory via a environment variable. This path will be included into the binary that will be build. You can however change the directory at a later date by setting a different one, or add it to your .bashrc/.zshrc so it persists.

```
mkdir ~/Documents/Recipes %Or another location if you prefer
export PYCPL_RECIPE_DIR=~/Documents/Recipes
python -m pip install --extra-index-url https://ftp.eso.org/pub/dfs/
    pipelines/libraries pycpl pyesorex
```

With this PyCPL and PyEsorex are installed. To make sure everything works so far run:

```
pyesorex --version
```

2.3 Installing EDPS

This step is fairly straight-forward. Still being in our virtual environment we run:

```
python -m pip install --extra-index-url https://ftp.eso.org/pub/dfs/
    pipelines/repositories/stable/src edps adari_core
```

This concludes the installation.

3 Getting started with EDPS

3.1 Setting up EDPS

With all the prerequisites installed we can now start setting up EDPS. This consists of editing a config file with a text editor of choice (Vim, Nano, Emacs, etc...) to set some paths so that EDPS can find out workflows.
Firstly we create a directory structure that EDPS expects.

```
mkdir ~/Documents/workflows
mkdir ~/Documents/workflows/my_wkf
```

Here **my_wkf** is most of the time the instrument name. Next we run EDPS for the first time. This is so it creates the necessary config files.
It will ask a couple questions if you want to change default paths. Just hit enter unless you want to change something.

```
edps
```

Now we can edit the config file.

```
vim ~/.edps/application_properties
```

Insert the path to the workflow directory we have created earlier. Additionally change **esorex** to **pyesorex**, as we are working with the Python implementation. It should then look like it's shown in figure You can exit the file now (don't forget to save!). With this you are ready to write your own workflows and recipes.

3.2 Writing simple recipes (Bias, Dark and Flat)

The Python code present in the Appendix utilizes PyCPL for a Bias subtraction, a Dark current correction, Flat fielding, as well as applying each of the Master Calib frames to the raw Science images. It is important to note that this only includes code which is required by PyCPL without utilizing any of the more advanced functions that are provided by the module. Each of these should be placed in individual python files in the Recipes directory we created earlier. They can be found in the Appendix sections A, B, C, D.

3.3 Writing the Workflow

We will split up the workflow in three individual Python files. These are separated in Classifications, Data Sources as well as the Tasks which make up our pipeline. Each of these files will be placed in the **my_wkf** directory we created earlier. One important thing to note is that the Python file which contains the tasks must end with ***_wkf.py** for EDPS to recognize it.

3.3.1 Classification rules

```
from edps import classifictaion_rule

#Change the IMAGETYP value to whatever your fits files use for
    identification
bias_class = classification_rule("BIAS", {"IMAGETYP": "bias"})
dark_class = classification_rule("DARK", {"IMAGETYP": "dark"})
prep_class = classification_rule("FLAT", {"IMAGETYP": "flat"})
science_class = classification_rule("SCIENCE", {"IMAGETYP": "object"})
```

3.3.2 Data Sources

```
    raw_bias = (data_source("BIAS")
                .with_classification_rule(bias_class)
                .with_grouping_keywords(["DATE-OBS"])
                .with_match_keywords(["ORIGIN"])
                .build())
    raw_darks = (data_source("DARK")
                .with_classification_rule(dark_class)
                .with_grouping_keywords(["DATE-OBS"])
                .with_match_keywords(["ORIGIN"])
                .build())
    raw_prep = (data_source("FLAT")
                .with_classification_rule(prep_class)
                .with_grouping_keywords(["DATE-OBS"])
                .with_match_keywords(["ORIGIN", "FILTER", "EXPTIME"])
                .build())
    raw_science = (data_source("SCIENCE")
                .with_classification_rule(science_class)
                .with_match_keywords(["FILTER"])
                .build())
```

3.3.3 Tasks

```
    bias_task = (task("bias")
                .with_recipe("bias_processor")
                .with_main_input(raw_bias)
                .build())

    dark_task = (task("dark")
                .with_recipe("dark_processor")
                .with_main_input(raw_darks)
                .with_associated_input(bias_task)
                .build())

    flat_task = (task("flat")
                .with_recipe("flat_processor")
                .with_main_input(prep_task)
                .with_associated_input(bias_task)
                .with_associated_input(dark_task)
                .build())

    science_task = (task("science")
                    .with_recipe("science_processor")
                    .with_main_input(raw_science)
                    .with_associated_input(bias_task)
                    .with_associated_input(dark_task)
                    .with_associated_input(flat_task)
                    .build())
```

4 Using EDPS

Now that everything is set up it's finally time to try out the workflow we just created. Two checks we can run to make sure everything is set up correctly is running the following commands:

```
    pyesorex --recipes
```

and

```
    edps -lw
    edps -w my_wkf.my_wkf -lt
```

The **pyesorex** command lists all of the three recipes we wrote earlier. The two EDPS commands list our workflows, which is only one, as well as all of the tasks which are contained in the specified workflow. Now to execute our workflow we run the following command. We assume our fits files are contained in a directory called **data** in our Documents directory.

```
    edps -w my_wkf.my_wkf -i "~/Documents/data" -t science
```

This will execute a cascade the runs the science task as well as all the tasks which it requires. The created fits files, as well as log files, will be saved in a **EDPS_Data** directory which is automatically created in the home directory.

5 Troubleshooting

If you use the exact code shown here on the recommended systems everything should work and you could start writing more complex workflows. If however an Error was thrown regardless, one of the following might be of help. If not, then make sure to also check the troubleshooting sections of the official ESO websites.

1. No jobs are created:
   This error could stem from EDPS not finding the fits files. The reason could be a wrong value in the classification rules or a missing header value in the data. To check if this is the case run the following:

   ```
           edps -w my_wkf.my_wkf -i "~/Documents/data" -c
   ```

   This will list all of fits files in the directory as well as if a matching classification applies.

A Bias

```
import cpl.core
import cpl.ui
import cpl.dfs
import cpl.drs

from typing import Any, Dict

class BiasProcess(cpl.ui.PyRecipe):
    _name = "bias_processor"
    _version = "0.1"
    _author = "EDPS User"
    _email = "test@testmail.com"
    _copyright = "GPL-3.0-or-later"
    _synopsis = "Basic Bias processor"
    _description = "This recipe takes a number of raw bias frames and produces
        a master bias."

    def __init__(self):
        self.parameters = cpl.ui.ParameterList(
            (
            )
        )
    def run(self, frameset: cpl.ui.FrameSet, settings: Dict[str, Any]) -> cpl.
        ui.FrameSet:
        for key, value in settings.items():
            try:
                self.parameters[key].value = value
            except KeyError:
                cpl.core.Msg.warning(
                    self._name,
```

```
                f"Settings includes {key}:{value} but {self} has no
                    parameter named {key}.",
            )

    output_file = "MASTER_BIAS.fits"

    raw_bias_frames = cpl.ui.FrameSet()
    product_frames = cpl.ui.FrameSet()

    for frame in frameset:
        if frame.tag == "BIAS":
            frame.group = cpl.ui.Frame.FrameGroup.RAW
            raw_bias_frames.append(frame)
            cpl.core.Msg.debug(self.name, f"Got raw bias frame: {frame.file
                }.")
        else:
            cpl.core.Msg.warning(
                self.name,
                f"Got frame {frame.file!r} with unexpected tag {frame.tag!r
                    }, ignoring."
            )

    if len(raw_bias_frames) == 0:
        cpl.core.Msg.error(
            self.name,
            f"No raw frames in frameset."
        )

    header = None
    processed_bias_images = cpl.core.ImageList()

    for idx, frame in enumerate(raw_bias_frames):
        cpl.core.Msg.info(self.name, f"Processing {frame.file!r}...")

        if idx == 0:
            header = cpl.core.PropertyList.load(frame.file, 0)

        cpl.core.Msg.debug(self.name, f"Loading bias image {idx}...")
        raw_bias_image = cpl.core.Image.load(frame.file)

        processed_bias_images.insert(idx, raw_bias_image)

    combined_image = processed_bias_images.collapse_median_create()

    product_properties = cpl.core.PropertyList()
    product_properties.append(
        cpl.core.Property("ESO PRO CATG", cpl.core.Type.STRING, r"
            OBJECT_REDUCED") #This is needed for CPL to not throw an error!
    )

    cpl.core.Msg.info(self.name, f"Saving product file as {output_file!r}."
        )

    cpl.dfs.save_image(
        frameset,
        self.parameters,
        frameset,
        combined_image,
        self.name,
        product_properties,
        f"demo/{self.version!r}",
        output_file,
            header=header,
        )

        product_frames.append(
            cpl.ui.Frame(
                file=output_file,
                tag="MASTER_BIAS",
                group=cpl.ui.Frame.FrameGroup.CALIB,
            )
        )

        return product_frames
```

B Dark

```
import cpl.core
import cpl.ui
import cpl.dfs
import cpl.drs
import re

from typing import Any, Dict

class DarkProcess(cpl.ui.PyRecipe):
    _name = "dark_processor"
    _version = "0.1"
    _author = "EDPS User"
    _email = "test@testmail.com"
    _copyright = "GPL-3.0-or-later"
    _synopsis = "Basic Dark processor"
    _description = "This recipe takes a number of raw dark frames, subtracts
       the bias and produces a master dark."

    def __init__(self):
        self.parameters = cpl.ui.ParameterList(
            (
            )
        )

    def run(self, frameset:cpl.ui.FrameSet, settings: Dict[str, Any]) -> cpl.ui
       .FrameSet:
        for key, value in settings.items():
            try:
                self.parameters[key].value = value
            except KeyError:
                    cpl.core.Msg.warning(
                        self._name,
                        f"Settings includes {key}:{value} but {self} has no
                           parameter named {key}.",
                    )

        output_file = "MASTER_DARK.fits"

        pattern = r'value\s+:\s+(\d+)'

        raw_Dark_Frames = cpl.ui.FrameSet()
        bias_frame = None
        product_frames = cpl.ui.FrameSet()
```

```
for frame in frameset:
    if frame.tag == "DARK":
        frame.group = cpl.ui.Frame.FrameGroup.RAW
        cpl.core.Msg.debug(self.name, f"Got raw dark frame: {frame.file
            }.")
        raw_Dark_Frames.append(frame)
    elif frame.tag == "MASTER_BIAS":
        frame.group = cpl.ui.Frame.FrameGroup.CALIB
        cpl.core.Msg.debug(self.name, f"Got master bias frame: {frame.
            file}.")
        bias_frame = frame
    else:
        cpl.core.Msg.warning(
            self.name,
            f"Got frame {frame.file!r} with unexpected tag {frame.tag!r
                }, ignoring."
        )

if len(raw_Dark_Frames) == 0:
    cpl.core.Msg.error(
        self.name,
        f"No raw frames in frameset."
    )

processed_dark_images = cpl.core.ImageList()

if bias_frame:
    bias_image = cpl.core.Image.load(bias_frame.file)

for idx, frame in enumerate(raw_Dark_Frames):
    cpl.core.Msg.info(self.name, f"Processing {frame.file!r}...")
    if idx == 0:
        exp_time_list = cpl.core.PropertyList.load_regexp(frame.file,
            0, "EXPTIME", False)
        exp_time = exp_time_list.dump(show=False)
        match_exp = float(re.search(pattern, exp_time).group(1)) # type
            : ignore
    cpl.core.Msg.debug(self.name, f"Loading dark image {idx}...")
    raw_dark_image = cpl.core.Image.load(frame.file)

    raw_dark_image.subtract(bias_image)

    processed_dark_images.insert(idx, raw_dark_image)

    combined_image = processed_dark_images.collapse_median_create()

combined_image.divide_scalar(match_exp) # type: ignore

product_properties = cpl.core.PropertyList()
product_properties.append(
    cpl.core.Property("ESO PRO CATG", cpl.core.Type.STRING, r"
        OBJECT_REDUCED")
)

cpl.core.Msg.info(self.name, f"Saving product file as {output_file!r}."
    )

cpl.dfs.save_image(
    frameset,
    self.parameters,
    frameset,
    combined_image,
            self.name,
            product_properties,
            f"demo/{self.version!r}",
            output_file,
        )

        product_frames.append(
            cpl.ui.Frame(
                file=output_file,
                tag="MASTER_DARK",
                group=cpl.ui.Frame.FrameGroup.CALIB,
            )
        )

        return product_frames
```

C Flat

```
import cpl.core
import cpl.ui
import cpl.dfs
import cpl.drs
import re

from typing import Any, Dict

class FlatProcess(cpl.ui.PyRecipe):
    _name = "flat_processor"
    _version = "0.1"
    _author = "Benjamin Eisele"
    _email = "benjamin.eisele0101@gmail.com"
    _copyright = "GPL-3.0-or-later"
    _synopsis = "Basic Flat processor"
    _description = "This recipe takes a number of chosen raw flat frames,
        subtracts the bias and dark to produce a master flat."

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.parameters = cpl.ui.ParameterList(
            (
                cpl.ui.ParameterEnum(
                    name = "mflat.stacking.method",
                    context = "mflat",
                    description = "Method used for averaging",
                    default = "mean",
                    alternatives = ("mean", "median"),
                ),
            )
        )

    def run(self, frameset:cpl.ui.FrameSet, settings: Dict[str, Any]) -> cpl.ui
        .FrameSet:
        for key, value in settings.items():
            try:
                self.parameters[key].value = value
            except KeyError:
                    cpl.core.Msg.warning(
                        self._name,
                        f"Settings includes {key}:{value} but {self} has no
                            parameter named {key}.",
            )

output_file = "MASTER_FLAT.fits"

raw_flat_frames = cpl.ui.FrameSet()
bias_frame = None
dark_frame = None
product_frames = cpl.ui.FrameSet()

pattern = r'value\s+:\s+(\d+)'

for frame in frameset:
    if frame.tag == "CHOSEN_FLAT":
        cpl.core.Msg.debug(self.name, f"Got raw flat frame: {frame.file
            }.")
        frame.group = cpl.ui.Frame.FrameGroup.RAW
        raw_flat_frames.append(frame)
    elif frame.tag == "MASTER_BIAS":
        cpl.core.Msg.debug(self.name, f"Got master bias frame: {frame.
            file}.")
        frame.group = cpl.ui.Frame.FrameGroup.CALIB
        bias_frame = frame
    elif frame.tag == "MASTER_DARK":
        cpl.core.Msg.debug(self.name, f"Got master dark frame: {frame.
            file}.")
        frame.group = cpl.ui.Frame.FrameGroup.CALIB
        dark_frame = frame
    else:
        cpl.core.Msg.warning(
            component=self.name,
            message=f"Got frame {frame.file!r} with unexpected tag {
                frame.tag!r}, ignoring."
        )

if len(raw_flat_frames) == 0:
    cpl.core.Msg.error(
        self.name,
        f"No raw frames in frameset."
    )

processed_flat_images = cpl.core.ImageList()

cpl.core.Msg.warning(
    self.name,
    f"Loading calib files."
)

if bias_frame:
    bias_image = cpl.core.Image.load(bias_frame.file)

if dark_frame:
    dark_image = cpl.core.Image.load(dark_frame.file)

for idx, frame in enumerate(raw_flat_frames):
    if idx == 0:
        exp_time_list = cpl.core.PropertyList.load_regexp(frame.file,
            0, "EXPTIME", False)
        exp_time = exp_time_list.dump(show=True)
        match_exp = float(re.search(pattern, exp_time).group(1)) # type
            : ignore
        dark_image.multiply_scalar(match_exp)
```

```
            raw_flat_image = cpl.core.Image.load(frame.file)

            raw_flat_image.subtract(bias_image)
            raw_flat_image.subtract(dark_image)
            median = raw_flat_image.get_median()
            raw_flat_image.divide_scalar(median)
            processed_flat_images.insert(idx, raw_flat_image)

        combined_image = None

        method = self.parameters["mflat.stacking.method"].value

        cpl.core.Msg.info(self.name, f"Combining dark images using method {
            method!r}")

        if method == "mean":
            combined_image = processed_flat_images.collapse_create()
        elif method == "median":
            combined_image = processed_flat_images.collapse_median_create()

        product_properties = cpl.core.PropertyList()
        product_properties.append(
            cpl.core.Property("ESO PRO CATG", cpl.core.Type.STRING, r"
                OBJECT_REDUCED")
        )

        cpl.core.Msg.info(self.name, f"Saving product file as {output_file!r}."
            )

        cpl.dfs.save_image(
            frameset,
            self.parameters,
            frameset,
            combined_image,
            self.name,
            product_properties,
            f"demo/{self.version!r}",
            output_file,
        )

        product_frames.append(
            cpl.ui.Frame(
                file=output_file,
                tag="MASTER_FLAT",
                group=cpl.ui.Frame.FrameGroup.CALIB,
            )
        )

        return product_frames
```

D Science

```
import cpl.core
import cpl.ui
import cpl.dfs
import cpl.drs
import re

from typing import Any, Dict
from numpy import mat

class ScienceProcess(cpl.ui.PyRecipe):
    _name = "science_processor"
    _version = "0.1"
    _author = "EDPS User"
    _email = "test@testmail.com"
    _copyright = "GPL-3.0-or-later"
    _synopsis = "Basic Science processor"
    _description = "This recipe takes a number of raw science frames and
        applies basic calibration frames."

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.parameters = cpl.ui.ParameterList(
            (
            )
        )

    def run(self, frameset: cpl.ui.FrameSet, settings: Dict[str, Any]) -> cpl.
        ui.FrameSet:
        for key, value in settings.items():
            try:
                self.parameters[key].value = value
            except KeyError:
                cpl.core.Msg.warning(
                    self._name,
                    f"Settings includes {key}:{value} but {self} has no
                        parameter named {key}.",
                )

        output_file = "SCIENCE_FRAME.fits"

        pattern_strg = r"value\s+:\s+'(\w+)'"
        pattern_fil = r"value\s+:\s+'(\w+\s\w)'"
        pattern_doub = r"value\s+:\s+(\d+)"

        raw_science_frames = cpl.ui.FrameSet()
        bias_frame = None
        dark_frame = None
        flat_frame = None
        object_images = cpl.core.ImageList()
        object_products = cpl.ui.FrameSet()

        for frame in frameset:
            if frame.tag == "SCIENCE":
                frame.group = cpl.ui.Frame.FrameGroup.RAW
                cpl.core.Msg.debug(self.name, f"Got raw science frame: {frame.
                    file}.")
                raw_science_frames.append(frame)
            elif frame.tag == "MASTER_BIAS":
                frame.group = cpl.ui.Frame.FrameGroup.CALIB
                cpl.core.Msg.debug(self.name, f"Got master bias frame: {frame.
                    file}.")
                bias_frame = frame
            elif frame.tag == "MASTER_DARK":
                frame.group = cpl.ui.Frame.FrameGroup.CALIB
                cpl.core.Msg.debug(self.name, f"Got master dark frame: {frame.
                    file}.")
                dark_frame = frame
            elif frame.tag == "MASTER_FLAT":
                frame.group = cpl.ui.Frame.FrameGroup.CALIB
                cpl.core.Msg.debug(self.name, f"Got master flat frame: {frame.
                    file}.")
                flat_frame = frame
            else:
                cpl.core.Msg.warning(
                    component=self.name,
                    message=f"Got frame {frame.file!r} with unexpected tag {
                        frame.tag!r}, ignoring."
                )

        if len(raw_science_frames) == 0:
            cpl.core.Msg.error(
                self.name,
                f"No raw frames in frameset."
            )

        cpl.core.Msg.warning(
            self.name,
            f"Loading calib files."
        )

        if bias_frame:
            bias_image = cpl.core.Image.load(bias_frame.file)
        if dark_frame:
            dark_image = cpl.core.Image.load(dark_frame.file)
        if flat_frame:
            flat_image = cpl.core.Image.load(flat_frame.file)


        for idx, frame in enumerate(raw_science_frames):
            cpl.core.Msg.info(self.name, f"Processing {frame.file!r}...")
            if idx == 0:
                exp_time_list = cpl.core.PropertyList.load_regexp(frame.file,
                    0, "EXPTIME", False)
                exp_time = cpl.core.PropertyList.dump(exp_time_list, show=False
                    )
                match_exp = float(re.search(pattern_doub, exp_time).group(1)) #
                     type: ignore
                dark_image.multiply_scalar(match_exp) # type: ignore

            obj_typ_list = cpl.core.PropertyList.load_regexp(frame.file, 0, "
                OBJTYP", False)
            obj_typ = obj_typ_list.dump(show=False)
            match_obj = re.search(pattern_strg, obj_typ).group(1) # type:
                ignore

            cpl.core.Msg.debug(self.name, f"Loading science image {idx}...")
            raw_science_image = cpl.core.Image.load(frame.file)
            cpl.core.Msg.debug(self.name, f"Processing image {idx}...")
            raw_science_image.subtract(bias_image)
            raw_science_image.subtract(dark_image)
            raw_science_image.divide(flat_image)
            object_images.append(raw_science_image)
            filter_typ_list = cpl.core.PropertyList.load_regexp(frame.file, 0,
                "FILTER", False)
            filter_typ = filter_typ_list.dump(show=False)
            match_filter = re.search(pattern_fil, filter_typ).group(1) # type:
                ignore

        product_properties = cpl.core.PropertyList()
        product_properties.append(
            cpl.core.Property("ESO PRO CATG", cpl.core.Type.STRING, r"
                OBJECT_REDUCED")
        )
        product_properties.append(
            cpl.core.Property("EXPTIME", cpl.core.Type.FLOAT, match_exp)
        )
        product_properties.append(
            cpl.core.Property("OBJTYP", match_obj)
        )
        product_properties.append(
            cpl.core.Property("FILTER", match_filter)
        )

        combined_object_image = object_images.collapse_median_create()

        cpl.core.Msg.info(self.name, f"Saving product file as {output_file!r}."
            )

        cpl.dfs.save_image(
            frameset,
            self.parameters,
            frameset,
            combined_object_image,
            self.name,
            product_properties,
            f"demo/{self.version!r}",
            output_file,
        )

        object_products.append(
            cpl.ui.Frame(
                file=output_file,
                tag="SCIENCE_FRAME",
                group=cpl.ui.Frame.FrameGroup.RAW,
            )
        )

        return object_products
```