

# A Getting Started Guide for JWST NIRSpec IFU Data Reduction

Authorship included equal contribution from Rose Hewald, Nikolas Younker, Prasanna Adhikari, Lauren A. Elicker, and Emmy Bursk with minor contributions from Matthew B. Bayliss

This how-to guide consists of documentation written for researchers of any level of experience to be able to install and run the JWST pipeline and reduce NIRSpec IFU data. It follows the public [documentation](#) created by the TEMPLATES ERS team.

## Installing the JWST Pipeline

1. Make sure that either [anaconda](#) or [mini conda](#) (preferred) is installed (microsoft/windows users need to use Linux subsystem for this pipeline as certain packages are linux/MAC specific; instructions to linux subsystem download can be found [here](#); once downloaded follow conda download instructions for linux users)
  - Using the command `conda list` will show installed packages
2. Download appropriate .yaml file from the [stenv repository](#) depending on operating system and desired version of Python
  - The repository is regularly updated by STSci
  - ex. for Linux and Python 3.10, download `stenv-Linux-py3.10-2023.04.05.yaml`
3. In terminal, change directory to the location where the .yaml file is stored
4. Create a conda environment using the following command:

```
conda env create -n jdrp -f stenv-Linux-py3.10-2023.04.05.yaml
```

- This will create an environment named '**jdrp**' ('JWST data reduction pipeline'), but other names can be chosen instead
  - You may need to close the terminal and reopen it before proceeding
  - The .yaml file name in this line creating the conda environment needs to match the file name of the .yaml file that you downloaded
5. Activate the environment: `conda activate jdrp`
    - If you named your conda environment something else, you'll need to replace 'jdrp' with your environment name

```
(base) rhewald@Envy:~$ conda activate jdrp
(jdrp) rhewald@Envy:~$
```

6. Check that the version of jwst is 1.10.1 or later: `conda list jwst`
  - To update your pipeline version type the following into terminal (with the jdrp environment activated):  
`pip install jwst --upgrade`

```
(jdrp) rhewald@Envy:~$ conda list jwst
# packages in environment at /home/rhewald/miniconda3/envs/jdrp:
#
# Name                      Version                      Build      Channel
jwst                        1.10.2                      pypi_0     pypi
```

**NOTE:** If using Linux operating system, jwst module may fail to import correctly

ERROR: Could not build wheels for spherical-geometry, astroscrappy, drizzlepac, which is required for installation

In this case, upgrade pip, then use pip install for each of the individual packages that failed

```
pip install --upgrade pip
sudo apt-get install build-essential
pip install astropy
pip install spherical-geometry
pip install stsci.image
pip install stsci.imagestats
pip install stsci.stimage
pip install drizzlepac
pip install wheel
pip install jwst
```

7. Open a Jupyter notebook that will run data reduction steps within the conda environment with the following line: `jupyter notebook`

8. Download data one of two ways:

From the Mikulski Archive for Space Telescopes ([MAST](#))

- On the “Select a collection...” dropdown list, select “MAST Observations by Proposal ID”
- Type in the proposal ID for the [TEMPLATES](#) JWST program: **01355**
  - The data can be filtered further by instrument, project, target name, etc.
- Select needed files, add to “Download Basket”, and download
- Note that the estimated download time, for us, was in all instances much greater than the time that it actually took

\*\*\*Using the python script [FetchInBulk.py](#)

- Install astroquery using either pip or conda:

```
(jdrp) rhewald@Envy:~$ pip install --pre astroquery
```

```
(jdrp) rhewald@Envy:~$ conda install -c astropy astroquery
```

- Save the python script and run from the command line
  - The project ID, 01355, is the first argument
  - The instrument is the second: nirspec, nircam, miri, niriss, or nirspec

- The kind of data is the third: UNCAL, RATE, CAL, I2D; specifying no kind of data will download all types

```
(jdrp) rhewald@Envy:~$ python JWST_API_Fetch_inBulk_templates.py 01355 nirspec UNCAL
Querying for program 01355
Querying for science instrument nirspec
Found 147 matching Observations...
Fetching product list, 8 Observations at a time.
Number of unique files: 3632
```

- Run the following two commands:

```
(jdrp) rhewald@Envy:~$ chmod u+x mastDownload_20230623102644.sh
(jdrp) rhewald@Envy:~$ ./mastDownload_20230623102644.sh
```

\*\*\*Using this python script will download data for all the targets in the proposal, so the files will need to be separated based on target to proceed through the pipeline.

9. Download [outline for NIRSpec pipeline](#) for data reduction as python file
  - Access in Jupyter Notebook and edit file paths as needed

10. Download JWST NIRSpec Clean ([NSClean](#)) and all accompanying python files in the NSClean GitHub (util.py, \_\_init\_\_.py, config.py)

## Running the JWST Pipeline:

### NIRSpec data reduction jupyter notebook

#### STAGE 1

IMPORTANT NOTES: Before proceeding - make sure pipeline installed into computer with no errors (all packages installed, "jwst" package exists in the virtual environment) as neglect will lead to errors in ipynb script.

1. The first 2 cells of the notebook are a source citing cell and a quick introduction into the ipynb script. The 3rd cell will show specific targets, data for the pipeline must correspond to 1 or more of those 4 targets (shown below in red)

- a. Comment out the targets that the pipeline is not running on (by adding a # before it)

```
# select target name for easier processing below
# MAKE SURE TARGET NAME MATCHES DATAFILE HEADERS!
target = 'SGAS1723'
target = 'SGAS1226'
target = 'SPT0418-47'
target = 'SPT2147-50'
```

2.

This next cell requires directories to be set up correctly:

- a. The first part requires setting up the path to the directory to where all the pipeline configuration files are located (aka where the pipeline was downloaded into); for WSL (Linux) users it will look like `"/wsl.localhost/Ubuntu/home/nby16/"` where Ubuntu can be interchanged with other linux platforms and "nby16" will be substituted with user specific folder name:

```
# Modify the paths to the relevant directories on your machine
# -----
# 1) point to where the jwst pipeline config files are located
home = "/wsl.localhost/Ubuntu/home/nby16/" # for B. Welch
#home = "/Users/tahutch1/programs/jwst-drp/" # for T. Hutchison
```

Here is an example of this on a MacOS system:

```
# Modify the paths to the relevant directories on your machine
# -----
# 1) point to where the jwst pipeline config files are located
home = "/Users/laurenlicker/Downloads/TEMPLATES"
```

- b. In the next part, the path to the uncalibrated MAST data is input into the ipynb. NOTE: for all WSL users that keep files under their C drive, `"/mnt/c/"` needs to precede all path directories as linux will need to mount the c drive. Provided below will be the examples for both systems on setting up the path. The same method can be applied to all 4 targets.

```
if target == 'SGAS1723':
    input_path = '/mnt/c/Users/nikol/data/1723/1723/'
```

If using the pipeline for multiple targets, you'll need to have each target's data separated into their own directory, for example:

```
if target == 'SGAS1723':
    #input_path = '/Users/bdwlch1/Documents/data/templates/sdss1723/full_uncal_data/'
    ##input_path = '1723/NIRSpecIFU_All/JWST/'
    input_path = '../Volumes/JWST/NIRSpec_UNCAL/1723'
if target == 'SGAS1226':
    # input_path = '/Users/bdwlch1/Documents/data/templates/sdss1226/nirspec/MAST_2023-01-03T0916/JWST/'
    input_path = '../Volumes/JWST/NIRSpec_UNCAL/1226'
if target == 'SPT0418-47':
    # input_path = '/Users/bdwlch1/Documents/data/templates/spt0418/MAST_2022-10-10T1412/JWST/'
    input_path = '../Volumes/JWST/NIRSpec_UNCAL/0418'
if target == 'SPT2147-50':
    # input_path = '/Users/bdwlch1/Documents/data/templates/spt2147/MAST_2022-11-02T1127/JWST/'
    #input_path = "/Users/tahutch1/data/raw/jwst/ers/templates/MAST_2022-09-23T1410/JWST/" # for T. Hutchison
    input_path = '../Volumes/JWST/NIRSpec_UNCAL/2147'
```

- c. Next, input the path to where the outputs will be stored into the ipynb. Similar to the step above, the same method can be applied to all 4 targets. More specific folders within this directory path will be created later in the notebook.

```
# 3) point to where you want your processed outputs to live
# for B. Welch
if target == 'SGAS1723':
    output_path = '/mnt/c/Users/nikol/data/templates/output/'
```

- i. NOTE: If no output path exists, the ipynb script automatically creates an output path

```
if os.path.exists(output_path) == False: # if folder doesn't exist
    print('Creating folder ' + output_path)
    os.system('mkdir ' + output_path) # creates the folder
```

- ii. But, I have run into issues with this line not actually making the folder when it doesn't already exist, so it is best to manually create the general output folder before running this step.
- d. The next few lines will set up the CRDS cache as environmental variables in the script. Make sure that file paths are set up correctly here as well. The WSL example shows the base "home + "crds...." is commented out as WSL will have to mount the c drive if that is where the crds files are located.

```
#####
# Set up CRDS path and server environment variables
os.environ["CRDS_PATH"] = "/mnt/c/Users/nikol/crds_cache/"#home + "crds_cache/jwst_ops"
os.environ["CRDS_SERVER_URL"] = "https://jwst-crds.stsci.edu"
```

But the path should already be set up correctly for the MacOS system users.

```
# Set up CRDS path and server environment variables
os.environ["CRDS_PATH"] = home + "crds_cache/jwst_ops"
os.environ["CRDS_SERVER_URL"] = "https://jwst-crds.stsci.edu"
```

- e. Once all directories are set up correctly you can proceed. NOTE: misspellings in the input path may lead to files not being found with no error messages displaying to user and misspellings in the output path will lead to errors being displayed to user
- 3. The next 3 cells in the script import JWST packages, any packages not downloaded/downloaded with error will present a problem in these cells as the "import jwst" will result in error. Make sure that when the last of these 3 cells is run that a version of jwst prints to the user. If you have an import error, you may get a "jwst module not found" error message.
    - a. Common errors on import encountered: wheels being unable to be built
      - i. If the error message in the terminal when packages are being installed states an issue with pip, look to update pip and restart process (done by running "pip install --upgrade pip" in the terminal window)
      - ii. If the error message says "this error originates from a subprocess and is likely not an issue with pip," look to check and make sure the packages are compatible with system by looking at package documentation online (mainly an issue with windows computers) and that the appropriate wheel building tools are installed (i.e. Windows C/C++ VS Build Tools which can be found [here](#))
      - iii. Recommendation if neither of the above works is to manually debug each package as it can get tricky, websites like stackexchange, the package documentation website, other programming debug community boards, and ChatGPT are helpful tools
      - iv. If the cell runs through an example of the output is below:

```
import jwst
print(jwst.__version__)
```

1.10.2

4. The next cell runs to collect the data from the directories. In this step the number of “IFU” (aka science) and “Sky” or “IFU-Offset” (aka background) files for the selected target will print to the user. If either of these numbers are 0, the directories are not set up correctly and data is not being found or is not downloaded. The output given below is from an example of a successful run with 16 science uncal fits files and 16 background uncal fits files:

```
print(f'Number of IFU: {ifu}, Number of sky: {sky}')
```

Number of IFU: 16, Number of sky: 16

5. The next cell will create the necessary output folders for the rate files that will be created shortly using the uncalibrated files. If the output path is set up correctly, the folders will be created and placed in the output path. There is no need to create the folders in this cell beforehand as the ipynb is set up to make them. These folders are set up to have names of “L2a/bkg” and “L2a/science”, but these can be modified if you like.
6. The next cell will take some time to run (depending how many uncal files you are sending through the pipeline). This cell is actually “running the pipeline”, creating the rate files, and splitting up the sci and bkg files. As the comment cells in the pipeline suggest, the amount of cores being used to run this cell can be changed. Depending on how many are used will either speed up or slow down the amount of time required to run this cell. NOTE: for smaller computer systems this cell has been known to kill the jupyter notebook kernel. As of 6/12/2023 there has been no fix to finding the solution to not killing the kernel. As previously stated, this cell will take a few minutes to run, DO NOT proceed through the script until the asterisk located next to the cell identification, in blue brackets, is gone. Location shown below where asterisk will be in the

```
In [ ]: # R
        # L
```

script:

If the cell is finished, there should be a number in the brackets and there should be rate files in the “L2a/sci” and “L2a/bkg” files.

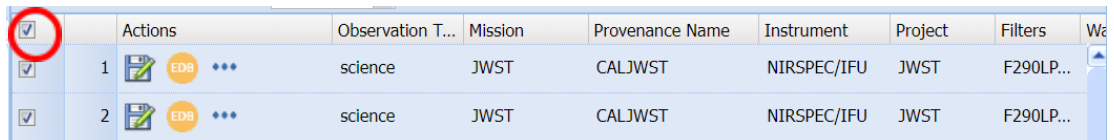
- a. We have found that running many files through the pipeline at once (>20 files) might crash a computer the pipeline is running on. To combat this problem, you can move some of the uncal folders out of the input folder/directory so that only a few are still in there and end up running through the pipeline at once. Once those few go through the pipeline and have rate files created, you can move those out of the folder and put other uncal ones in the folder, iterating this process until all the uncal files have been reprocessed to create a rate file.
7. The next section is **Correct 1/f noise**. The first cell in this section is similar to the cell in number 5. The cell will create and place folders in the output path where the user specified. As long as

the output path from the first code cell is correct and no errors arise, the user can run the cell and proceed onto the next.

8. For this next cell, both the rate files created in the last section and the cal files downloaded from MAST will be needed. These cal files are pre-existing stage 2 pipeline products.

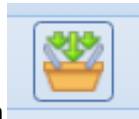
- a. To get the cal files off of MAST

- i. Select the appropriate method to search for the data under the “Select a Collection” in the top left (i.e. for TEMPLATES “MAST Observations by Proposal ID”)
    - ii. Directly next to the right under “and enter one or more Proposal IDs” enter in the key that is appropriate to the category selected and will give the wanted data
    - iii. From there filters on the left need to be selected, first under “Mission” select “JWST”. Then scroll down and under “Instrument” select “NIRSPEC/IFU”. Then scroll down more to “Target Name” and select the targets that you want to gather the calibration files for.
    - iv. Next is to select all the data that comes up in the middle section of the screen. This can be done by clicking on the box in the top left of the white header bar



	Actions	Observation T...	Mission	Provenance Name	Instrument	Project	Filters	We
<input checked="" type="checkbox"/>	1	...	science	JWST	CALJWST	NIRSPEC/IFU	JWST	F290LP...
<input checked="" type="checkbox"/>	2	...	science	JWST	CALJWST	NIRSPEC/IFU	JWST	F290LP...

- v. Then the data needs to be added to the download basket, this is done by clicking the basket icon with 3 green arrows above it which is located right above all of



the data

- vi. After that is selected, a pop up screen will come up. On the left one more filter needs to be selected. In the “Group” section scroll to the bottom where it says “CAL”. Select that box and then download the data by once again selecting all the boxes and hitting download which will be in the top right of the pop up. NOTE: the pop up might say it will take >100,000 minutes to download data (or some other large number); it will (should) not take that long. In our experience, it took a couple minutes to download all the data.

9. In the same cell as step 8, at the bottom make sure directories are correct to where the file locations for both rate files and calibration files are correct. Rate files will most likely be fine, as they are created and sorted by the script but double checking will never hurt. Calibration files need to be downloaded and separated by target to make sure the correct files get run through the pipeline for each target. The function in this cell will create masks for the data based on the calibration files and the rate files. Any error that may occur in this function is most likely a missing rate or cal file (ex: a rate or cal file not being downloaded correctly, so the cell stops running due to error). There most likely will not be the same number of cal files as there are rate files for each target; the code should work fine with this. An example of separating the directory



path is shown below:

```
if target == 'SGAS1723':
    ratefiles = sorted(glob.glob(output_path + folders_L2a[0] + '**/jw*_0[2,6]101*rate.fits'))
    caldir = '/mnt/c/Users/nikol/data/templates/output/L2a_destripe'
if target == 'SGAS1226':
    ratefiles = sorted(glob.glob(output_path + folders_L2a[0] + '**/jw*_02101*rate.fits'))
    caldir = '/Users/bdwelch1/Documents/data/templates/sdss1226/nirspec/pmap1046/L2b_destripe/'
if target == 'SPT0418-47':
    ratefiles = sorted(glob.glob(output_path + folders_L2a[0] + '**/jw*_02101*nrs1_rate.fits'))
    caldir = '/mnt/c/Users/nikol/data/templates/output/0418/L2a_destripe/0418_cal/0418_cal/'
if target == 'SPT2147-50':
    ratefiles = sorted(glob.glob(output_path + folders_L2a[0] + '**/jw*_02101*nrs1_rate.fits'))
    caldir = '/mnt/c/Users/nikol/data/templates/output/2147/L2a_destripe/2147_cal/2147_cal/'
```

The mask files should be located in the same folder as the rate files when this cell is done running.

10. Moving onto the next cell... As long as the above cell ran correctly, there should be little to change in this cell. The one thing to change will be at the bottom and appending the path to importing NSClean. **The NSClean and associated python scripts must be initially created as .py files.** They cannot be created as .ipynb files and then renamed as a .py file; this will cause errors! All NSClean python files need to be downloaded for this code to work (NSClean.py, util.py, config.py, \_\_init\_\_.py). You must add the directory path to the NSClean files to the line “sys.path.append(MAKE THIS YOUR DIRECTORY PATH TO NSCLEAN FILES)”. It also would not hurt to download the .py files into your jupyter notebook directory in case any errors arise, they can be easily called from there as well using correct notation. **The names of the python files and classes need to match those in the jupyter notebook script, including capitalization.**

For example both the NSClean python file and the class name within the python script need to be named lowercase “nsclean” for the line below:

```
from nsclean import nsclean as nc
```

Note that the first “nsclean” is the python file name and the second “nsclean” is the class name. Here is where the class name is located in the nsclean.py file. In this case, the line would need to be “from nsclean import NSClean as nc” to match the capitalization of the class name:

```
13 class NSClean:
14     """
15     NSClean is the base class for removing residual correlated
16     read noise from JWST NIRSpec images. It is intended for use
17     on Level 1 pipeline products, i.e. IRS$^2$ corrected slope
18     images. All processing is done in detector coordinates, with
19     arrays transposed and flipped so that the IRS$^2$ "zipper"
20     appears along the bottom. For most users, NSClean's `clean`
21     method automatically transposes and flips the data as
22     necessary.
23     """
24
25     # Class variables. These are the same for all instances.
26     ny = 2048 # Number of lines in detector space
27     nx = 2048 # Number of columns in detector space
28     sigrej = 3.0 # Standard deviation threshold for flagging
29                 # statistical outliers.
```



Also, note that we had issues when importing “as nc” (from the “from nsclean import nsclean as nc” line). We recommend removing the “as nc” part from the line.

- a. LINUX NOTE: Linux users need to /mnt/ before accessing any drive and folders that is not in their linux directory

11. The NSClean files need to be modified before being used to clean the rate files. Open all of the NSClean files (util, init, config) (using e.g. PyCharm or VSCode). At the top of the python files where all the modules are imported, all of the “.” need to be deleted before the files as they are not necessary and result in import errors. For example “.util” needs to be changed to just “util”:

```
from .util import make_lowpass_filter
```

Check the import statements in the util.py, config.py, and \_\_init.py\_\_ files and change them accordingly.

12. Going into the next cell a few things need to be changed in order to get the cell to run correctly. These next couple steps also apply to the cell below as they have the same function just with a difference between nrs1 and nrs2. At the top of the cell make sure the following 3 import statements are there: “from NSClean import NSClean”, “import config”, and “import util”.

```
M = fits.getdata(maskfiles[1])  
cleaner = NSClean('NRS1', M)
```

In this line on the left, change nc.NSClean to just NSClean as shown in the picture (with the capitalization matching your naming system).

```
H0['comment'] = 'Processed by NSClean Rev. '+config.__version__
```

In this line shown above, change “nc.\_\_version\_\_” to “config.\_\_version\_\_” as shown above in the screenshot.

The last thing to change should be changing “nc.chsuf” to “util.chsuf” as shown below.

```
output_path + folders_l2a_describe[1] + \  
    util.chsuf(os.path.basename(files[i]), '.cln_mask.fits'),  
    overwrite=True)
```

- a. If these same changes are made to the nrs2 cell, both NSClean cells should run through. You will know it is running through when outputs saying “starting file #” and “Done file #” appear under the cell. An example of this output is below:

```
starting file 0  
Done file 0  
starting file 1  
Done file 1  
starting file 2  
Done file 2  
starting file 3  
Done file 3  
starting file 4  
Done file 4  
starting file 5  
Done file 5  
starting file 6  
Done file 6  
starting file 7  
Done file 7
```

At the end of stage 1 the outputs from NSClean produces uncalibrated destripped files. Before NSClean, if these files are viewed in a software, there will be extraneous noise which is unneeded as well as a large stripe through the file. The calibrated files (cal.fits) that are downloaded off MAST will clean these files up so that they can be masked in Stage 2 of the processor. After it has undergone NSClean the files can be viewed to be less messy as well as the stripe being reduced in visualness. These files are placed in the “L2a\_destripe” folder and will be called “jw\*\*\*\*\*\_\*\*\*\*\*\_\*\*\*\*\*\_nrs\*\_rate.cln\_mask.fits”. To make sure that the correct amount are there, you can print out the length mask files to the user along the way and make sure none are lost in the process by “print(len(maskfiles))” or “print(len(masks))”.

## STAGE 2

Stage 2 is much easier than stage 1 in regards to running/changing any of the code. As stage 2 largely relies on what was done previously.

1. The first cell in stage 2 creates directory files for the calibrated exposures to go into. As long as the output path is unchanged from before there should be no errors from this cell and it will output that 3 folders have been created (if not done previously when ran through this section).
2. The second cell is a function that adds leakcal files which most TEMPLATES users will not use as provided in the description below the cell. If the user wants to use, follow steps located in the text below the code for this cell.
3. The next block of cells deals with groupings for the targets and SHOULD be left untouched unless errors arise. Once the appropriate target is reached, an output that looks like

```
2023-06-20 10:23:41,241 - stpipe - WARNING - /home/user/anaconda3/envs/jdrp/lib/python3.10/site-packages/jwst/associations/association.py:215: UserWarning: 'exptime' contains path, but should only be a filename. All input files should be in a single directory, so no path is needed.
  warnings.warn(err_str, UserWarning)

0418/rate/L2b_destripe/jw01355-o011_02101_spec2_01_nrs1_asn.json
0418/rate/L2b_destripe/jw01355-o011_02101_spec2_02_nrs1_asn.json
0418/rate/L2b_destripe/jw01355-o011_02101_spec2_03_nrs1_asn.json
0418/rate/L2b_destripe/jw01355-o011_02101_spec2_04_nrs1_asn.json
```

should pop up. Do not worry about the warning, the script will proceed with no issues.

4. The next cell takes forever to run, anticipate at least 45 min-1 hour for the cell to run. This cell is the pipeline with the asn files that were just created. Once this cell is complete, the user can move onto stage 3.

## STAGE 3

This part of the script will generate the data cube to be used for viewing.

1. The first cell is another one that creates files in your directory for you. If no errors arise proceed to the next cell.
2. The following cell is just to make sure that the folder is created and will print the amount of them created.
3. The next 3 cells should run through with little/no errors if everything above has worked. Any errors that come up in these cells are probably from appropriate files not being created from previous steps. Or from Python package import errors. At the end of these 3 cells, the asn.json file is created to be turned into a cube. The output from this stage will be a .json file under the L3\_destripe folder.



4. Before the next cell is ran, the cell with “# Next Step, call stage 3 pipeline on new asn file”, there need to be some changes in the directory and files need to be moved. If you open up the L3asn.json file in a text editor it will show a directory path to a file next to expname... when this file is called, it will be called starting from the output path which is not entirely correct. To fix this error with a little manual labor follow the path in your output path to the “L3\_destripe folder”. From here, look at the directory path in the “L3asn.json” file and create those same folders in sequence as shown. A walkthrough will be shown below to help users.
  - a. For example: output path = /home/user/Downloads/0418/rate/
  - b. L3asn.json file path = “expname”: “0418/rate/L2b\_destripe/sci/filename”
  - c. Go into output path and go into “L3\_destripe” folder
  - d. Create new folder in there >L3\_destripe> 0418
  - e. Create new folder so that >L3\_destripe>0418>rate
  - f. Create new folder so that >L3\_destripe>0418>rate>L2b\_destripe
  - g. Create new folder so that >L3\_destripe>0418>rate>L2b\_destripe>sci
  - h. Then under the sci folder COPY and PASTE the file names that are given in the L3asn.json file. The file names will end with something like ...nrs”\_cal2.fits. These cal2 fits files can be found under the output file path in the “L2b\_destripe>sci” folders. Simply copy the necessary files from the L2b path and paste them into the L3 path. This should solve the error that will come up in this cell.
    - i. This cell should take roughly ~5-20 minutes to run, wait for the asterisk next to the cell number to go away before proceeding.
  - i. After this cell, data cubes are created and can be viewed in QFits or any other data cube visualization software.
  - j. The following 2 cells are clean up cells which as of 6/27 have no intended use due to files that do not exist publicly. The data cubes are already created and can be visualized as well as further cleaned with different software.

Final two cells:

Retrieving background.txt files: ...

Rerun cell Stage 3 - Creating Final Data Cubes for each target

It starts with # *Make association file* )

```
In [52]: # Make association files
calfiles = glob.glob(output_path + folders_L2b[1] + '*cal2.fits')

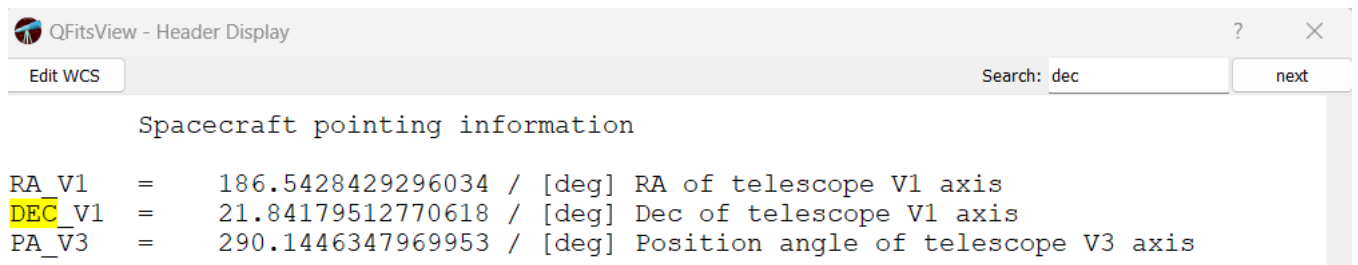
bkgfiles = glob.glob(output_path + folders_L2b[2] + '*x1d.fits')
#bkgfiles_cal = glob.glob(output_path + 'L2b/bkg/*cal2.fits') # for background testing only
```

**Getting the backgrounds:** utilizing the Python script [getting-backgrounds-jdrp.py](#)

1. Change the coordinates within the script to match those of the target that you are pointing to.

```
pointings = [{'ra':186.71373333333333,'dec':21.872213888888889,
              'name':'SGAS1226','date':'2022-12-31'},
              {'ra':260.89956541666666,'dec':34.196991666666666,
              'name':'SGAS1723','date':'2022-07-15'}]
```

2. This information can be found several ways, one of which is to open the header within the QFitsView and use the search feature.



3. Once the coordinates have been updated, switch between the listed targets by changing the target number (Note: [0] indicates the first entry, [1] indicates the second entry, and so on).

```
# choosing target
target = pointings[1]
```

## Galaxy Masking

\*\* Done in a different environment \*\*

Code found [here](#) on GitHub by Dr. Taylor Hutchinson

## Part I

1. Under the “First things first” section, update file names and paths as necessary

We're going to start by reading in the straight-from-the-pipeline cube.

```
In [2]: # reading in the galaxy cube
target = 'SGAS1723'

path = '/path/to/data/'
filename = 'filename.fits'

path = f'/Users/tahutch1/data/raw/jwst/ers/templates/reduced/{target}/' # TAH-defined
filename = 'Level3_SGAS1723_BGSUB_g140h-f100lp_s3d.fits' # TAH-defined
```

2. Scroll through slices of the data cube on QFitsView to select the central slice and number of slices on either side that you would like the signal-to-noise map to be created using.

```
In [3]: central_slice = 854 # roughly center of [OIII] emission line
lower_end = 5 # slices from center
upper_end = 5 # slices from center

# setting up the slice indexes
start = central_slice - lower_end
end = central_slice + upper_end + 1 # for indexing, have to add 1
```

3. The following cell calculates the error across the selected slices.
4. The next cell can be changed to alter the size, range, and color of the plot and will produce the first plot after running.
5. The next section focuses on removing unwanted pixels from the plot using Dr. Hutchinson's [pixel-patching](#) function.
  - a. Patches can be created as an ellipse, circle, ring & wedge, or rectangle. They are added one after the other within the cell that begins "*# making a copy of snr\_map to use in masking*". An example of adding each of the possible patches is shown below:

```
ellipse = Ellipse((16,16),
                  width = 7,
                  height = 20,
                  angle = 34,
                  alpha = 0.3,
                  facecolor = 'C2')

circle = Circle((18,20),
               radius = 5,
               alpha = 0.3,
               facecolor = 'C1')

wedge = Wedge((16,16),
              r = 8,
              theta1 = 0,
              theta2 = 270,
              width = 3,
              alpha = 0.3,
              facecolor = 'C3')

rectangle = Rectangle((8,13),
                     width = 16,
                     height = 5,
                     alpha = 0.3,
                     facecolor = 'C4')
```

- b. After running this cell, the previous plot with the addition of any patches should be produced.

6. The section titled “Making the new map” created the mask by assigning 1 to numerically significant spaxels and 0 to the rest (including those which were previously patched over)
  - a. To change individual spaxels between their ‘galaxy’ or ‘not galaxy’ assignment, use the following line of code:

```
In [11]: # new_map[y,x] = np.nan
```

- b. Insert the coordinates of the desired spaxel and change ‘np.nan’ to either 0 or 1.
- c. This can be repeated as many times as necessary.
- d. Note: these lines of code will need to be copied and pasted later in Part II

## Part II

1. The first two cells use the final masked galaxy from Part I to create a contour map of the signal-to-noise values to identify the bins for sigma clipping.
  - a. These cells will output the contour map.
  - b. The following cell will produce an array of the bin values.
  - c. The next will output plots of each of the different signal-to-noise bins as their own layer.
2. The two cells under the heading “Making S/N bins into mask layers” uses the previously output bin ranges to assemble a map of each of the layers and create the completed galaxy mask.
  - a. The individually edited spaxels from part one need to be pasted into the first of these cells:

```
# NOTE: IF YOU MANUALLY ADDED PIXELS OR REMOVED THEM  
# IN THE PREVIOUS PART (when making the full galaxy mask)  
# YOU'LL NEED TO PASTE THAT CODE HERE AS WELL  
  
# They'll be added to the lowest S/N level  
# (or, if you removed a pixel, they'll be removed here)
```

3. The last cell saves the galaxy mask as a .fits file for future use.