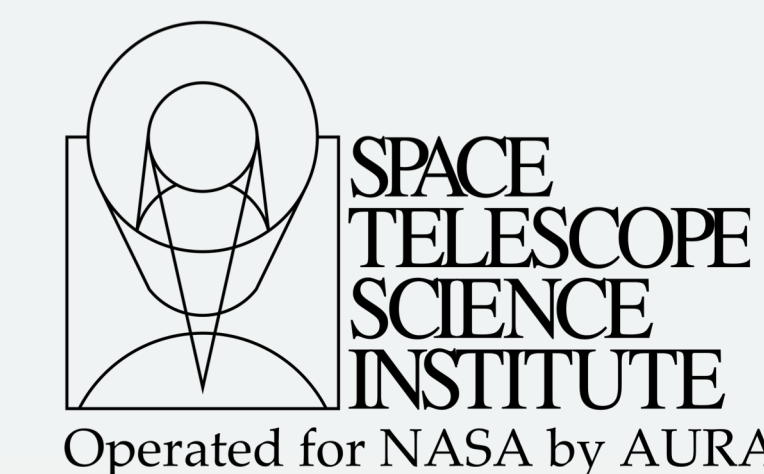




# The JWST Data Calibration Pipeline

Howard Bushouse, Jonathan Eisenhamer, James Davies (STScI)



## Overview

The STScI pipeline for the James Webb Space Telescope (JWST) data uses a Python environment called **stpipe**.

- Common framework and code base for all 5 JWST science instruments
- **stpipe** provides common configuration handling, parameter validation and persistence, and I/O management for all steps
- Provides interface to the Calibration Reference Data System (CRDS) for retrieving reference files (darks, flats, ...)
- Steps written as Python classes
  - Invoked from within Python or from **stpipe** command line
  - Sets of step classes can be configured into a pipeline
  - **stpipe** handles flow of data between steps
- Environment includes the use of standard data models
  - Data models defined in yaml, using json schema
  - Validates format of incoming data files and output models
  - Presents an abstract interface to isolate steps from details of file storage

## Very Simple Step Example

```
class FlatFieldStep(Step):
    reference_file_types = ['flat']
    def process(self, input):
        input_model = datamodels.ImageModel(input)
        self.flat_name = self.get_reference_file(input_model, 'flat')
        flat_model = datamodels.FlatModel(self.flat_name)
        result = flat_field.do_correction(input_model, flat_model)
        return result
    def do_correction(input, flat):
        input.data /= flat.data
        input.err /= flat.data
        input.dq = np.bitwise_or(input.dq, flat.dq)
        return input
```

Declare type(s) of reference files needed by step

Load input into data model

Retrieve name of flat; Load it into a data model

Apply correction

Apply flat to science data and error array

Propagate flat's data quality flags into science exposure flag array

## Very Simple Pipeline Example

```
class SimplePipeline(Pipeline):
    step_defs = {"bias": bias_step.SuperBiasStep,
                "dark": dark_step.DarkCurrentStep,
                "reset": reset_step.ResetStep}
    def process(self, input):
        input = self.bias(input)
        input = self.dark(input)
        if input.meta.instrument.name == "MIRI":
            input = self.reset(input)
        return input
```

Define steps to be used

Apply steps to input data model

Return modified data model, to be handled by stpipe

## Running a Step

From the command line:

```
> strun jwst.flat_field.FlatFieldStep myimage.fits
```

or

```
> strun flat_field.cfg myimage.fits
```

From Python:

```
>>> from jwst.flat_field import FlatFieldStep
>>> result = FlatFieldStep.call('myimage.fits',
                                config_file='flat_field.cfg')
```

Stpipe creates output file with result

Result returned as new data model

Configuration (.cfg) files use ini-file format. **stpipe** uses ConfigObj library to parse them. Without a .cfg file, steps/pipelines run with default parameters.

## Data Models

- **stpipe** environment handles all reading/writing of data to/from disk files
  - Relieves developers from having to do format-specific I/O in steps
- Data models defined for all types/modes of JWST data products (images, 1-D spectra, 2-D spectra, 3-D spectra, time-series, ...)
- Insulates steps, pipelines, and developers from vagaries of file formats
- Presents developers and users with a common data framework in all steps and pipelines
- Each model is a bundle of array or tabular data, plus meta data
  - Structure and contents defined using YAML and json schema
  - Schemas are modular – a given schema can include sub-schema
  - Schemas used to validate contents of model when serializing to/from a disk file

## Simple Data Model

```
allOf:
- $ref: core.schema.yaml
- type: object
  properties:
    data:
      title: The pixel data array
      fits_hdu: SCI
      ndim: 2
      datatype: float32
    err:
      title: Error array
      fits_hdu: ERR
      datatype: float32
    dq:
      title: Data quality array
      fits_hdu: DQ
      default: 0
      datatype: uint32
```

Include sub-schema that defines meta data

Define basic properties for each data attribute

"fits\_hdu" used for serializing to/from FITS files

## Meta Data Schema

```
type: object
properties:
  meta:
    telescope:
      title: Telescope used to acquire the data
      type: string
      fits_keyword: TELESCOP
    instrument:
      title: Instrument used to acquire the data
      type: string
      enum: [FGS, MIRI, NIRCAM, NIRISS, NIRSPEC]
      fits_keyword: INSTRUME
    zero_frame:
      title: Zero frame was downlinked separately
      type: Boolean
      fits_keyword: ZEROFRAM
```

Define basic properties for each meta attribute

enum list for defining allowed values

"fits\_keyword" used for serializing to/from FITS files