iris$_pipelineDocumentation$

Release 0.2.0

Andrea Zonca, Arun Surya

Jul 16, 2020

CONTENTS

The IRIS Data Reduction System is based on the `stpipe` package released by Space Telescope for the James Webb Space Telescope.

With `stpipe` we can configure each step of a pipeline through one or more text based .INI style files, then we provide one input FITS file or a set of multiple inputs defined in JSON (named Associations). Custom analysis steps and pipelines for IRIS are defined as classes in the current repository `iris_pipeline`

Then execute the pipeline from the command line using the `tmtrun` executable or using directly the Python library.

The pipeline also dynamically interfaces to the `CRDS` the Calibration References Data System, to retrieve the best calibration datasets given the metadata in the headers of the input FITS files. The `CRDS` client can also load data from a local cache, so for now we do not have a actual `CRDS` server and we only rely on a local cache.

The `CRDS` is not under our control, the Thirty Meter Telescope will deliver a database system to replace the `CRDS` and we can adapt our code to that in the future.

Part I

Getting Started

CHAPTER

ONE

GETTING STARTED

1.1 Requirements

First we need to install the requirements of the jwst package, see the JWST instructions, reported here for convenience:

```
conda create -n jwst_dev python=3.6 astropy
source activate jwst_dev
```

then we need to install the jwst package, currently iris_pipeline is being tested with jwst 0.13.7:

```
pip install https://github.com/spacetelescope/jwst/archive/0.13.7.zip
```

Then you need to download the CRDS cache:

```
git clone https://github.com/oirlab/tmt-crds-cache $HOME/crds_cache
```

the CRDS cache contains metadata for IRIS, the calibration files, flat fields, and a set of rules on how to choose the right calibration file given a set of metadata, you can browse the content on Github.

Finally, we need a custom version of the CRDS library that contains some modules specific to TMT:

```
git clone https://github.com/oirlab/tmt-crds.git
cd tmt-crds
pip install .
```

1.2 Development install

First fork the repository under your account on Github, then clone your fork on your machine.

Then enter the root folder and create a development install with:

```
pip install -e .
```

Part II

Example run

CHAPTER

TWO

EXAMPLE PIPELINE EXECUTION

Here is an example of what it takes to configure and run a pipeline with flat-fielding and background subtraction,

2.1 Setup CRDS

Make sure you have a local checkout of the CRDS cache as explained in the Getting started page. Run the `setup_local_crds.sh` to setup the enviroment variables needed to point the `crds` software to the CRDS cache. Optionally source this in your shell configuration to automatically set this up.

2.2 Get input simulations

Download simulated input FITS files for the IRIS imager from Figshare. It contains a raw science frame, a raw flat frame and a raw background frame.

2.3 Preprocess the flat frame

First we need to remove the dark frame from the flat frame and normalize it. A dark frame is already available in the CRDS and the pipeline knows how to retrieve it based on the metadata in the FITS file headers.

We can check in the package `documentation` what are the available pipelines and check the configuration options of the `pipeline.PreprocessFlatfield` class.

We do not need to customize it so we can directly call it from `tmtrun` and pass the input FITS file:

```
tmtrun iris_pipeline.pipeline.PreprocessFlatfield raw_flat_frame_cal.fits
```

This will pickup the relevant dark frame from the CRDS and process the file:

```
2019-10-04 17:59:48,057 - stpipe.PreprocessFlatfield - INFO - PreprocessFlatfield instance created.
2019-10-04 17:59:48,059 - stpipe.PreprocessFlatfield.dark_current - INFO - DarkCurrentStep instance␣
↪created.
2019-10-04 17:59:48,060 - stpipe.PreprocessFlatfield.normalize - INFO - NormalizeStep instance created.
2019-10-04 17:59:48,099 - stpipe.PreprocessFlatfield - INFO - Step PreprocessFlatfield running with␣
↪args ('raw_flat_frame_cal.fits',).
2019-10-04 17:59:48,554 - stpipe.PreprocessFlatfield - INFO - Prefetching reference files for dataset:
↪'raw_flat_frame_cal.fits' reftypes = ['dark']
2019-10-04 17:59:49,306 - stpipe.PreprocessFlatfield - INFO - Prefetch for DARK reference file is '/
↪home/azonca/crds_cache/references/tmt/iris/tmt_iris_dark_0001.fits'.
2019-10-04 17:59:49,307 - stpipe.PreprocessFlatfield - INFO - Starting preprocess flatfield ...
```

(continues on next page)

(continued from previous page)

```
2019-10-04 17:59:53,490 - stpipe.PreprocessFlatfield - INFO - Processing product raw_flat_frame
2019-10-04 17:59:53,490 - stpipe.PreprocessFlatfield - INFO - Working on input raw_flat_frame_cal.fits .
↪..
2019-10-04 17:59:53,641 - stpipe.PreprocessFlatfield.dark_current - INFO - Step dark_current running␣
↪with args (<IRISImageModel(4096, 4096) from raw_flat_frame_cal.fits>,).
2019-10-04 17:59:53,658 - stpipe.PreprocessFlatfield.dark_current - INFO - Using DARK reference file /
↪home/azonca/crds_cache/references/tmt/iris/tmt_iris_dark_0001.fits
2019-10-04 17:59:54,058 - stpipe.PreprocessFlatfield.dark_current - INFO - Step dark_current done
2019-10-04 17:59:54,101 - stpipe.PreprocessFlatfield.normalize - INFO - Step normalize running with␣
↪args (<IRISImageModel(4096, 4096) from raw_flat_frame_cal.fits>,).
2019-10-04 17:59:54,472 - stpipe.PreprocessFlatfield.normalize - INFO - Step normalize done
2019-10-04 17:59:54,472 - stpipe.PreprocessFlatfield - INFO - Finished processing product raw_flat_frame
2019-10-04 17:59:54,473 - stpipe.PreprocessFlatfield - INFO - ... ending preprocess flatfield
2019-10-04 17:59:54,811 - stpipe.PreprocessFlatfield - INFO - Saved model in raw_flat_frame_flat.fits
2019-10-04 17:59:54,811 - stpipe.PreprocessFlatfield - INFO - Step PreprocessFlatfield done
```

We have an output file `raw_flat_frame_flat.fits` and we can rename it:

```
mv raw_flat_frame_flat.fits flat.fits
```

2.4 Configure the image processing pipeline

The `Image2Pipeline` can be configured using a INI-style configuration file, `image2_iris.cfg`:

```
name = "Image2Pipeline"
class = "iris_pipeline.pipeline.Image2Pipeline"
save_results = True
    [steps]
      [[bkg_subtract]]
      [[assign_wcs]]
        skip = True
      [[flat_field]]
        config_file = flat_field.cfg
      [[photom]]
        skip = True
      [[resample]]
        skip = True
```

first we specify that we want to execute the pipeline defined in `iris_pipeline.pipeline.Image2Pipeline`, then we can configure each of the steps, for example skip some of them. Also we can import the configuration of a step from another file, in this case `flat_field.cfg`:

```
name = "flat_field"
class = "jwst.flatfield.FlatFieldStep"
# Optional filename suffix for output flats (only for MOS data).
flat_suffix = None
override_flat = 'flat.fits'
```

If we do not define `override_flat`, the pipeline will look up a suitable flat from the CRDS, in this case instead we specify a local `flat.fits` file.

2.5 Define the input data

JWST created a specification for defining how input files should be used by a pipeline, it is a JSON file named an association, see the JWST documentation.

In our example we need to specify a input raw science frame ad a background to be subtracted, see `asn_subtract_bg_flat.json`:

```
{
    "asn_rule": "Asn_Lv2Image",
    "asn_pool": "pool",
    "asn_type": "image2",
    "products": [
        {
            "name": "test_iris_subtract_bg_flat",
            "members": [
                {
                    "expname": "raw_science_frame_sci.fits",
                    "exptype": "science"
                },
                {
                    "expname": "raw_background_frame_cal.fits",
                    "exptype": "background"
                }
            ]
        }
    ]
}
```

2.6 Execute the pipeline from the command line

We can use `tmtrun` from a terminal to execute the pipeline:

```
tmtrun image2_iris.cfg asn_subtract_bg_flat.json
```

here is the output log:

```
2019-10-04 18:13:46,453 - stpipe.Image2Pipeline - INFO - Image2Pipeline instance created.
2019-10-04 18:13:46,454 - stpipe.Image2Pipeline.bkg_subtract - INFO - BackgroundStep instance created.
2019-10-04 18:13:46,456 - stpipe.Image2Pipeline.assign_wcs - INFO - AssignWcsStep instance created.
2019-10-04 18:13:46,458 - stpipe.Image2Pipeline.dark_current - INFO - DarkCurrentStep instance created.
2019-10-04 18:13:46,460 - stpipe.Image2Pipeline.flat_field - INFO - FlatFieldStep instance created.
2019-10-04 18:13:46,461 - stpipe.Image2Pipeline.photom - INFO - PhotomStep instance created.
2019-10-04 18:13:46,463 - stpipe.Image2Pipeline.resample - INFO - ResampleStep instance created.
2019-10-04 18:13:46,500 - stpipe.Image2Pipeline - INFO - Step Image2Pipeline running with args ('asn_
↪subtract_bg_flat.json',).
2019-10-04 18:13:47,130 - stpipe.Image2Pipeline - INFO - Prefetching reference files for dataset: 'raw_
↪science_frame_sci.fits' reftypes = ['dark']
2019-10-04 18:13:47,645 - stpipe.Image2Pipeline - INFO - Prefetch for DARK reference file is '/home/
↪azonca/crds_cache/references/tmt/iris/tmt_iris_dark_0001.fits'.
2019-10-04 18:13:47,645 - stpipe.Image2Pipeline - INFO - Override for FLAT reference file is '/home/
↪azonca/p/software/iris_pipeline/iris_pipeline/tests/data/flat.fits'.
2019-10-04 18:13:47,645 - stpipe.Image2Pipeline - INFO - Prefetching reference files for dataset: 'raw_
↪background_frame_cal.fits' reftypes = ['dark']
2019-10-04 18:13:47,651 - stpipe.Image2Pipeline - INFO - Prefetch for DARK reference file is '/home/
↪azonca/crds_cache/references/tmt/iris/tmt_iris_dark_0001.fits'.
```

(continues on next page)

(continued from previous page)

```
2019-10-04 18:13:47,651 - stpipe.Image2Pipeline - INFO - Override for FLAT reference file is '/home/
↪azonca/p/software/iris_pipeline/iris_pipeline/tests/data/flat.fits'.
2019-10-04 18:13:47,651 - stpipe.Image2Pipeline - INFO - Starting calwebb_image2 ...
2019-10-04 18:13:47,659 - stpipe.Image2Pipeline - INFO - Processing product test_iris_subtract_bg_flat
2019-10-04 18:13:47,659 - stpipe.Image2Pipeline - INFO - Working on input raw_science_frame_sci.fits ...
2019-10-04 18:13:47,918 - stpipe.Image2Pipeline.bkg_subtract - INFO - Step bkg_subtract running with␣
↪args (<IRISImageModel(4096, 4096) from raw_science_frame_sci.fits>, ['raw_background_frame_cal.fits
↪']).
2019-10-04 18:13:53,796 - stpipe.Image2Pipeline.bkg_subtract - INFO - Step bkg_subtract done
2019-10-04 18:13:53,854 - stpipe.Image2Pipeline.assign_wcs - INFO - Step assign_wcs running with args (
↪<IRISImageModel(4096, 4096) from raw_science_frame_sci.fits>,).
2019-10-04 18:13:53,855 - stpipe.Image2Pipeline.assign_wcs - INFO - Step skipped.
2019-10-04 18:13:53,856 - stpipe.Image2Pipeline.assign_wcs - INFO - Step assign_wcs done
2019-10-04 18:13:53,898 - stpipe.Image2Pipeline.dark_current - INFO - Step dark_current running with␣
↪args (<IRISImageModel(4096, 4096) from raw_science_frame_sci.fits>,).
2019-10-04 18:13:53,945 - stpipe.Image2Pipeline.dark_current - INFO - Using DARK reference file /home/
↪azonca/crds_cache/references/tmt/iris/tmt_iris_dark_0001.fits
2019-10-04 18:13:54,503 - stpipe.Image2Pipeline.dark_current - INFO - Step dark_current done
2019-10-04 18:13:54,566 - stpipe.Image2Pipeline.flat_field - INFO - Step flat_field running with args (
↪<IRISImageModel(4096, 4096) from raw_science_frame_sci.fits>,).
2019-10-04 18:13:55,328 - stpipe.Image2Pipeline.flat_field - INFO - Step flat_field done
2019-10-04 18:13:55,369 - stpipe.Image2Pipeline.photom - INFO - Step photom running with args (
↪<IRISImageModel(4096, 4096) from raw_science_frame_sci.fits>,).
2019-10-04 18:13:55,369 - stpipe.Image2Pipeline.photom - INFO - Step skipped.
2019-10-04 18:13:55,370 - stpipe.Image2Pipeline.photom - INFO - Step photom done
2019-10-04 18:13:55,370 - stpipe.Image2Pipeline - INFO - Finished processing product test_iris_subtract_
↪bg_flat
2019-10-04 18:13:55,370 - stpipe.Image2Pipeline - INFO - ... ending calwebb_image2
2019-10-04 18:13:55,606 - stpipe.Image2Pipeline - INFO - Saved model in test_iris_subtract_bg_flat_cal.
↪fits
2019-10-04 18:13:55,606 - stpipe.Image2Pipeline - INFO - Step Image2Pipeline done
```

After completion, the reduced science frame `test_iris_subtract_bg_flat_cal.fits` is written to disk, it includes all the metadata it had initially and additional details about the processing steps that were executed.

Part III

Design

CHAPTER

THREE

IRIS DATA REDUCTION SYSTEM DESIGN

CHAPTER

FOUR

PURPOSE

The IRIS Data Reduction System is planned to perform:

- real-time (< 1 minute) and offline data processing of IRIS images and spectroscopic data with the `iris_pipeline` Python package based on JWST's pipeline package `stpipe`, see the documentation
- raw readout processing from the IRIS imager and spectrograph into raw science quality frames with the C library `iris_readout` at https://github.com/oirlab/iris_readout, which will be used directly during real-time operations and will be wrapped into Python modules in `iris_pipeline` for offline processing.
- visualization of raw and reduced data to facilitate data assessment and analysis for real-time and offline use. These tools will be developed later and will possibly be based on existing community software tools like DS9 or cubeviz.

CHAPTER

FIVE

SOFTWARE INFRASTRUCTURE

We rely on the excellent work mostly by Space Telescope to grow the Python in Astronomy ecosystem around the `astropy` package. They also developed a suite of open-source tools to operate JWST based on their experience operating the Hubble Space telescope.

The `jwst` Python package bundles several tools:

- a `jwst.datamodel` package to handle custom schemas for complex hierarchical metadata
- a `stpipe` package to configure and execute processing pipelines
- a large array of data processing modules to analyze data from all instruments on board of JWST

We leverage this effort by:

- building a custom schema for IRIS
- using `stpipe` to execute our pipelines
- starting from JWST processing modules and customizing them for IRIS and publishing them on the `iris_pipeline` repository https://github.com/oirlab/iris_pipeline.

CHAPTER

SIX

FILE FORMAT

All data will be stored in FITS file format, following as closest as possible the conventions by JWST, see https://jwst-docs.stsci.edu/display/JDAT/Working+with+FITS+Files.

CHAPTER

SEVEN

EXAMPLE RUN

The best way to understand how `iris_pipeline` works is to checkout an example reduction of a raw science frame to a reduced science frame with flat-fielding and background subtraction.

CHAPTER

EIGHT

ACCESS CALIBRATION FILES VIA THE CALIBRATION REFERENCE DATA SYSTEM (CRDS)

See the section about Calibration

METADATA

iris_pipeline requires a set of metadata from TMT and from other subsystems to process the data, see the list of required metadata.

Moreover, iris_pipeline will add to the header of processed FITS files categorizing the data in:

OB-STYPE	OBSNAME	Description
Calibration (CAL)	IMG1-NFF, SLI-NFF LEN-SPX IMG1-DRK, SLI-DRK, LEN-DRK IMG1-TEL, SLI-TEL, LEN-TEL	Flat field Lenslet Spectral Extraction Master dark Telluric Star
Engineering (ENG)	SLI-IDP, LEN-IDP	Instrumental dispersion
Science (SCI)	IMG1-SCI, LEN-SCI, SLI-SCI IMG1-SKY, LEN-SKY, SLI-SKY	Science Sky

Part IV

Calibration

CHAPTER

TEN

CALIBRATION

10.1 Calibration files

Auxiliary data used in DRP algorithms are called calibration data. This includes both on-sky data (that is not of the astronomical target itself), daytime calibration frames, and other sub-component metadata. Metadata is non-image information that will typically come from the header of raw FITS files, or from IRIS, and/or the adaptive optics system via the observatory telemetry service. The NFIRAOS Science Calibration Unit (NSCU) will include a calibration system that will facilitate the taking of daytime calibration frames, such as arc lamp spectra, white light flat field images, and pinhole grids for measuring distortion. The following table summarizes the required calibration files necessary for the Data Reduction Software.

Notes about the table: Note: * = SPEC only, PTG = pointing, D-Map = Distortion Map, Env = Environmental, DTC = Daytime calibration, NTC = Nightime calibration.

Table 1: Calibration frames

Name	Reference Type	Source	Algorithms
Atm. Dispersion Residual	Metadata	IRIS ADC	Atmospheric Correction
Arc lamp spectra*	CAL (2D)	IRIS DTC (NSCU)	Wavelength solution
Bad pixel map	CAL (2D)	IRIS DTC	Correction of detector artifacts
Dark Frame	CAL (2D)	IRIS DTC and NTC	Dark subtraction
Env metadata	Metadata	ESW, FITS header	All
Fiber image	CAL (2D, 3D)	IRIS DTC (NSCU)	PSF Calibration
Flux calibration star	CAL (2D, 3D)	IRIS On-sky	Extract Star, Remove Absorption Lines
Instrument config	Metadata	ESW, FITS header	All
Lenslet scan*	Rect Matrix CAL (2D)	IRIS DTC (NSCU)	Spectral Extraction
NFIRAOS config	Metadata	ESW, FITS header	All
Pinhole Grid (D-Map)	CAL (2D)	IRIS DTC (NSCU)	Field distortion correction
PSF metadata	Metadata	ESW, FITS header	PSF calibration
PSF star	CAL (2D, 3D)	IRIS on-sky	PSF calibration
Sky frame	CAL (2D, 3D)	IRIS on-sky	Sky-subtraction
Telescope config PTG	Metadata	ESW, FITS header	All

Table 2: Real time Calibration frames

Name	Reference Type	Source	Algorithms
Atm. Dispersion Residual	Metadata	IRIS ADC	Atmospheric Correction
Arc lamp spectra*	CAL (2D)	IRIS DTC (NSCU)	Wavelength solution
Bad pixel map	CAL (2D)	IRIS DTC	Correction of detector artifacts
Dark Frame	CAL (2D)	IRIS DTC and NTC	Dark subtraction
Env metadata	Metadata	ESW, FITS header	All
Instrument config	Metadata	ESW, FITS header	All
NFIRAOS config	Metadata	ESW, FITS header	All
Sky frame	CAL (2D, 3D)	IRIS on-sky	Sky-subtraction
Telescope config PTG	Metadata	ESW, FITS header	All

10.2 Access calibration files via the Calibration Reference Data System (CRDS)

The Calibration Reference Data System (CRDS) is a set of tools developed by Space Telescope to organize and retrieve calibration reference files, e.g. flat frames, dark frames, for JWST and HST. When `stpipe` is executing a pipeline, it can automatically connect to the JWST CRDS server and get the right flat based on the metadata in the header of the data FITS files. The logic necessary to choose the right file is encoded in text files. Those configuration files and the actual calibration FITS files are also cached locally so that the CRDS client library works even without any connection to a central server.

We have created a CRDS cache folder in the Github repository https://github.com/oirlab/tmt-crds-cache, this includes in the `mappings/tmt` folder the metadata for IRIS and the rules to choose the right flat-field frame, for now there is only a dummy rule but this can be easily customize querying the metadata in the input file.

Currently we do not have any CRDS server running, but the users can download the CRDS cache locally and use it anyway, see the Getting started documentation.

Also, the CRDS client library needs to have minimal knowledge about metadata for TMT, therefore we maintain a fork of that library which simply adds a submodule dedicated to IRIS, https://github.com/oirlab/tmt-crds, it is quite easy to upgrade this to newer releases of CRDS by Space Telescope.

If TMT decided to use CRDS as their Data Management System, it would leverage the extensive set of tools and documentation available and would not require modifications to `stpipe`; otherwise, we will implement support for the DMS API into (our own fork of) `stpipe`.

Part V

Algorithms

CHAPTER

ELEVEN

ALGORITHMS

Algorithms to be implemented for the IRIS imager and Integral Field Spectrograph. Once the actual classes are implemented in `iris_pipeline`, we will just link their implementations.

11.1 Generate master dark

The master dark is generated by the median of 5-10 individual dark frames taken at the same exposure. This removes any of the frame-to-frame noise variation. Individual dark frames are zero illumination calibration frames with a shutter or blocking filter to not allow any light to hit the detector.

```
# 4096x4096 at 100 seconds exposures, N-frames
# dark1, dark2, dark3, . . . , darkN
darks = np.array([dark1, dark2, dark3, darkN])
master_dark = np.median(darks, axis=0)
```

11.2 Dark subtraction

Each science and calibration frame will have dark current, residual electric current that is flowing through the array when there is no light. Dark current increases with time, so all science and calibration frames need equivalent exposure dark frames. To remove the dark current, we subtract a single dark in the real-time or a master dark in the final pipeline, at the same exposure time as the science and calibration frames.

```
dark_subtracted_science = science - dark
dark_subtracted_flat = flat - dark
```

11.3 Remove detector artifacts

The detector will have two types of artifacts; permanent/semi-permanent and transient artifacts. Dead pixels, hot pixels or "frozen" pixel fall with the permanent/semi-permanent artifacts while cosmic rays (CR) are within the transient artifacts. Dead pixel are pixels that no longer function. Hot pixels are sensitive pixels that can have a non-linear response to incoming photons. They can also turn on and off and can even respond normally until a certain flux is achieved in which they become non-linear. "Frozen" pixels are pixels that have a low response rate (opposite of the hot pixels), with similar types of problems. CRs are high energy photons from the sky that can hit the detector randomly and leave bright artifacts.

For all types of detector artifacts, they are generally difficult to remove or remove completely with flat-fielding. To deal with detector artifacts, they need to be either masked out or subtracted off. For permanent artifacts, a bad pixel mask is used to mask the frame. Bad pixel masks need to be throughout the lifetime of the detector. Over time more pixels may become dead, hot or frozen. In some cases, hot or frozen pixels might be able to be recovered, depending on their severity. Hot, dead or frozen pixels, can be found by taking various length N number of dark exposures and median combining them, if features are 10-20 sigma above and below the noise level, they will be clipped added to the masked. Since the pixel-to-pixel response will change, we may apply a set of flat-fielding before clipping. CRs on the other hand need to be removed from the frame. There are several methods for removing CRs such as L.A. Cosmic (Dokkem et al. 2001) which takes the Laplacian to find artifacts with steep slopes in individual frames. In the most severe cases it is possible find cosmic rays by taking a median of several science and calibration frames. In order to properly mitigate CRs in the median case, one needs 3 or more frames in multiples of odd numbers (i.e. 3, 5, 7…).

- See details about creating and using a bad pixel map in *the documentation about data quality initialization*.
- See an example notebook on how to inizialize the bad pixel mask.

11.3.1 Data Quality (DQ) Initialization

Description

The Data Quality (DQ) initialization step in the calibration pipeline populates the DQ mask for the input dataset. Flags from the appropriate static mask reference file in CRDS are copied into the `PIXELDQ` array of the input dataset, because it is assumed that flags in the mask reference file pertain to problem conditions that are group- and integration-independent.

We use the same flagging convention used for JWST, see their documentation.

A bad pixel mask is a `datamodels.TMTMaskModel` object with a dq extension with size `(4096x4096)` of time `uint32`.

It can be created with:

```
from jwst.datamodels import TMTMaskModel
f = TMTMaskModel()
```

First we need to setup metadata:

```
f.meta.name = "IRIS"
f.meta.detector = "IRIS1"
```

Then we can create the 2D array and set some flag value:

```
f.dq = np.zeros((4096,4096))
f.dq[np.random.randint(0, 4096, size=(10,2))] = 1024 # dead pixel
f.dq[np.random.randint(0, 4096, size=(10,2))] = 2048 # hot pixel
```

check the content of the flag:

```
np.histogram(f.dq, bins=3)
(array([16777196,       10,       10]),
 array([   0.        ,  682.66666667, 1365.33333333, 2048.        ]))
```

And finally write to the `CRDS` cache:

```
f.write(Path.home() / "crds_cache/references/tmt/iris/tmt_iris_mask_0001.fits")
```

Which flag is picked up by the pipeline is determined by the `tmt_iris_mask_0001.rmap` file, see the current file content on Github.

The actual process consists of the following steps:

- Determine what MASK reference file to use via the interface to the bestref utility in CRDS.
- If the `PIXELDQ` or `GROUPDQ` arrays of the input dataset do not already exist, which is sometimes the case for raw input products, create these arrays in the input data model and initialize them to zero. The `PIXELDQ` array will be 2D, with the same number of rows and columns as the input science data. The `GROUPDQ` array will be 4D with the same dimensions (nints, ngroups, nrows, ncols) as the input science data array.
- Check to see if the input science data is in subarray mode. If so, extract a matching subarray from the full-frame MASK reference file.
- Copy the DQ flags from the reference file mask to the science data `PIXELDQ` array using numpy's `bitwise_or` function.

See an example notebook on how to inizialize the bad pixel mask.

Step Arguments

The Data Quality Initialization step has no step-specific arguments.

Reference Files

The `dq_init` step uses a MASK reference file.

iris_pipeline.dq_init Package

Classes

`DQInitStep`([name, parent, config_file, . . .])	Initialize the Data Quality extension from the mask reference file.

DQInitStep

class `iris_pipeline.dq_init.`**`DQInitStep`**(*name=None*, *parent=None*, *config_file=None*, *_validate_kwds=True*, ***kws*)

Bases: `jwst.stpipe.Step`

Initialize the Data Quality extension from the mask reference file.

The dq_init step initializes the pixeldq attribute of the input datamodel using the MASK reference file. For some FGS exp_types, initalize the dq attribute of the input model instead. The dq attribute of the MASK model is bitwise OR'd with the pixeldq (or dq) attribute of the input model.

Create a `Step` instance.

Parameters

name
[str, optional] The name of the Step instance. Used in logging messages and in cache filenames. If not provided, one will be generated based on the class name.

parent
[Step instance, optional] The parent step of this step. Used to determine a fully-qualified name for this step, and to determine the mode in which to run this step.

config_file
: [str path, optional] The path to the config file that this step was initialized with. Use to determine relative path names.

****kws**
: [dict] Additional parameters to set. These will be set as member variables on the new Step instance.

Attributes Summary

`reference_file_types`	

Methods Summary

`process`(input)	Perform the dq_init calibration step

Attributes Documentation

`reference_file_types = ['mask']`

Methods Documentation

`process`(*input*)
Perform the dq_init calibration step

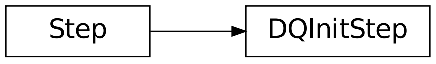
Parameters

input
[JWST datamodel] input jwst datamodel

Returns

output_model
[JWST datamodel] result JWST datamodel

Class Inheritance Diagram



11.4 Flat Fielding

The response of individual pixels may vary significantly pixel-to-pixel. In order to correct for this in the imager and the IFS slicer, it is necessary illuminate the detector with a uniform light source. This can be accomplished by an internal flat field, twilight flats or science fields with a sparse number of sources. Internal flat fields are usually provided by the calibration unit, in this case the NSCU. Twilight flats are flats in which the telescope points to an area of the sky, while tracking, and dithering around while taking exposures. Science fields with sparse number of sources can be also used for flat fields, as long as only few bright sources need to be masked out and the there is a large enough dither such that every pixel will be on blank sky at some point. The exposures for all cases need be before the detectors become non-linear but with enough signal to illuminate each pixel, this is typically 10000-20000 counts for a Hawaii-2RG. The flats are processed by the normalize flats algorithm . Science frames are then divided by the normalized flats. Sources detected in sky flats or science flats will be masked out.

11.5 Scaled sky-subtraction

Imager: Science images can as be used sky in sparse or medium density fields (density compared to the dither size). In high density fields or a source that fills the detectors, a separate sky will be needed. The sky background will be estimated by measuring the median of each detector. For the real-time pipeline the median of the sky background will be subtracted off. The F-DRP will include an advanced algorithm that iteratively determines the sky background and subtracts it off following the procedure outlined in Clement et al. 2012;

1. Subtract median from each image (detector)
    - Account for gain and sky, apply 5 sigma rejection (i.e. MAD)
    - Keep track of the values subtracted
2. For each individual median subtracted image:
    - Run SExtractor to detect all objects in frame and generate an object mask.
        - Save "-object" check image
3. Compute the median of each pixel using the mask computed previously but not including the pixel from the image you are working on
4. Add background value from (3) to the median computed in (1), this will be the new sky background
5. Subtract the new background from original image

The iterative process would include the mosaicking algorithm which would be the following

1. Stack final images
2. Redetect sources
3. Mask sources

This process would repeat until no more additional sources are detected for masking.

IFS: The sky-subtraction algorithm will scale the sky frame to match each of the individual science frames, utilizing the Davies et al. 2007 methodology. Various OH lines arise from families of vibrational transitions. While sky lines can vary randomly throughout the night, these families fluctuate together. Using brighter sky lines, comparing the science and sky data cubes it is possible to determine the ratio between OH lines for each transition family. These ratios can be applied to the sky data cube in order to minimize the residuals in the subtracted cube. The scaling ratios are applied to the entire sky data cube, rather than to an extracted spectrum, such that any spatial or wavelength variations in the sky lines across the cube will still be accurately matched and cancelled out in the sky subtraction.

11.6 Flux calibration

Imager: To convert from DN to flux (erg/s/cm^2/Hz) or AB magnitude a standard star needs to be observed in the same instrument configuration, airmass, and close in time as the science observations.

For science fields and filters that overlap with SDSS (citation), Pan-STARRS (citation) or UKIDDS (citation) will be able to use the stars within the science frame for as standards (2MASS may be too bright and low resolution to use). For science fields outside these surveys or for more precise photometry will require observing a standard star from a standard field. Apertures of increasing radii will be used to determine the curve of growth and the appropriate aperture to use with the PSF and seeing, maximizing S/N. Once an aperture size is determined, the flux is integrated the flux for a given band to produce the flux of the star in DN. Aperture corrections will be applied based PSF and the seeing. For relative photometry, $m_1 - m_2 = -2.5log_{10}\left(i\frac{f_1}{f_2}\right)$, where m1 and m2 are magnitudes of the sources and f1 and f1 are fluxes of the sources. This can be performed with a single source or the entire field with known sources to scale image. The zeropoints of the image can be determined from the known sources integrated flux and magnitude, (i.e. $m = -2.5log_{10}\left(\frac{DN}{exptime}\right) + zeropoint$). On sky tests will be required to determine the extinction corrected instrumental zeropoints. IFS: To convert from DN to flux units (erg/s/cm^2/Ang) a standard star needs to be observed in the same instrument configuration, airmass,and close in time as the science observations. In the near-IR the standard star at minimum needs to have zJHK photometry or ideally a spectrophotometric standard (in which a calibrated spectrum already exists). For a standard star with zJHK photometry, the photometry will be fit with a Planck law (or Rayleigh-Jeans approximation $1/\lambda^4$). Apertures of increasing radii will be used to determine the curve of growth and the appropriate aperture to use with the PSF and seeing. Once an aperture size is determined, the flux is integrated for a given wavelength to produce the spectrum of the standard star in DN. Aperture corrections based on the with growth curve and imager data. The science data cube and standard data cube are normalized by the exposure time such that they are each DN/s (count rate).

For the standard, we take the ratio of the flux (ergs/s/cm^2/Ang) over the count rate (DN/s). Each spaxel in the science data cube is multiplied by the ratio (flux/count rate) from the standard $F_{sci} = \frac{F_{std}}{R_{std}} * R_{sci}$, where F is flux (erg/s/cm^2) and R is count rate (DN/s)

11.7 Mosaic/Combine SCI

Imager: Mosaicking in the imager will be based on the dither pattern selected, and integer and non-integer pixel shifts will be supported. The dithers will be stored in the FITS header keywords and there will be support for an external file with the offsets. For integer pixel shits, frames will be combined using the median or mean, with sigma clipping to clip out deviant pixels. The clipping options will include using the standard deviation or median absolute deviation (MAD). For non-integer pixel shifts, there are widely used efficient software packages that handle drizzling and resampling, such as SWarp and DrizzlePac (previously known as AstroDrizzle).[j]

IFS: Mosaicking in the IFS will be relative to a source or the dither keywords in the FITS headers at a fixed PA. There will also be an option to stack the images based on an external offsets file. Currently, only integer pixel shifts will be supported. Frames will be combined using the median or mean, with sigma clipping to clip out deviant pixels. The clipping options will include using the standard deviation or median absolute deviation (MAD).

CHAPTER

TWELVE

IMAGER ALGORITHMS

12.1 Field distortion correction

The field distortion correction will correct the distortion in the imager field due to the optics of the system. These distortions can be chromatic and may need to be corrected per band. A calibration file with the distortion solution will be used using the distortion solution algorithm . The final image will need to be rectified and resampled based on the distortion solution. Software already exists to perform this task such as, SWarp, for rectifying and resampling the image based on the new distortion solution.

CHAPTER

THIRTEEN

IFS ALGORITHMS

13.1 Spectral extraction

Lenslet: Flux from an individual lenslet will be spread out into neighboring lenslets. Depending on the spacing between the lenslets, will determine how much flux falls into a neighboring lenslet. In order to recover the flux for an individual lenslet, it will be necessary to perform a deconvolution on the entire lenslet array, assigning flux to the appropriate lenslet. OSIRIS uses the Gauss-Seidel method to iteratively assign flux to individual spatial pixels (spaxels; Krabbe et al. 2004). The biggest assumption of the method is the knowledge of the PSF. In order to mitigate this problem, the PSF needs to be mapped in 2D and the structure of each lenslets PSF needs to be known precisely. Thus, the spectral extraction requires additional calibration files, rectification matrix (rectmat), which contains information about each lenslets PSF as a function of wavelength. Additional methods may be needed during INT. Slicer: Spectral extraction of the slicer will be similar to MOS (multi-object spectroscopy). The trace of each spectrum will be performed, typically fitting a low order polynomial. An aperture will be used over the spectrum, optimizing signal-to-noise (Horne 1986?). The extraction will be highly dependent on the extraction region and sky-subtraction algorithms .

13.2 Wavelength calibration

Wavelength calibration is performed using on arc lamps taken during daytime calibrations, typically Ar, Kr, and Xe. The arc lamps provide better velocity resolution and stability over OH skylines. A global wavelength solution is found for all of the spectra by fitting a low order polynomial. Legendre polynomials are preferred as they can be inverted (i.e. wavelength(pixel) $\rightarrow$ pixel(wavelength)) without significant errors in the coefficients. Using the global solution, a solution is found for each spaxel (spatial pixel). The solutions will be resampled to a common linear wavelength scale. These solutions are found be fairly stable in OSIRIS and we expect them to be similar. We anticipate checking the solution monthly for any changes. The solutions will be static based on the input lamp spectra and date they were taken.

13.3 Cube assembly

The spectral data cubes are assembled in this algorithm. The algorithm takes each extracted spectrum from spectral extraction routine and maps them to an x, y position on the sky (spatial rectification) based on the WCS information, and their z positions are shifted based on their individual wavelength solutions. The data cube format is (x, y, wavelength), which is common among data cubes with wavelength and frequency (i.e. VLT/SINFONI, ALMA and VLA).

13.4 Residual ADC

If necessary, implement residual ADC module (TBD). The ADC corrects the for the refraction caused by the atmosphere, at varying airmasses (or elevation). If the residuals from ADC correction are significant (like 4th order), it may be necessary to implement a module. To calibrate it, on-sky tests are required. One such test is to use a star to map the dispersion through the system at varying airmasses. Once the system is calibrated, temperature and pressure from the local weather, dome, telescope and instrument can be incorporated into the correction of the residuals per wavelength of light. With temperature/pressure lookup table, the DRS will have the correct spectral trace for the extraction. See instrument dispersion for how this is dealt with internally.

13.5 Telluric correction

Telluric absorption is caused by the Earths atmosphere, in which all spectra are attenuated by it. In order to correct for it, typically a featureless star is used to measure the attenuation carefully and apply a correction to the science spectra. Telluric correction as outlined by Vacca et al. 2003:

1. Normalization of the observed A-type main sequence star spectrum (e.g. O, B, and A should be fine with "featureless" spectra, as well as white dwarfs) in the vicinity of a suitable absorption feature (as defined below);
2. Determination of the radial velocity shift of the A-type star;
3. Shifting the Vega model spectrum to the radial velocity of the A-type star;
4. Scaling and reddening the Vega model spectrum to match the observed magnitudes of the A-type star;
5. Construction of a convolution kernel from a small region around an absorption feature in the normalized observed A-type and model Vega spectra;
6. Convolution of the kernel with the shifted, scaled, and reddened model of Vega;
7. Scaling the equivalent widths of the various H lines to match those of the observed A-type star.

Finally, the convolved model is divided by the observed A-type spectrum and the resulting telluric correction spectrum is multiplied by the observed target spectrum.

CHAPTER

FOURTEEN

ADVANCED ALGORITHMS

14.1 Optimizing readouts

All of the algorithms used with ROP-DRS, including the various sampling techniques (i.e UTR, MCDS), will be available offline for an end user that wants extra control of optimizing the readouts of their science. For example, a user may want to include readouts with a specific seeing constraint (i.e. removing poor seeing frames).

14.2 PSF-reconstruction

Knowledge of the PSF is essential in the reduction of AO data. However, this is challenging because of changing conditions (seeing) and the rate at which they change as well as the structure of the PSF. In order to reconstruct the PSF for a given observation, a simulated PSF from the NFIRAOS PSF simulator will be used to do the deconvolution on the imager and IFS. Laurent Jolissaint et al. 2011 (AO4ELT 2011)

CHAPTER

FIFTEEN

CALIBRATION ALGORITHMS

15.1 Rectmats

IFS lenslets: Rectmat (or rectification matrix) is a calibration file used for the spectral extraction of the IFS lenslet data for a specific scale and filter. An individual rectmat contains information about each lenslets PSF as a function of wavelength. The rectmats are constructed from spectral white light scans, which scan each individual lenslet to determine their PSF and contribution to neighboring lensiets. This information can also be used to remove the variation from lenslet-to-lenslet, similar to a flat field.

15.2 Distortion solution

The distortion solution algorithm will determine the distortion of the image on the imager. It will be constructed by using a static uniform grid pinholes (pinhole mask) and on sky calibration using dense stellar field (i.e. globular cluster). The distortion solution will be determined by fitting some type of nth order 2D polynomial (surface) to the position of the pinholes. Software already exists to perform these tasks such as; (1) SExtractor, for detecting the sources and (2) SCAMP, for determining the distortion.

15.3 Super sky

Super sky frames are median combined sky frames. The purpose of combining them is increase the signal-to-noise of the sky. The super sky frames are used for scaled sky subtraction of the imager (in the case where the source fills the imager) and the IFS slicer.

15.4 Super dark

See generate master dark

15.5 Instrumental dispersion

The optics of IRIS (including from NFIRAOS) can cause spectral curvature, or instrumental chromatic dispersion. A white light fiber can be used to map the dispersion (x and y position of the spectra) in the system. In OSIRIS, most of the instrumental dispersion was caused by the dichroic used in the AO system.

15.6 Normalize flats

Imager: The normalize flats algorithm takes the imaging flats and generates a normalized flat (values 1 or close to one) which are used to correct the pixel-to-pixel variation. To normalize the imaging flats, N number of flats are median combined, subtracted by a dark (real-time) or master dark (F-DRP) and then divided by either the median or mode (depending on the distribution of pixel values on the detector) of the combined flats.

```
normalize_flat = (np.median(flat) - dark)/np.median(flat)
```

IFS slicer: The normalize flats algorithm takes the spectral flats and generates a normalized flat (values 1 or close to one) which are used to correct the pixel-to-pixel variation. The spectral flats median combined and subtracted by a dark (real-time) or master dark (F-DRP). To normalize the spectral flats, the spectral response is fit with a polynomial and subtracted off each flat, and then divided by either their median or mode (depending on the distribution of pixel values on the detector).

Part VI

Reference/API

CHAPTER

SIXTEEN

IRIS_PIPELINE PACKAGE

16.1 Functions

monkeypatch_jwst_datamodels()	
test(**kwargs)	Run the tests for the package.

16.1.1 monkeypatch_jwst_datamodels

iris_pipeline.**monkeypatch_jwst_datamodels**()

16.1.2 test

iris_pipeline.**test**(***kwargs*)

Run the tests for the package.

This method builds arguments for and then calls `pytest.main`.

Parameters

package
[str, optional] The name of a specific package to test, e.g. 'io.fits' or 'utils'. Accepts comma separated string to specify multiple packages. If nothing is specified all default tests are run.

args
[str, optional] Additional arguments to be passed to `pytest.main` in the `args` keyword argument.

docs_path
[str, optional] The path to the documentation .rst files.

open_files
[bool, optional] Fail when any tests leave files open. Off by default, because this adds extra run time to the test suite. Requires the `psutil` package.

parallel
[int or 'auto', optional] When provided, run the tests in parallel on the specified number of CPUs. If parallel is `'auto'`, it will use the all the cores on the machine. Requires the `pytest-xdist` plugin.

pastebin
[('failed', 'all', None), optional] Convenience option for turning on py.test pastebin output. Set to 'failed' to upload info for failed tests, or 'all' to upload info for all tests.

pdb
[bool, optional] Turn on PDB post-mortem analysis for failing tests. Same as specifying `--pdb` in `args`.

pep8
[bool, optional] Turn on PEP8 checking via the pytest-pep8 plugin and disable normal tests. Same as specifying `--pep8 -k pep8` in `args`.

plugins
[list, optional] Plugins to be passed to `pytest.main` in the `plugins` keyword argument.

remote_data
[{'none', 'astropy', 'any'}, optional] Controls whether to run tests marked with @pytest.mark.remote_data. This can be set to run no tests with remote data (`none`), only ones that use data from http://data.astropy.org (`astropy`), or all tests that use remote data (`any`). The default is `none`.

repeat
[`int`, optional] If set, specifies how many times each test should be run. This is useful for diagnosing sporadic failures.

skip_docs
[`bool`, optional] When `True`, skips running the doctests in the .rst files.

test_path
[str, optional] Specify location to test by path. May be a single file or directory. Must be specified absolutely or relative to the calling directory.

verbose
[bool, optional] Convenience option to turn on verbose output from py.test. Passing True is the same as specifying `-v` in `args`.

16.2 Classes

`BackgroundStep`([name, parent, config_file, ...])	BackgroundStep: Subtract background exposures from target exposures.
`DQInitStep`([name, parent, config_file, ...])	Initialize the Data Quality extension from the mask reference file.
`FlatFieldStep`([name, parent, config_file, ...])	Flat-field a science image using a flatfield reference image.
`Image2Pipeline`(*args, **kwargs)	Image2Pipeline: Processes JWST imaging-mode slope data from Level-2a to Level-2b.
`NormalizeStep`([name, parent, config_file, ...])	DarkCurrentStep: Performs dark current correction by subtracting dark current reference data from the input science data model.
`PreprocessFlatfield`(*args, **kwargs)	PreprocessFlatfield: Remove dark and normalize exposure to create a flat field to be later added to the CRDS.
`SubtractImagesStep`([name, parent, ...])	SubtractImagesStep: Subtract two exposures from one another to accomplish background subtraction.
`UnsupportedPythonError`	

16.2.1 BackgroundStep

class iris_pipeline.**BackgroundStep**(*name=None*, *parent=None*, *config_file=None*, *_validate_kwds=True*, ***kws*)

Bases: `jwst.stpipe.Step`

BackgroundStep: Subtract background exposures from target exposures.

Create a `Step` instance.

Parameters

name
[str, optional] The name of the Step instance. Used in logging messages and in cache filenames. If not provided, one will be generated based on the class name.

parent
[Step instance, optional] The parent step of this step. Used to determine a fully-qualified name for this step, and to determine the mode in which to run this step.

config_file
[str path, optional] The path to the config file that this step was initialized with. Use to determine relative path names.

****kws**
[dict] Additional parameters to set. These will be set as member variables on the new Step instance.

Attributes Summary

`spec`

Methods Summary

`process`(input, bkg_list)	Subtract the background signal from target exposures by subtracting designated background images from them.

Attributes Documentation

spec = '\n sigma = float(default=3.0) # Clipping threshold\n maxiters = integer(default=None) # Number

Methods Documentation

`process`(*input*, *bkg_list*)

Subtract the background signal from target exposures by subtracting designated background images from them.

Parameters

input: JWST data model
input target data model to which background subtraction is applied

bkg_list: filename list
list of background exposure file names

Returns

result: JWST data model
the background-subtracted target data model

16.2.2 DQInitStep

class `iris_pipeline.`**`DQInitStep`**(*name=None*, *parent=None*, *config_file=None*, *_validate_kwds=True*, ***kws*)

Bases: `jwst.stpipe.Step`

Initialize the Data Quality extension from the mask reference file.

The dq_init step initializes the pixeldq attribute of the input datamodel using the MASK reference file. For some FGS exp_types, initalize the dq attribute of the input model instead. The dq attribute of the MASK model is bitwise OR'd with the pixeldq (or dq) attribute of the input model.

Create a `Step` instance.

Parameters

name
[str, optional] The name of the Step instance. Used in logging messages and in cache filenames. If not provided, one will be generated based on the class name.

parent
[Step instance, optional] The parent step of this step. Used to determine a fully-qualified name for this step, and to determine the mode in which to run this step.

config_file
[str path, optional] The path to the config file that this step was initialized with. Use to determine relative path names.

****kws**
[dict] Additional parameters to set. These will be set as member variables on the new Step instance.

Attributes Summary

`reference_file_types`	

Methods Summary

`process`(input)	Perform the dq_init calibration step

Attributes Documentation

`reference_file_types = ['mask']`

Methods Documentation

`process`(*input*)

Perform the dq_init calibration step

Parameters

input
[JWST datamodel] input jwst datamodel

Returns

output_model
[JWST datamodel] result JWST datamodel

16.2.3 FlatFieldStep

`class iris_pipeline.`**`FlatFieldStep`**(*name=None*, *parent=None*, *config_file=None*, *_validate_kwds=True*, ***kws*)

Bases: `jwst.stpipe.Step`

Flat-field a science image using a flatfield reference image.

Create a `Step` instance.

Parameters

name
[str, optional] The name of the Step instance. Used in logging messages and in cache filenames. If not provided, one will be generated based on the class name.

parent
[Step instance, optional] The parent step of this step. Used to determine a fully-qualified name for this step, and to determine the mode in which to run this step.

config_file
[str path, optional] The path to the config file that this step was initialized with. Use to determine relative path names.

****kws**
[dict] Additional parameters to set. These will be set as member variables on the new Step instance.

Attributes Summary

`reference_file_types`	
`spec`	

Methods Summary

`process`(input)	This is where real work happens.
`skip_step`(input_model)	Set the calibration switch to SKIPPED.

Attributes Documentation

`reference_file_types = ['flat']`

`spec = '\n # Suffix for optional output file for interpolated flat fields.\n # Note that this is only us`

Methods Documentation

`process`(*input*)
This is where real work happens. Every Step subclass has to override this method. The default behaviour is to raise a NotImplementedError exception.

`skip_step`(*input_model*)
Set the calibration switch to SKIPPED.

This method makes a copy of input_model, sets the calibration switch for the flat_field step to SKIPPED in the copy, closes input_model, and returns the copy.

16.2.4 Image2Pipeline

class `iris_pipeline.`**`Image2Pipeline`**(**args*, ***kwargs*)
Bases: `jwst.stpipe.Pipeline`

Image2Pipeline: Processes JWST imaging-mode slope data from Level-2a to Level-2b.

Included steps are: background_subtraction, assign_wcs, flat_field, photom and resample.

See `Step.__init__` for the parameters.

Attributes Summary

image_exptypes	
spec	
step_defs	

Methods Summary

process(input)	This is where real work happens.
process_exposure_product(exp_product[, . . .])	Process an exposure found in the association product

Attributes Documentation

image_exptypes = ['MIR_IMAGE', 'NRC_IMAGE', 'NIS_IMAGE']

spec = '\n save_bsub = boolean(default=False) # Save background-subracted science\n '

step_defs = {'assign_wcs': <class 'jwst.assign_wcs.assign_wcs_step.AssignWcsStep'>, 'bkg_subtract': <cl

Methods Documentation

process(*input*)
This is where real work happens. Every Step subclass has to override this method. The default behaviour is to raise a NotImplementedError exception.

process_exposure_product(*exp_product*, *pool_name=' '*, *asn_file=' '*)
Process an exposure found in the association product

Parameters

exp_product: dict
A Level2b association product.

pool_name: str
The pool file name. Used for recording purposes only.

asn_file: str
The name of the association file. Used for recording purposes only.

16.2.5 NormalizeStep

class iris_pipeline.**NormalizeStep**(*name=None*, *parent=None*, *config_file=None*, *_validate_kwds=True*, ***kws*)

Bases: `jwst.stpipe.Step`

DarkCurrentStep: Performs dark current correction by subtracting dark current reference data from the input science data model.

Create a `Step` instance.

Parameters

name
[str, optional] The name of the Step instance. Used in logging messages and in cache filenames. If not provided, one will be generated based on the class name.

parent
[Step instance, optional] The parent step of this step. Used to determine a fully-qualified name for this step, and to determine the mode in which to run this step.

config_file
[str path, optional] The path to the config file that this step was initialized with. Use to determine relative path names.

****kws**
[dict] Additional parameters to set. These will be set as member variables on the new Step instance.

Attributes Summary

`spec`

Methods Summary

`process`(input)	This is where real work happens.

Attributes Documentation

spec = '\n dark_output = output_file(default = None) # Dark model subtracted\n '

Methods Documentation

process(*input*)

This is where real work happens. Every Step subclass has to override this method. The default behaviour is to raise a NotImplementedError exception.

16.2.6 PreprocessFlatfield

class iris_pipeline.**PreprocessFlatfield**(**args*, ***kwargs*)

Bases: `jwst.stpipe.Pipeline`

PreprocessFlatfield: Remove dark and normalize exposure to create a flat field to be later added to the CRDS.

Included steps are: dark_current, normalize

See `Step.__init__` for the parameters.

Attributes Summary

`step_defs`

Methods Summary

`process`(input)	This is where real work happens.
`process_exposure_product`(exp_product[, ...])	Process an exposure found in the association product

Attributes Documentation

```
step_defs = {'dark_current': <class 'iris_pipeline.dark_current.dark_current_step.DarkCurrentStep'>, 'n
```

Methods Documentation

process(*input*)

This is where real work happens. Every Step subclass has to override this method. The default behaviour is to raise a NotImplementedError exception.

process_exposure_product(*exp_product*, *pool_name=' '*, *asn_file=' '*)

Process an exposure found in the association product

Parameters

exp_product: dict
A Level2b association product.

pool_name: str
The pool file name. Used for recording purposes only.

asn_file: str
The name of the association file. Used for recording purposes only.

16.2.7 SubtractImagesStep

class iris_pipeline.**SubtractImagesStep**(*name=None*, *parent=None*, *config_file=None*, *_validate_kwds=True*, ***kws*)

Bases: `jwst.stpipe.Step`

SubtractImagesStep: Subtract two exposures from one another to accomplish background subtraction.

Create a `Step` instance.

Parameters

name
[str, optional] The name of the Step instance. Used in logging messages and in cache filenames. If not provided, one will be generated based on the class name.

parent
[Step instance, optional] The parent step of this step. Used to determine a fully-qualified name for this step, and to determine the mode in which to run this step.

config_file
[str path, optional] The path to the config file that this step was initialized with. Use to determine relative path names.

****kws**
[dict] Additional parameters to set. These will be set as member variables on the new Step instance.

Attributes Summary

`spec`

Methods Summary

`process`(input1, input2)	Subtract the background signal from a JWST data model by subtracting a background image from it.

Attributes Documentation

spec = '\n '

Methods Documentation

process(*input1*, *input2*)
Subtract the background signal from a JWST data model by subtracting a background image from it.

Parameters

input1: JWST data model
input science data model to be background-subtracted

input2: JWST data model
background data model

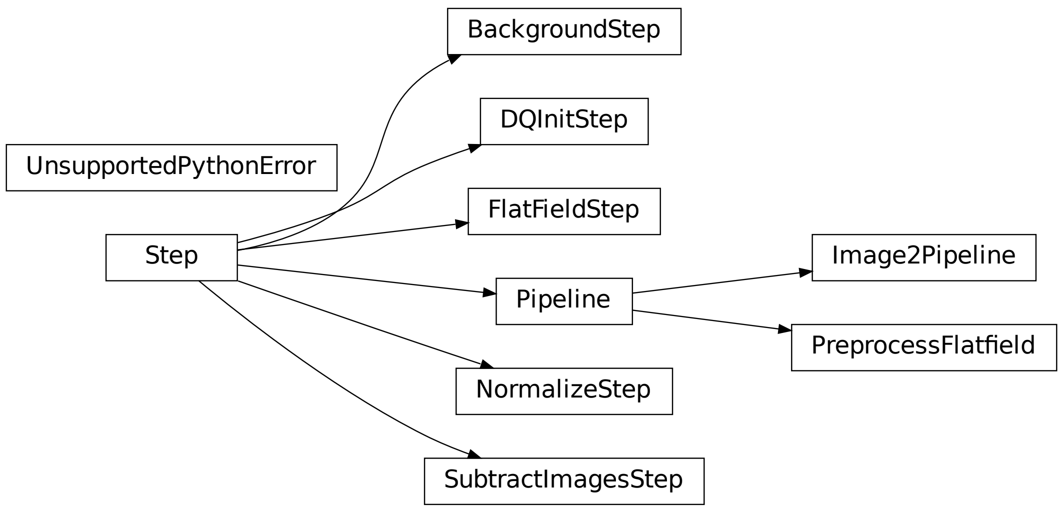
Returns

result: JWST data model
background-subtracted science data model

16.2.8 UnsupportedPythonError

`exception iris_pipeline.UnsupportedPythonError`

16.3 Class Inheritance Diagram



PYTHON MODULE INDEX

INDEX