

# Changes in Version 2016

**Important Note:** The Ureka package has been deprecated as of April 26, 2016. As a result, the MOSFIRE pipeline has migrated to a version which is not dependent on IRAF/PyRAF, but only on python packages. It should work with any python install which provides the required packages and versions.

## New features

- DRP is no longer dependent on IRAF/PyRAF
  - The use of IRAF's `geoxytran`, `imcombine`, and `imarith` tasks have been replaced with python equivalents.
  - The DRP should now work with any python install which has the required python packages
- Improved slit tracing using a better thresholding algorithm
- An updated (and now web based) instruction manual
- The DRP now performs optimal spectral extraction Horne 1986 and outputs a 1D spectrum. Please note that this is intended as a quick look tool, not for final science use.
- The `handle` step now writes `filelist.txt` which contains a list of all the files processed by `handle` instead of printing that output to the screen. The file also contains messages for files not categorized for processing explaining why. In addition, `handle` now no longer writes list files with no content. This is intended to make it easier to quickly see what files are available for reduction.

## Improvements and bug fixes

- Changed dependence on `pylab` to `matplotlib.pyplot`
- Uses `astropy.io.fits` instead of `pyfits` when available
- Adjust log messages to send more to DEBUG instead of INFO. Leads to less clutter in messages visible to user.

## Changes in Version 2015A

### New features

- Reduction of long2pos and long2pos\_specphot
- Reduction of longslit data
- Automatic generation of driver file
- Logging and diagnostic information on screen and on disk
- Package-style installation as a Ureka sub-package
- Support for Ureka 1.5.1

### Improvements and bug fixes

- Fix incorrect determination of the slit parameters which prevented the use of large slits
- Fix incorrect determination of the average wavelength dispersion for the long2pos mode
- Added ability of specifying the output name of the files
- Improved robustness of non-interactive wavelength solution, added possibility of switching from interactive to non-interactive during the reduction, added k-sigma clipping of the sky or arc lines
- Fixed the problem with the interactive wavelength window not closing at the end of the fit
- Fixed the problem with the interactive fitting window showing up empty on the first fit (no need to use the x key to unzoom)
- Added procedure to fix the header of old images acquired with an outdated version of long2pos
- Disabled cosmic ray rejection for the case of less than 5 exposures
- There is no need to specify one of the observations in Rectify: Rectify will use the first of the files listed in the Offset files.

This manual describes the installation and usage of the MOSFIRE data reduction pipeline on a unix-like computer. Although primarily tested and developed on a Mac, the pipeline operates on both OSX and Linux systems. In the section 4, we describe an installation procedure for a Mac OSX system. Later sections describe code usage, execution, and outputs.

The MOSFIRE spectrograph data reduction pipeline was architected by the MOSFIRE commissioning team and written by Nick Konidaris with extensive checking and feedback from Chuck Steidel and other MOSFIRE team members. The pipeline is maintained on an online code repository <https://github.com/Keck-DataReductionPipelines/MosfireDRP>. Please use this website to track issues and and submit requests.

## Requirements

The 2016 version of the pipeline no longer requires IRAF/PyRAF, so the installation should be simpler than previous versions.

The pipeline requires the following python modules:

- numpy
- astropy
- ccdproc
- scipy

## Installing python

### Using the Anaconda Distribution

Install Anaconda as per the instructions on the Anaconda web site. The pipeline currently only runs on python 2.7, so download and install that version, not the python 3.x version.

Once anaconda is installed, you can use the **conda** command line tool to get the other packages you will need. Begin by updating conda itself by running:

```
conda update conda
```

If you like, you can now update all packages which are out of date by running:

```
conda update --all
```

Install ccdproc (an “astropy affiliated” package) using the astropy channel:

```
conda install -c astropy ccdproc
```

You should now have all the requirements to run the 2016 version of the MOS-FIRE DRP.

Note: An Anaconda-based install has been tested on both Mac OS X 10.11.5 (“El Capitan”) and on linux: Cent OS 7 (64-bit).

### Using Other Python Install Methods

The DRP support group recommends the anaconda python install and has tested the DRP using that installer, but if an appropriate version of python (e.g. python 2.7) is installed via some other package manager (e.g. apt-get, brew, yum, etc.), then you should be able to install the python package dependencies using either that package manager (if they are available via that package manager) or using pip. For example:

```
pip install numpy
pip install astropy
pip install ccdproc
```

## Download and Install the DRP

Download the zip file of the released master branch from the github page for the DRP, or directly from this link.

Move the zip file to a location on your computer where you want the source code to reside, then unzip the file:

```
unzip MosfireDRP-master.zip
```

Change in to the resulting `MosfireDRP-master/` directory:

```
cd MosfireDRP-master
```

Run the install program:

```
python setup.py install
```

The executable `mospy` should now be in your path. If you used the Anaconda based install, it will be in the Anaconda bin directory (e.g. `~/anaconda2/bin/mospy`).

As of version 2015A, the pipeline can be installed as Ureka package. If you prefer to use this installation procedure, download the pipeline tar file and unpack it. Then execute:

```
ur_setup
```

Go to the mosfire python directory where the `setup.py` file is located (it is the main directory above the MOSFIRE and apps directories). Run:

```
python setup.py install
```

This will copy the MOSFIRE pipeline as a python package located into your ureka environment.

Alternatively, if you would like to modify the MOSFIRE pipeline files, run the following instead:

```
python setup.py develop
```

This will make symoblic links to the MOSFIRE files instead of copying them. Your changes to the pipeline file will now automatically be used when loading the MOSFIRE package.

Directories created by you

- DATA – sub directory in which you can store your data. This is not a necessary sub-directory but may help you manage files. Raw data may be stored in other areas on your disk.
- REDUX – sub directory where reductions will be stored. Also not critical, but helpful.

From now on, if you want to run any pipeline commands, you will always execute “mospy” as seen in our first step in section 5 below. Remember to run `ur_setup` before running the MOSFIRE pipeline.

## Alternate Installation Method

If you prefer the previous installation method, follow these instructions:

### 1) Install Ureka

The pipeline relies on the Ureka Python distribution produced by STScI and Gemini Observatory: <http://ssb.stsci.edu/ureka/>

The DRP was developed using UREKA version 1.0. Navigate to the 1.0 distribution using the url listed above. Follow the instructions at the links to install the package. The UREKA instructions indicate that you need to run `ur_setup` to put ureka in the path. This is automatically completed when you run the drp and it is found in the mospy code. However, if you want to test the ureka package yourself, you will need to run `ur_setup` manually. The latest version of Ureka that is confirmed to work with the pipeline is 1.5.1

### 2) Download the pipeline

1. Start an xterm session on your local machine
2. Run “cd” to navigate to the home directory
3. Download either the .zip file, or the .tar.gz file from the website <https://keck-datareductionpipelines.github.io/MosfireDRP/>. Note that this is the stable and supported version of the pipeline. Alternatively, if you are a github user, you can just clone the repository using: <https://github.com/keck-DataReductionPipelines/MosfireDRP.git>. This is the development version, and it is NOT supported.
4. Expand the zip or tar file and rename the resulting directory. For example:

```
mkdir ~/MOSFIRE
mv MosfireDRP-1.1 ~/MOSFIRE/DRP_CODE
cd ~/MOSFIRE/DRP_CODE # to navigate to that directory
```

### 3) Create Data Directories

Create sub directories for raw data, and reduced data. These sub directories are not specific. You can set up sub directories any way you would like. For the purposes of this manual, we have chosen generic directory names. You may choose to store the raw and reduced data using any directory structure you

would prefer. For our example, we created a raw data directory in the code repository:

```
mkdir ~/MOSFIRE/DRP_CODE/DATA
```

and a reduction directory in the code repository that will store reduced data:

```
mkdir ~/MOSFIRE/DRP_CODE/REDUX
```

#### 4) Copy the mospy file into your bin dir

Navigate to the newly created bin dir:

```
cd ~/MOSFIRE/DRP_CODE/bin
```

Copy the mospy executable to the bin dir

```
cp ../apps/mospy .
```

#### 5) edit mospy in your bin dir and update a few lines of code

Using your favorite editor (emacs ../bin/mospy), update the path for the `ur_setup`. Replace `/home/npk/.ureka/ur_setup` with your `/your_full_path_name/.ureka/ur_setup` full path.

Update the path for the `ur_forget`. Replace `/home/npk/.ureka/ur_setup` with `/your_full_path_name/.ureka/ur_forget`

Update the `MOSPATH` with the full path to the source code directory. Replace `/src2/mosfire/DRP/mosfire` with `/your_full_path_name/MOSFIRE/DRP_CODE`

As an example, the original file might look like the following:

```
#Update the full path to the ureka install for the
# two aliases below.
alias ur_setup 'eval `~/home/npk/.ureka/ur_setup -csh \!*`'
alias ur_forget 'eval `~/home/npk/.ureka/ur_forget -csh \!*`'

# If pythonpath is not previously defined, define it so that
# the setenv works below..
if (! $?PYTHONPATH ) setenv PYTHONPATH

#Update the full path to the mosfire DRP code repository
# example: /src2/mosfire/DRP/mosfire change to /Users/myname/MOSFIRE/DRP_CODE
# in which the sub dirs drivers, apps, badpixel, etc. live
setenv MOSPATH /src2/mosfire/DRP/mosfire
setenv PYTHONPATH ${PYTHONPATH}:${MOSPATH}
```

And the modified version for an observers particular setup may look something like this:

```
#Update the full path to the ureka install for the
# two aliases below.
alias ur_setup 'eval `~/Users/mkassis/.ureka/ur_setup -csh \!*`'
```

```
alias ur_forget 'eval `~/Users/mkassis/.ureka/ur_forget -csh \!*`'

# If pythonpath is not previously defined, define it so that
# the setenv works below..
if (! $?PYTHONPATH ) setenv PYTHONPATH

# Update the full path to the mosfire DRP code repository
# example: /src2/mosfire/DRP/mosfire
# in which the sub dirs drivers, apps, badpixel, etc. live
setenv MOSPATH /Users/mkassis/Documents/KeckInstrs/MOSFIRE/DRP_CODE_March2014/
setenv PYTHONPATH ${PYTHONPATH}:${MOSPATH}
```

#### 6) Ensure that mospy is executable

```
chmod +x mospy
```

#### 7) Update your .cshrc file

Update your .cshrc file with the code bin dir in the path. Add the following line to your .cshrc file:

```
set path = ( #mosfire_drp_bin_dir# $path )
```

for example:

```
set path = ( ~/MOSFIRE/DRP_CODE/bin $path )
```

If you do not normally run csh or tcsh, you may not have a .cshrc file. You will need to create one or download an example file like this one: <http://www2.keck.hawaii.edu/inst/mosfire/.cshrc>. The .cshrc file must be in your home directory. By default, MacOSX does not show files that start with a . But you can access them via the terminal.

For a bash shell:

```
# Adding MOSFIRE pipeline
PATH="/pathtomosfiredrp/MOSFIRE/DRP_CODE/bin:${PATH}"
export PATH
```

#### 8) Now source your .cshrc file

```
source ~/.cshrc
```

This will put your bin dir into your executable path.

The installation is now complete. Take a moment to inventory your directory structure.

DRP\_CODE – Main Code Directory containing all sub-dirs for executable code and in our example the raw and reduced sub-directories.

- MOSFIRE – directory containing the reduction code
- apps – directory containing a few additional applications:
- what – useful pretty printer for files

- handle – the entry point for creating driver files (more later)
- badpixels – directory containing badpixel maps.
- Drivers – directory containing example driver files. These files are used to initiate the reduction process and you will modify them for your specific data sets. This will be discussed in more detail later.
- Driver.py – used for YJH reductions
- K\_driver.py – Contains code specific to K band observations
- Longslit\_driver.py – Longslit reductions
- Long2pos\_driver.py – long2pos and long2pos\_specphot reductions
- Platescale – contains a file that describes the detector plate scale

Directories created by you

- DATA – sub directory in which you can store your data. This is not a necessary sub-directory but may help you manage files. Raw data may be stored in other areas on your disk.
- REDUX – sub directory where reductions will be stored. Also not critical, but helpful.
- bin – has the modified mospy executable command

From now on, if you want to run any pipeline commands, you will always execute “mospy” as seen in our first step in section 5 below.

Before running the drp, you will need a set of spectroscopic data to reduce that includes flats, science observations, and if the observations are K-band, arcs and thermal flats. NOTE: You must preserve Keck’s file naming convention as the DRP uses the file name to parse data sets. The standard naming convention is `mYYMMDD_####.fits`.

If you need to retrieve your data, you may either use a secure copy (scp) assuming your data is still accessible from the Keck/MOSFIRE data directory (contact your SA if you need assistance) or use KOA –the Keck Observatory Archive to navigate to the KOA log in page. From there, KOA has forms where you specify the data to retrieve and will create a tar ball for you to download.

A useful tool is the file translator script that will convert your KOA file names to the standard filenames as they were written to disk during your observing session (koa\_translator). Again, your filenames must preserve the standard naming convention and the koa\_translator script does this for you.

If you do not have data of your own and wish to download the example: Grab the data from: [http://mosfire.googlecode.com/files/DRP\\_Test\\_Case\\_Hband.zip](http://mosfire.googlecode.com/files/DRP_Test_Case_Hband.zip).

Move the test case zip file to a directory of your choice (we will assume ~/Data for the examples in this manual) and unzip the DRP test case file:

```
unzip DRP_Test_Case_Hband.zip
```

This will create a DRP\_Test\_Case\_Hband subdirectory under your current directory which will contain the raw data which you can use to follow along in subsequent steps of the DRP manual.



Now that you have data to reduce, we need to set up the pipeline with the appropriate files so that the drp knows what files to use in the reduction. The handle step will parse the FITS header information and determine what files are associated with each of your masks.

Because the DRP no longer has a designated output directory, you will need to run handle in your designated reduction sub-directory (**reduced** in our example).

```
mkdir reduced
cd reduced
mospy handle /home/[yourhomedir]/Data/DRP_Test_Case_Hband/2012sep10/*fits
```

Please use the full path to the raw data when invoking `mospy handle`.

A lot of data summarizing the observations is output. This includes a table of the observations:

m130514_0132	Flat:mos Y mosmaskA	16.0 s	mosmaskA	Y	YJ
m130114_0133	Flat:mos Y mosmaskA	16.0 s	mosmaskA	Y	YJ
m130114_0134	Flat:mos Y mosmaskA	16.0 s	mosmaskA	Y	YJ
m130114_0135	Flat:mos Y mosmaskA	16.0 s	mosmaskA	Y	YJ
m130114_0136	Flat:mos Y mosmaskA	16.0 s	mosmaskA	Y	YJ
m140114_0137	Flat:mos Y mosmaskA	16.0 s	mosmaskA	Y	YJ
...					

and file lists that organize the observation types:

```
mosmaskA /2013jan14/Y/Unknown.txt
mosmaskA /2013jan14/Y/Align.txt
mosmaskA /2013jan14/Y/MIRA.txt
mosmaskA /2013jan14/Y/Ne.txt
mosmaskA /2013jan14/Y/Offset_2.txt
mosmaskA /2013jan14/Y/Offset_-2.txt
mosmaskA /2013jan14/Y/Flat.txt
mosmaskA /2013jan14/Y/Image.txt
mosmaskA /2013jan14/Y/FlatThermal.txt
mosmaskA /2013jan14/Y/Dark.txt
mosmaskA /2013jan14/Y/Ar.txt
mosmaskA /2013jan14/Y/Aborted.txt
...
```

The handle step creates a set of directories organized as

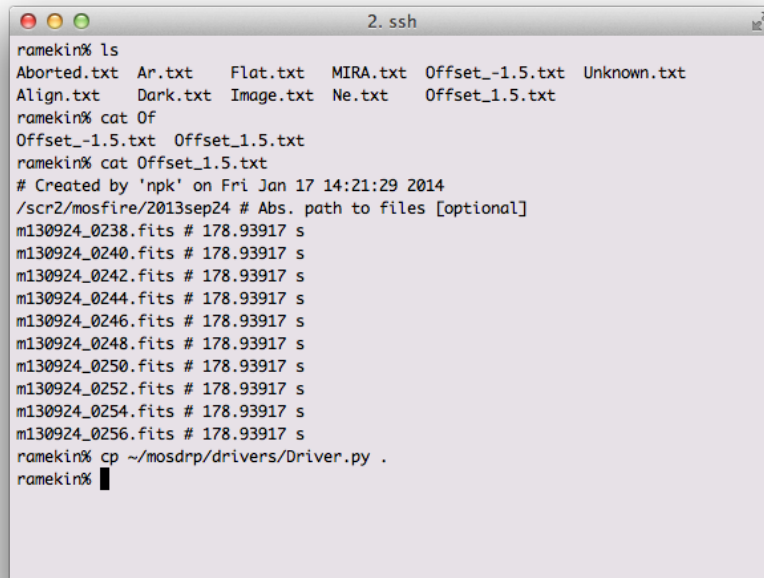
```
[maskname]/[date]/[band]/
```

Containing

- Aborted.txt: Aborted files
- Align.txt: Alignment frames
- Ar.txt: Argon spectra
- Dark.txt: Darks

- Flat.txt: Flat fields
- FlatThermal.txt: Thermal Flats (lamps off)
- Image.txt: Imaging mode
- MIRA.txt: MIRA focus images
- Ne.txt: Neon lamp spectra
- Unknown.txt: Unknown files
- Offset\_\_[p].txt: Science frames

The output directory structure is designed to make finding reduced data easy, and to separate reductions of the same mask across multiple dates. Below is a screen shot showing the example output from an Offset\*.txt file.



```

ramekin% ls
Aborted.txt  Ar.txt      Flat.txt    MIRA.txt    Offset_-1.5.txt  Unknown.txt
Align.txt    Dark.txt    Image.txt   Ne.txt      Offset_1.5.txt
ramekin% cat Of
Offset_-1.5.txt  Offset_1.5.txt
ramekin% cat Offset_1.5.txt
# Created by 'npk' on Fri Jan 17 14:21:29 2014
/scr2/mosfire/2013sep24 # Abs. path to files [optional]
m130924_0238.fits # 178.93917 s
m130924_0240.fits # 178.93917 s
m130924_0242.fits # 178.93917 s
m130924_0244.fits # 178.93917 s
m130924_0246.fits # 178.93917 s
m130924_0248.fits # 178.93917 s
m130924_0250.fits # 178.93917 s
m130924_0252.fits # 178.93917 s
m130924_0254.fits # 178.93917 s
m130924_0256.fits # 178.93917 s
ramekin% cp ~/mosdrp/drivers/Driver.py .
ramekin% █

```

Figure 1: Screenshot

For longslit observations, separate offsets files are created for each object, but the same offset files are used if the same object is observed many times during the night. You might want to separate the different observations of the same object.

For long2pos observations, again different offset files are created for each object. Besides, the suffixes `_PosA` and `_PosC` are added to the offset files to identify the two left and right positions used in the observations.

The following section describes in details how to use the driver file to control

the pipeline processes, and points at a number of standard driver files that we provide for reference.

The pipeline is able to produce a driver file automatically for most cases, thus removing the need to copy one of the standard files and manually edit it.

To generate the driver file, go to the directory where your Offset files live, and where the reduction is going to happen and type:

**mospy AutoDriver**

This will generate a file called `Driver.py`, which you should inspect before running it. Highly specialized cases such as particular combinations of sky lines and arcs might not be dealt with correctly. Note that the automatic generation of the driver file works for `long2pos` and `longslit` as well.

To handle special cases, the automatic generation of the driver file makes a number of assumptions, that might not be correct for your science case.

1. If either `Ar.txt` or `Ne.txt` or both are available, they are being used.
2. If the band is K, and `FlatThermal.txt` is available, it is used
3. For `long2pos`: if no arcs are available, only `specphot` in non spectrophotometric mode can be reduced and the pipeline will attempt to use the sky lines. Note that is likely to fail, as sky lines are too faint in short exposures for an accurate wavelength determination. The spectrophotometric mode contains wide slits that cannot be reduced using sky lines only.
4. In `longslit`, the pipeline will try to determine the size of the slit using the mask name. For example, if the maskname is `LONGSLIT-3x0.7`, the pipeline assumes that you have used 3 slits to generate the `longslit` and that they are centered around the middle line of the detector.
5. If any of the mandatory observations are missing (such as the flat fields), the pipeline will still generate a `Driver.py` file, but it will contain warnings about the missing files, and it will NOT run correctly.
6. If multiple observations of different stars are done in `long2pos` or in `longslit` mode, the pipeline will generate multiple driver files, one for each object. If the same object is observed multiple times during the same night, all the observations will end up in the same driver file. If you are observing a telluric standard at different times and you need to have separate spectra, you need to manually create Offset files and Driver files.

The driver file controls all the pipeline steps, and in the `drivers` sub-directory, you will find a number of driver files: `Driver.py`, `K_Driver.py`, `Long2pos_driver.py`, and `Longslit_Driver.py`. The `Driver` and `K_Driver` will reduce your science data for bands Y, J, and H (this includes the sample data set). The K band requires a special approach because there are too few bright night-sky emission lines at the red end and so the `K_Driver` synthesizes arcclamps and night sky lines. The `Long2pos_driver.py` handles `long2pos` and `long2pos_specphot` observations, while the `Longslit_driver.py` deals with observations of single objects using a `longslit` configuration.

The driver.py files included with the code download contains execution lines that are commented out. For this example, we will run the driver file one line at a time, but as you become familiar with the DRP process, you will develop your own driver file execution sequencing. Although in the future we hope to further automate the driver file, currently some steps require you to update the inputs with filenames created from previous steps.

Below is a driver.py file:

```
import os, time
import MOSFIRE

from MOSFIRE import Background, Combine, Detector, Flats, IO, Options, \
    Rectify
from MOSFIRE import Wavelength

import numpy as np, pylab as pl, pyfits as pf

np.seterr(all="ignore")

#Update the insertmaskname with the name of the mask
#Update S with the filter band Y,J,H,or K
maskname = 'insertmaskname'
band = 'S'

flatops = Options.flat
waveops = Options.wavelength

obsfiles = ['Offset_1.5.txt', 'Offset_-1.5.txt']

#Flats.handle_flats('Flat.txt', maskname, band, flatops)
#Wavelength.imcombine(obsfiles, maskname, band, waveops)
#Wavelength.fit_lambda_interactively(maskname, band, obsfiles,
#    waveops)
#Wavelength.fit_lambda(maskname, band, obsfiles, obsfiles,
#    waveops)

#Wavelength.apply_lambda_simple(maskname, band, obsfiles, waveops)
#Background.handle_background(obsfiles,
#    'lambda_solution_wave_stack_H_m130429_0224-0249.fits',
#    maskname, band, waveops)

redfiles = ["eps_" + file + ".fits" for file in obsfiles]
#Rectify.handle_rectification(maskname, redfiles,
#    "lambda_solution_wave_stack_H_m130429_0224-0249.fits",
#    band,
#    "/scr2/npk/mosfire/2013apr29/m130429_0224.fits",
```

```
# waveops)
#
```

To set up your driver file do the following:

1. Navigate to the desired output directory created by handle: `cd ~/Data/reducedMOSFIRE_DRP_MASK/2012sep10/H`
2. Copy the appropriate driver file: `cp ~/MosfireDRP-master/drivers/Driver.py`. NOTE: If you are observing a K band mask you'll want to copy the `K_driver.py` file over.
3. Edit driver.py (see bold text in driver file example)
  - Update maskname
  - Update band to be Y,J,H
  - Update the `Offset_#.txt` name. Handle creates offset files with names that are specific to the nod throw. The default driver file uses 1.5 arcsec offsets in the file name.

In the sections that follow, we will describe the function and outputs of the commented lines found in the driver file starting with the creation of flats.

If you prefer to override the standard naming convention of the output files, you can specify

```
target = "targetname"
```

at the beginning of the driver file. If you do so, remember to also add `target=target` to both the Background and Rectify steps. Example:

```
Background.handle_background(obsfiles,
    'lambda_solution_wave_stack_H_m150428_0091-0091.fits',
    maskname, band, waveops, target=target)
```

The first action the driver file will take is to generate a pixel flat and slit edge tracing. To initiate the flat generation, uncomment the line below in the Driver.py file:

```
#Flats.handle_flats('Flat.txt', maskname, band, flatops)
```

and in your xterm run the DRP

```
> mospay Driver.py
```

Example output from the xterm session

```
> mospay Driver.py
... Truncated output ...
Flat written to combflat_2d_H.fits
```

```
00] Finding Slit Edges for BX113 ending at 1901. Slit composed of 3 CSU slits
01] Finding Slit Edges for BX129 ending at 1812. Slit composed of 2 CSU slits
02] Finding Slit Edges for xS15 ending at 1768. Slit composed of 1 CSU slits
Skipping (wavelength pixel): 10
```

03] Finding Slit Edges for BX131 ending at 1680. Slit composed of 2 CSU slits

The slit names output to the screen should look familiar as they originated from the mask design process. The output files from this process are the following:

Filename	Contains
<code>combflat_2d_J.fits</code>	FITS image of the flats
<code>flatcombine.lst</code>	The list of files used in the creation of the flat. Contains the full path name to the files.
<code>pixelflat_2d_J.fits</code>	FITS image of the normalized flat. This is the flat used in other reduction steps.
<code>slit-edges_J.npy</code>	File containing the slit edge information
<code>slit-edges_J.reg</code>	DS9 regions file that may be overlayed to show the locations of the slits.

At the end, check the output in ds9. For example:

```
> ds9 pixelflat_2d_H.fits -region slit-edges_H.reg
```

The regions file overlayed on the pixelflat image should look something like:

The green lines must trace the edge of the slit. If they don't, then the flat step failed. All values should be around 1.0. There are some big features in the detector that you will become familiar with over time.

## K-band flats

At K-band, the dome is hot enough that light is detected at the longest wavelengths at a level of a few hundred counts. Little to no light is seen at the shortest wavelengths. The light from the dome is not entering MOSFIRE at the same angles that the light from the spot illuminated on the dome by the dome lights. Some observers may wish to correct for this difference by subtracting the thermal flat emission from the dome flat emission before normalizing the flats. To complete this flat subtraction, you use the optional keyword `lampofflist` in the flat process as seen in the command below:

```
Flats.handle_flats('Flat.txt', maskname, band, flatops, lampOffList='FlatThermal.txt')
```

If thermal flats were included in your calibration sequence (default behavior for K-band), then the `FlatThermal.txt` file should be populated with a list of thermal flats. Use `FlatThermal.txt` as the list or modify it as you see necessary.

The outputs from the flat process will include two additional files.

- `combflat_lamps_off_2d_K.fits`
- `combflat_lamps_on_2d_K.fits`

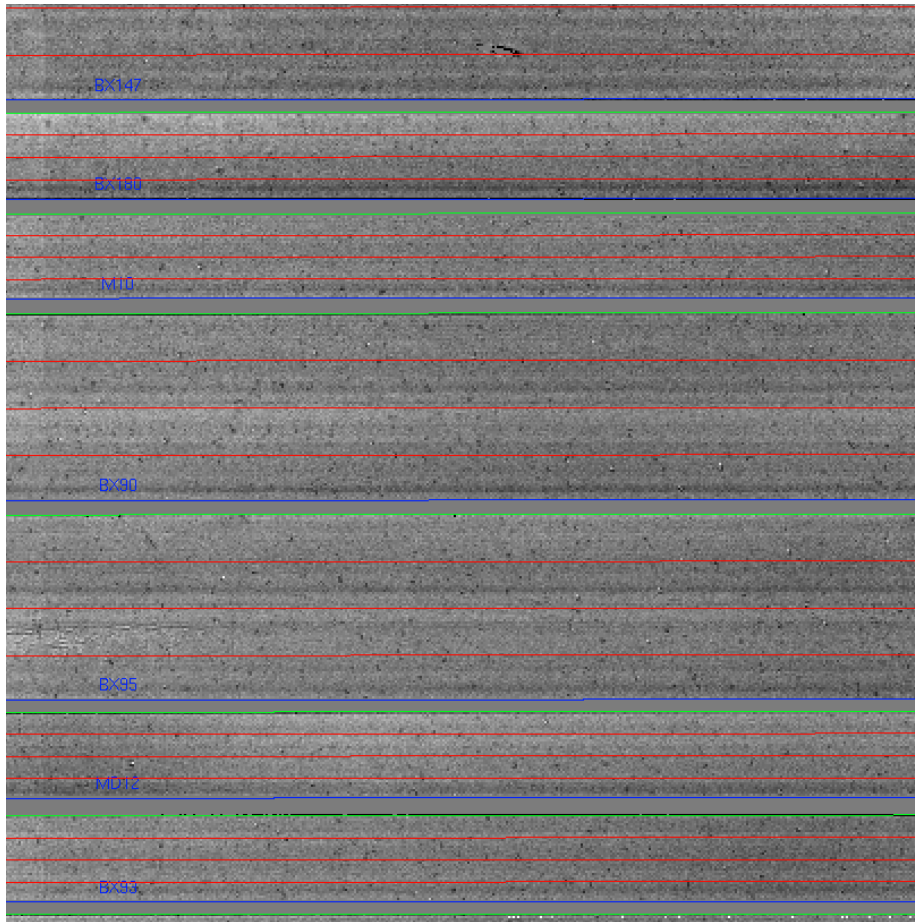


Figure 2: Screenshot

and now the `combflat_2d_K.fits` being the difference between the two files.

In the shorter wavebands, when using the recommended exposure times, the wavelength calibration is performed on night sky lines. The mospay Wavelength module is responsible for these operations. See the example driver file in section 7.

## Combine files

First step is to produce a file with which you will train your wavelength solution. Since we're using night sky lines for training, the approach is to combine individual science exposures. This is performed by the python `Wavelength.imcombine` routine. For a lot of users, this will look something like in the `Driver.py` file:

```
Wavelength.imcombine(obsfiles, maskname, band, waveops)
```

The first parameter is `obsfiles` which is a python string array indicating the list of files in the offset positions. Note that `obsfiles` has defaults of "`Offset_1.5.txt`" and "`Offset_-1.5.txt`" and may need to be updated as described in section 6.

Suppose you want to exclude a file for reasons such as weather or telescope fault, simply remove the offending file from the appropriate `Offset_*.txt`. Likewise, you are welcome to add files in as you like, such as observations from the previous night.

Outputs of this step are:

Filename	Contains
<code>wave_stack_[band]_[range].fits</code>	Aligned and combined image of the files to be used for the wavelength solution.

## Interactive wavelength fitting

The next step is to use the `wave_stack_*.fits` file and determine an initial wavelength solution for each slit. During this process, we interactively fit the lines using a gui that displays. To initiate this process, uncomment the line in the `Driver.py` file:

```
#Wavelength.fit_lambda(maskname, band, obsfiles, obsfiles, waveops)
```

And then re-execute the driver file:

```
mospay Driver.py
```

when you run this step, a GUI window appears.

The interactive wavelength solving window. This is a J-band night sky spectrum.



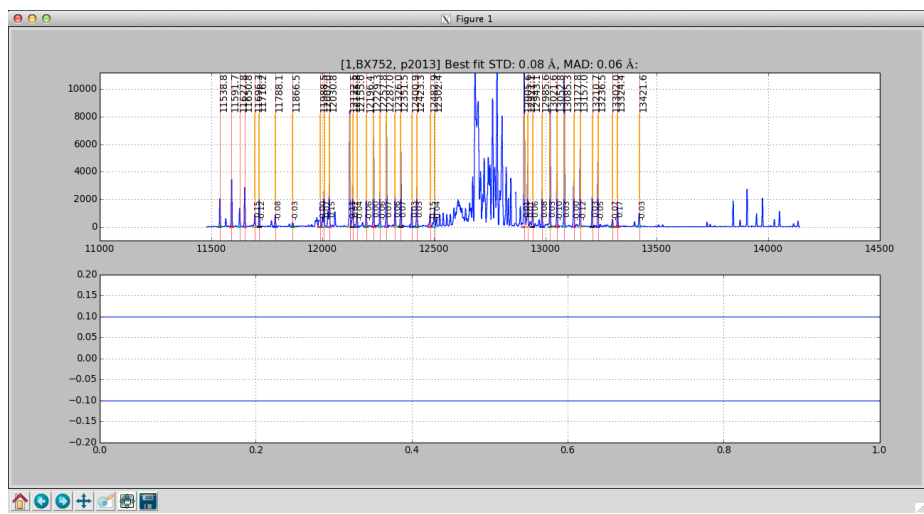


Figure 3: Screenshot

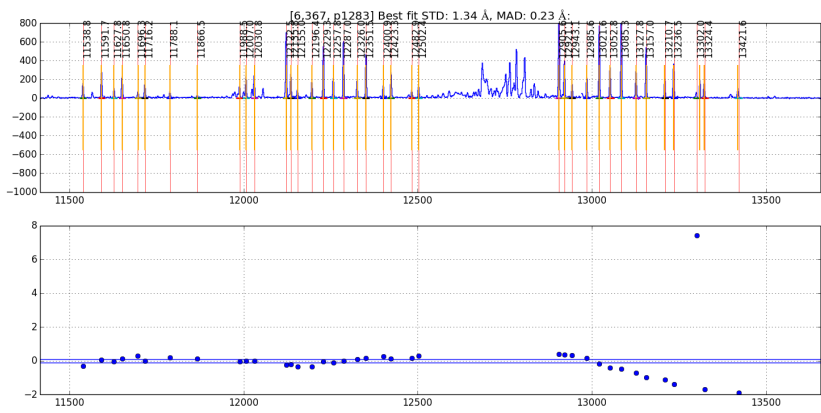


Figure 4: Screenshot

The interactive wavelength solving window showing an initial fit. This is a J-band night sky spectrum and one of the night sky lines on the right hand side is clearly a poor fit compared to the rest of the identified lines.

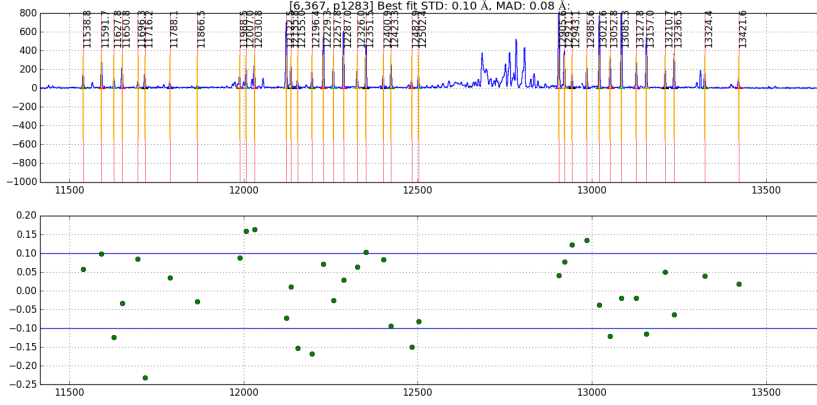


Figure 5: Screenshot

The interactive wavelength solving window showing a good fit with the initial poor line removed from the calculation. The interactive wavelength solving window showing an initial fit. This is a J-band night sky spectrum and one of the night sky lines on the right hand side is clearly a poor fit compared to the rest of the identified lines.

Plotted in the gui will be a sky line spectrum and vertical lines denoting positions and wavelengths of the sky lines. Your goal is to help the pipeline by identifying the night sky lines in the center of each slit. Once you come up with a good solution in the center, the pipeline will propagate it spatially along the slit. In the gui, Press ? to see a list of commands in the console window. The list of commands available on the GUI are:

- c - to center on the nearest peak (first thing to do to shift the initial wavelength guess)
- c - Center the nearest line at the cursor position
- - Fit fit the data
- f - Alternate way to fit the data, equivalent to but may cause the spectrum to become full screen.
- k - Toggles k-sigma clipping on and off. Also performs a new fit.
- b - Turns on bypass mode. From now on, the interactive window is not displayed and the fit continues automatically.
- d - Delete a point (remove the wackadoos)
- n - proceed to the Next object
- p - return to back to the Previous object

- r - Reset the current slit (try this if the plot looks strange)
- z - Zoom at cursor position
- x - Unzoom: full screen
- s - Save figure to disk
- h - Help
- q - Quit and save results

Below is a rough procedure for completing the interactive fitting process. The steps you need to take are as follows.

- First, check to see if the orange lines match up with obvious night sky lines. If not the expected position does not match the actually position of the line do the following:
  - Place your cursor over a line
  - Press the “c” button that will shift the predicted position to the observed line.
- Press “f” to fit. An initial fit is done automatically (starting with version 2015A) A Chebyshev polynomial,  $f$ , such that  $f(\text{pixel \#})$  returns the wavelength in Angstroms.
- You can chose to use k-sigma clipping to avoid the manual operation of removing bad lines with the “k” key.
- Press “x” to unzoom to the full size region
- Assess the fit:
  - If a line is poorly fit and should be removed
    - \* Move the cursor to the line
    - \* Press “d” to delete the line from the fit
  - For good fits, the residual points turn green.
- When the satisfied with the fit, press “n” to move to the next object.
- If you want to disable the interactive fit and switch to the automatic fit, press “b” (bypass)
- Repeat the process above until you see the red Done! text in the center of your screen.
- Press “q” to quit the interactive gui and move to the next step.

The prompt should return following the fitting process. The outputs from this process are:

Filename	Contains
<code>barset.npy</code>	bar positions for each slit are specified
<code>lambda_center_centers_of_line_stack_and_fill_pos.npy</code>	The coefficients of the polynomial fit to the measured lines.

## Wavelength fitting for the entire slit

The next step in the wavelength fitting process is to propagate the solution spatially along each slit. To complete this process we uncomment the line in the Driver.py file:

```
#Wavelength.fit_lambda(maskname, band, obsfiles, obsfiles, waveops)
```

This is one of the longer running processes and the output should look something like:

```
...
resid ang S09 @ p 978: 0.10 rms 0.07 mad [shift-22]
resid ang S09 @ p 979: 0.09 rms 0.06 mad [shift-22]
resid ang S09 @ p 980: 0.10 rms 0.06 mad [shift-22]
resid ang S09 @ p 981: 0.09 rms 0.06 mad [shift-22]
resid ang S09 @ p 982: 0.08 rms 0.05 mad [shift-22]
resid ang S09 @ p 983: 0.08 rms 0.04 mad [shift-22]
...
```

The prompt should return following the fitting process. The outputs from this process are:

Filename	Contains
lambda_coeffs_waveops	Stack of 130 fit for 0443-0445 within the slit

## Apply the wavelength solution

The last step in the wavelength fitting process is to apply the solution and create maps of the wavelength for the data set. To complete this process we uncomment the line in the Driver.py file:

```
#Wavelength.apply_lambda_simple(maskname, band, obsfiles, waveops)
```

The prompt should return following the fitting process. The outputs from this process are:

Filename	Contains
lambda_solution_waveops	Stack of 130 wave length for each pixel in the spectra
sigs_solution_waveops	Stack of 130 sigs and wavelength position for each pixel in the spectra
rectified_wave_stack	Stack of 130 14-0443-0445 wave length rectified resampled sky emission. A column in the image contains all pixels at the same wavelength.

The night sky lines at the red end of the K-band are too faint to achieve small-fraction of a pixel RMS wavelength calibration. You will have to observe a Neon and Argon arc lamps during your afternoon calibrations. By default, the calibration script at the observatory is setup to acquire both the Ne and Argon arcs.

Because the beams emanating from the arclamp do not follow the same path as the beams coming from the sky, there will be a slight difference between the two solutions. For the aforementioned beam matching reason, the most accurate solution is the night sky lines. Thus, the code has to be clever about merging the two solutions.

The following subsections describe the additional steps that are necessary to process the arcline data and combine the arcs and night sky line wavelength solutions.

## Combine the arc line spectra

Just like the step in section 8.1 where you combined the science frames to create nightsky line spectra, we first need to combine the arcline data. The arcs are typically three files and you should see them listed in the Ne.txt and Ar.txt file lists in your K band sub directory. To combine the images simply uncomment and run:

```
Wavelength.imcombine('Ne.txt', maskname, band, waveops)
Wavelength.imcombine('Ar.txt', maskname, band, waveops)
```

## Identify arc lines using night sky solution

Instead of having to interactively determine the wavelength solution for the arcs like we did in section 8.2 for the night sky lines, we are going to use the solutions for the night sky lines as a first approximation for the arcs. This may usually be done because the arcs differ from the night sky lines by a fractions of pixels. You are welcome to interactively solve the neon lamp solution with the `Wavelength.fit_lambda_interactively` routine; however, the need to run the interactive solution method should be rare.

To apply the solution from the night sky lines to the arcs center slit position, uncomment and run the following lines.

```
Wavelength.apply_interactive(maskname, band, waveops, apply=obsfiles, to='Ne.txt', neon=True)
Wavelength.apply_interactive(maskname, band, waveops, apply=obsfiles, to='Ar.txt', argon=True)
```

This step, when run will produce output like:

```
slitno  1 STD: 0.16 MAD: 0.06
slitno  2 STD: 0.03 MAD: 0.02
```

```
slitno 3 STD: 0.04 MAD: 0.04
slitno 4 STD: 0.05 MAD: 0.01
```

For each slit, a new solution is generated for the neon line. The output mimics that described previously where STD is the standard deviation and MAD is the median absolute deviation in angstroms.

## Wavelength fitting for the entire slit using arcs

The next step in the wavelength fitting process is to propagate the arc solution spatially along each slit. Again this is the same process essentially as the fit for the night sky lines. This moves along each row for the slit to determine a wavelength solution. The output files are comparable to those in step 8.3

```
Wavelength.fit_lambda(maskname, band, 'Ne.txt', 'Ne.txt', waveops, wavenames2='Ar.txt')
```

You will note that the Ar.txt file is listed as an optional argument. If you choose not to use the Argon lamps, then you may simply remove the optional wavenames2 and execute this using only the Ne arcs.

Again, this process takes some time to complete.

## Merge the arc and sky lists

In this portion of the procedure, we merge the two lists. These commands may not be run individually. Instead any command containing the variable LROI needs to be run in one mospdy driver file session in order to pass the LROI variable. In this section we determine the offsets between the region of overlap between the night skylines and the arclines. A plot of that region is displayed. To move on you will have to close the plot.

To execute this step you will need to uncomment the following lines in the driver file.

```
LROI = [[21000, 22800]] * 1
LROIIs = Wavelength.check_wavelength_roi(maskname, band, obsfiles, 'Ne.txt', LROI, waveops)
Wavelength.apply_lambda_simple(maskname, band, 'Ne.txt', waveops)
Wavelength.apply_lambda_sky_and_arc(maskname, band, obsfiles, 'Ne.txt', LROIIs, waveops, ne
```

The merged output solution will have a filename that looks like:

```
merged_lambda_coeffs_wave_stack_K_m130114_0451-0453_and_wave_stack_K_m140508_0197-0199.npy
merged_lambda_solution_wave_stack_K_m130114_0451-0453_and_wave_stack_K_m140508_0197-0199.fits
merged_rectified_wave_stack_K_m130114_0451-0453_and_wave_stack_K_m140508_0197-0199.fits.gz
```

The output files have the same format as those in section 8.4 and will need to be used as inputs to the Background and Rectify section below.

This DRP assumes that targets are nodded along the slit with integration times as described on the instrument web page. The integration times described were selected such that the shot-noise in the region between night sky lines is over 5x larger than the read noise of a 16-fowler sample. For MOSFIRE, we define this as background limited.

Despite MOSFIRE's (unprecedented) f/2.0 camera, the desired integration time for background-limited operation is longer than the time for the atmosphere to vary by several percent. As a result, a further background subtraction step is required to remove the residual features. The step is performed by a function called `background_subtract_helper()` and follows the notation and procedure outlined in Kasen (2003; PASP 115). For most users, you'll want to use the standard Driver file and not worry about the details.

In the Driver.py file you want to uncomment the following:

```
Background.handle_background(obsfiles, 'lambda_solution_wave_stack_J_m130114_0443-0445.fits
```

The `lambda_solution_wave_stack` file needs to be updated in your driver file. If reducing Kband, be sure to use the merged `wave_stack` solution. It is one of the outputs from the last wavelength step (see section 8).

In this step:

- Apply the flat field corrections
- A position files are combined (`Offset_*.txt`)
- B postion files are combined (`Offset_*.txt`)
- Subtract A-B
- Correct for small differences in the background sky emission

## Output Files

The background subtraction step produces the following files. As usual elements in [brackets] are replaced with the value for that mask.

Filename	Content (units)
<code>eps_Offset_[###].txt.fits</code>	Average signal in the ### stack ()
<code>var_Offset_[###].txt.fits</code>	Total variance in each pixel of above file ()
<code>itimes_Offset_[###].txt.fits</code>	Total exposure time in each pixel of above files ()
<code>sub_[maskname]_[bandname]_[plan].fits</code>	Difference (but non background subtracted) file ()
<code>bsub_[maskname]_[bandname]_[plan].fits</code>	Background subtracted signal ()
<code>bmod_[maskname]_[bandname]_[plan].fits</code>	Background model signal ()
<code>var_[maskname]_[bandname]_[plan].fits</code>	Total variance

Filename	Content (units)
<code>itime_[maskname]_{bandname}_[plan].fits</code>	Average integration time

There is redundant information in the above set of files. For instance:

```
sub_Mask_K_A-B.fits = eps_Offset_1.5.txt.fits - eps_Offset_-1.5.txt.fits
var_Mask_K_A-B.fits = var_Offset_1.5.txt.fits + var_Offset_-1.5.txt.fits
itime_Mask_K_A-B.fits = mean(itime_Offset_1.5.txt.fits, itime_Offset_-1.5.txt.fits)
```

If you want to drill further into how these are constructed, examine the `Background.py` `imcombine` and `handle_background` functions.

Recitified outputs are also computed as tabulated in the table below.

Filename	Content (units)
<code>[maskname]_rectified_[bandname]_[plan].fits</code>	Signal ()
<code>[maskname]_rectified_itime_[bandname]_[plan].fits</code>	Integration time
<code>[maskname]_rectified_var_[bandname]_[plan].fits</code>	Variance
<code>[maskname]_rectified_sn_[bandname]_[plan].fits</code>	Signal to noise ()

Note that signal to noise is computed as follows:

$$sn = \frac{\text{signal} \cdot \text{integration time}}{\sqrt{\text{variance}}}$$

Figure 6: Equation

yes, we violate the first normal form for convenience. Also note that the STD is computed assuming the detector has a read noise of `Detector.RN` (documented in the MOSFIRE Pre Ship Review as 21 electron) per fowler sample. Thus, the final STD is

$$STD = \frac{21 \text{ electron}}{\sqrt{N_{reads}}} + \sqrt{\# \text{ detected electron}}$$

Figure 7: Equation

assuming the gain in `Detector.gain`. Note that there is no shot noise from dark current, which was measured to be negligible at pre-ship review.



An example of what the output looks like is here:

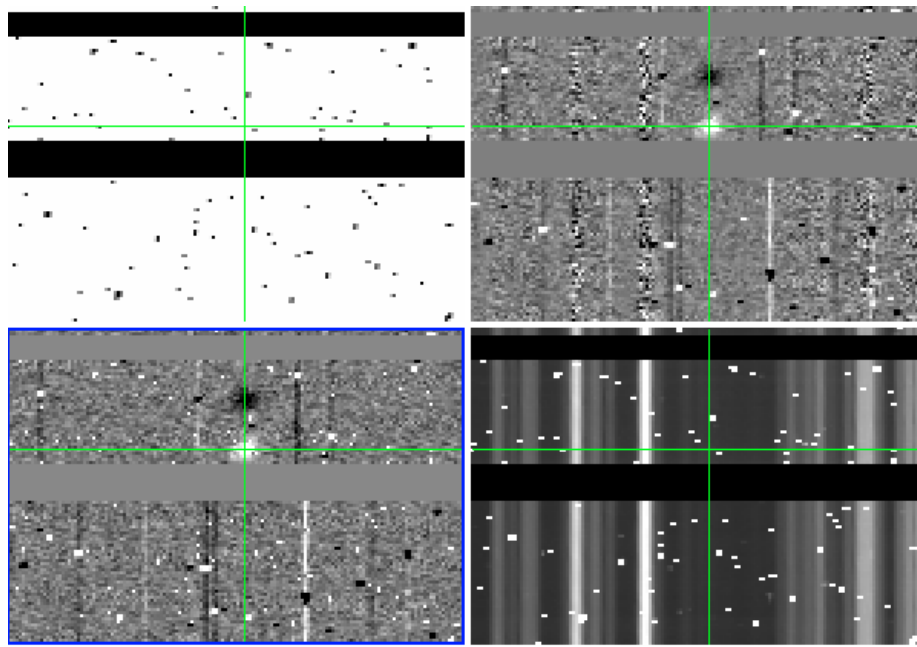


Figure 8: Screenshot

Image showing the itime, bsub, and rectified wavelength images. The green crosses are marking the location of the same pixel in each image.

The next step in the reduction process is to combine the wavelength solution with the background subtracted images and then shift and combine the nod positions. If reducing Kband, be sure to use the merged wave\_stack solution. To do this we uncomment the following lines in the Driver.py file:

```
redfiles = ["eps_" + file + ".fits" for file in obsfiles]
Rectify.handle_rectification(maskname, redfiles,
    'lambda_solution_wave_stack_J_m130114_0443-0445.fits',
    band,
    waveops)
```

The output from this procedure produces four files for every slit.

Filename	Content (units)
[maskname]_[band]_[object]_eps.fits	Signal ()
[maskname]_[band]_[object]_itime.fits	Integration time
[maskname]_[band]_[object]_sig.fits	Variance ?
[maskname]_[band]_[object]_snrs.fits	Signal to noise ()

There is also four images without the “object” in the name. These four files contain the composit spectra with all spectra aligned spectrally and both beams combined. In the \*eps.fits files, you will see two negative traces and one positive trace. For a two position nod, the eps files is (A-B) +((B-A)shifted).

When extracting the emission from an object or measuring the position of an emission line, you should be accessing the \*eps.fits files with the wavelength solution written into the WCS information.

## Interactive Spectral Extraction Instructions

The final step is to extract a 1D spectrum for each object in each slit. The final line of the `Driver.py` (or equivalent) file will looks something like this:

```
Extract.extract_spectra(maskname, band, interactive=(not bypassflag))
```

This will iterate through all the slits for this mask-band combination and if the interactive flag is set, then it will show a plot of the spatial profile of each slit (collapsed in the spectral direction). By default, the software will make a guess at one aperture for each slit. The apertures are indicated by a yellow shaded region and their center position and half width in pixels is annotated near the top of each shaded region.

The apertures define the pixels which will be used as input to the optimal spectral extraction (Horne 1986) algorithm. Having wide a wide aperture should not add additional noise as that will be optimized during the spectral extraction step. The apertures are shown here in order for the user to verify 1) that there is no overlap between adjacent objects, 2) that the apertures are wide enough to reasonably encompass all flux from the object, and 3) that all objects have properly defined apertures.

The user can add, modify, or delete apertures interactively using this plot window.

To delete an existing aperture: place the mouse near the center of the aperture and press the “d” key.

To add an aperture by fitting a gaussian to the profile: place the mouse near the peak of the profile and press the “g” key. The half width of the aperture will be set at 5 times the sigma of the fitted gaussian.

To add an aperture manually: place the mouse in the X position where the new aperture should be centered and press the “a” key. Then type the half width (in pixels) for that aperture in response to the query in the terminal.

To modify the half width of an existing aperture: place the mouse near the center of the aperture and press the “w” key. Then type the half width (in pixels) for that aperture in response to the query in the terminal.

To modify the center position of an existing aperture: place the mouse near the center of the aperture and press the “p” key. Then type the position (in pixels)

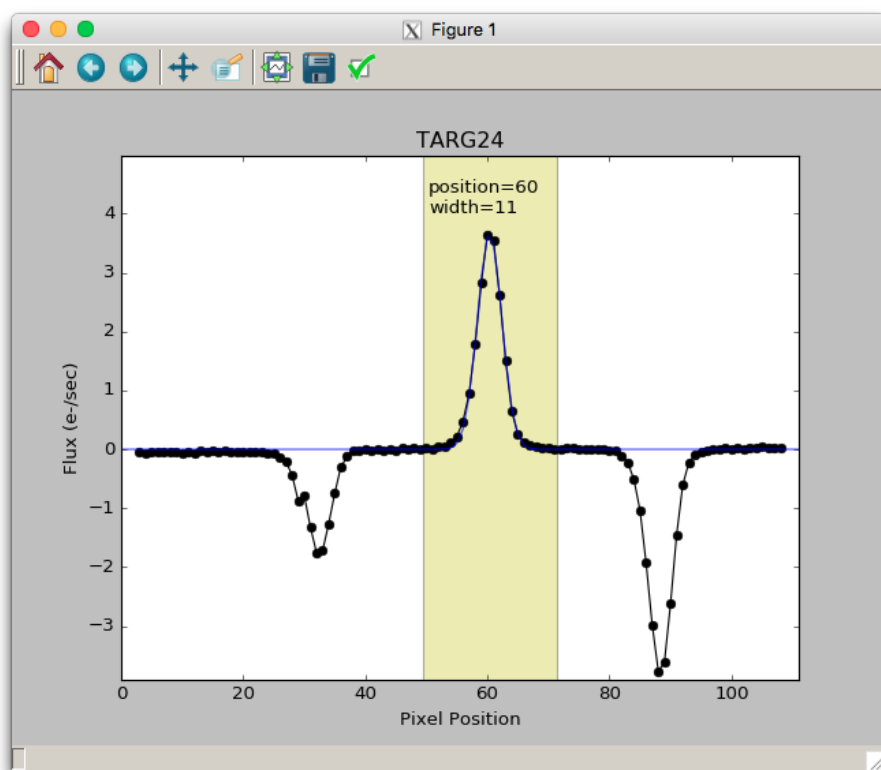


Figure 9: Screenshot

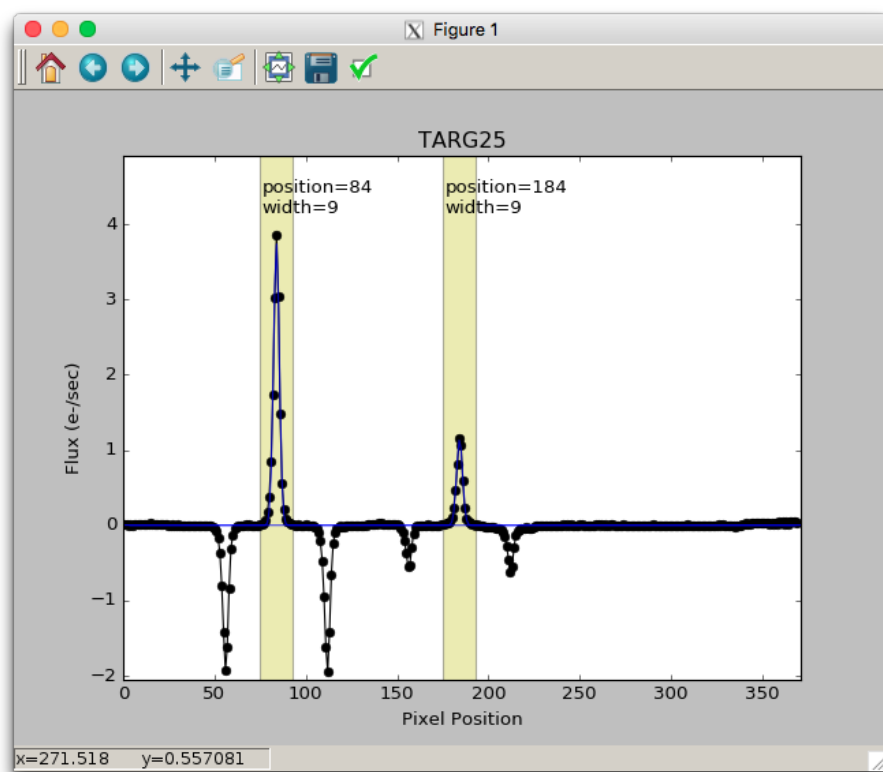


Figure 10: Screenshot

for that aperture in response to the query in the terminal.

When you are done adding or removing apertures, close the interactive plot window by clicking the close button in the upper right corner (or by whatever method is typical for your OS or windowing system) or press the “q” or “n” keys (for “quit” or “next” respectively).

## Spectral Extraction Results

Whether you used the interactive tool for spectral extraction or allowed the software to automatically guess at the apertures to extract, the software will output both a FITS version of the resulting 1D spectrum and an PNG plot.

These filenames will have the form:

```
[maskname]_[band]_[targetname]_[aperture].png  
[maskname]_[band]_[targetname]_1D_[aperture].fits
```

where [aperture] is a two digit integer indication which aperture for that slit this corresponds to (zero based). If the apertures were determined automatically by the software, then only one aperture will have been generated for each slit, so all files will end in \_00.png or \_1D\_00.fits.

A special driver is provided for long2pos reductions. The driver can also be generated automatically.

As a reminder, these observations are taken using a script which is run either from the command line (acq\_long2pos) or via the background menu. The script produces different results depending on whether the long2pos mask was setup in science mode (only narrow slits) or in alignment mode (narrow and wide slits).

In the general case of a combination of narrow and wide slits, each run of the script generates 6 images, 3 for each of the two slits. We will refer to the two positions as position A and position C (position B is the initial position used only for alignment).

Depending on when your data was generated, you might find different Offset files in your directory. Files generated before June 10, 2015 use a different set of YOFFSET keywords than files generated after that date. Unfortunately, the set of keywords generated before June 10, 2015 is not compatible with the pipeline and must be updated: for this we provide a special set of instructions as part of as the driver file to automatically update the keywords.

For files generated before June 10, 2015, you will find 6 Offset files, named Offset\_XXX\_object\_name\_PosY.txt, where XXX can be -21, -14, -7, 7, 14 and 21, the object name is taken from the object keyword, and Y can be either A or C. Similar names are produced if the observations has the correct keywords, but in that case XXX will be one of -7, 0, or 7.

It is important to notice that the reduction described here is based on the assumption that proper arc lamps are obtained in the afternoon. Specifically, either a Ne or Ar calibration must be obtained with the long2pos mask executed in science mode, and not in alignment mode. In science mode the wide part of the slits is not present. If the slit was executed in alignment mode, the wide part of the slits would prevent a wavelength calibration.

Note that this also means that if you took your science data at night in long2pos\_specphot mode, the mask name of your science file might be long2pos\_specphot, rather than long2pos, and the arcs and flats might end up in the wrong subdirectory when the files are processed via mospy handle. In this case it will be necessary to copy Ar.txt, Ne.txt and Flat\*.txt from the directory long2pos to your long2pos\_specphot directory.

Let's now look at the driver file. The declaration "longslit =" is used to define the pixel boundaries of the long2pos observations. In general, it is correct and should not be changed. It might need to be updated in the future if a new long2pos mask is used. Note that it is important to specify 'mode'='long2pos'

The following section describes the rather long list of Offset files that we will use for the reduction.

For observations obtained before June 10, 2015, this section might look like this:

```
obsfiles_posC_narrow = ['Offset_-21_HIP85871_PosC.txt', 'Offset_-7_HIP85871_PosC.txt']
targetCnarrow = "HIP85871_posC_narrow"
```

```
obsfiles_posA_narrow = ['Offset_7_HIP85871_PosA.txt', 'Offset_21_HIP85871_PosA.txt']
targetAnarrow = "HIP85871_posA_narrow"
```

```
obsfiles_posC_wide = ['Offset_-14_HIP85871_PosC.txt', 'Offset_-7_HIP85871_PosC.txt']
targetCwide = "HIP85871_posC_wide"
```

```
obsfiles_posA_wide = ['Offset_14_HIP85871_PosA.txt', 'Offset_21_HIP85871_PosA.txt']
targetAwide = "HIP85871_posA_wide"
```

Files -21\_PosC and -7\_PosC are the A and B positions for the C pointing, files 7 and 21 are the A and B positions for the A pointing. For the wide slits, file 7\_PosC is used as a sky (B) for the -14\_PosC position, and file 21\_PosA is used as a sky for the 14\_PosA position. The target keywords must also be specified to avoid accidental overwrite of intermediate files.

For files obtained after June 10, 2015, the same section would look like this:

```
obsfiles_posC_narrow = ['Offset_7_FS134_posC.txt', 'Offset_-7_FS134_PosC.txt']
targetCnarrow = "FS134_posC_narrow"
obsfiles_posA_narrow = ['Offset_7_FS134_posA.txt', 'Offset_-7_FS134_PosA.txt']
targetAnarrow = "FS134_posA_narrow"
obsfiles_posC_wide = ['Offset_0_FS134_posC.txt', 'Offset_-7_FS134_PosC.txt']
targetCwide = "FS134_posC_wide"
```

```
obsfiles_posA_wide = ['Offset_0_FS134_posA.txt', 'Offset_-7_FS134_PosA.txt']
targetAwide = "FS134_posA_wide"
```

The first step is to produce a flat field.

```
Flats.handle_flats('Flat.txt', maskname, band, flatops, longslit = longslit)
```

or

```
Flats.handle_flats('Flat.txt', maskname, band, flatops, lampOffList='FlatThermal.txt', longslit = longslit)
```

Using argon (or neon) lines, we can now produce a wavelength calibration.

```
Wavelength.imcombine(argon, maskname, band, waveops)
Wavelength.fit_lambda_interactively(maskname, band, argon, waveops, longslit=longslit, argon_lines=argon_lines)
Wavelength.fit_lambda(maskname, band, argon, argon, waveops, longslit=longslit)
Wavelength.apply_lambda_simple(maskname, band, argon, waveops, longslit=longslit, smooth=True)
```

While using the interactive fitting, note that there are two slits to fit.

The next section of the driver reduces the narrow slits. The optional line

```
IO.fix_long2pos_headers(obsfiles)
```

is ONLY necessary if your observations were taken before June 10, 2015. It is safe to leave this line on for a second run: the script will not modify the same files twice.

Remember to update the `lambda_solution_wave_stack` file: you can update this in the variable `wavelength_file`, which will be used by the following instructions.

The driver contains instructions on how to perform background subtraction and finally rectification, in a similar way as for a normal mask.

The resulting files are the same as in the standard reduction, but the main results are contained in:

```
{object}_posA_narrow_{filter}_eps.fits
```

and

```
{object}_posC_narrow_{filter}_eps.fits
```

For the wide slits, since there is no AB pattern, we use the sky provided by one of the observations in the narrow slits, and we do not perform the final rectification.

In this case the final science results are contained in:

```
bsub_{object}_posC_wide_{filter}_A-B.fits
```

and

```
bsub_{object}_posA_wide_{filter}_A-B.fits
```

The longslit reductions require transferring the Longslit\_Driver.py file into the reduction directory. A few key parameters have to be adjusted in Longslit\_Driver.py to help the pipeline figure out where to extract the longslit from.

- `cd /path/to/LONGSLIT/`
- `cp ~/MosfireDRP-master/drivers/Longslit_driver.py .`
- Check all the .txt files to make sure your observations are included. You may have to merge files from various LONGSLIT\* directories. This happens when your observations use a shorter longslit than the calibrations.
- Note that mospy handle generates separate offset files for each of your targets, using the target name, but does NOT separate repeated observations of the same targets at different times of the night.
- edit `Driver_Longslit.py`
  - Examine a longslit image (see figure below) and adjust ‘yrange’: [709, 1350] to the vertical range covered by the slit
  - From the same examined longslit, select ‘row\_position’ so that it is uncontaminated by the spectrum. See Figure 1.
  - make sure that ‘mode’:‘longslit’ is specified in the longslit variable
  - The result should look like Figure 2.
- For each step in a section, uncomment the necessary line and run mosdrp on the Driver file. Once the `apply_lambda_simple` step is complete, fill in the ‘`lambda_solution_wave_stack...`’ line with the correct wave stack file.

You now have two options based on the results. If the night sky lines are not bright enough to identify in the interactive step you should use arclamps. In the following instructions, replace wavefiles with ‘Ne.txt’ or ‘Ar.txt’ and specify `neon=True` or `argon=True`.

An example of an uncontaminated row (#1127) in the longslit.

**Example of a modified Driver\_Longslit.py.** Notice that pixel 991 is selected as the row to perform the initial wavelength solution on. In Figure 2, this is the equivalent of 1127.

layout: page title: A Word About Header Comments permalink: /manual/headercomments —

Files produced by the DRP have a series of information in the FITS header that helps users determine the pedigree of files involved in the reduction. Since many files go into reductions, FITS headers are enormous and some documentation about them is useful.

The derived product FITS headers are organized as follows. The header of the first science file in the ‘A’ frame goes directly into the header. As the rest of the



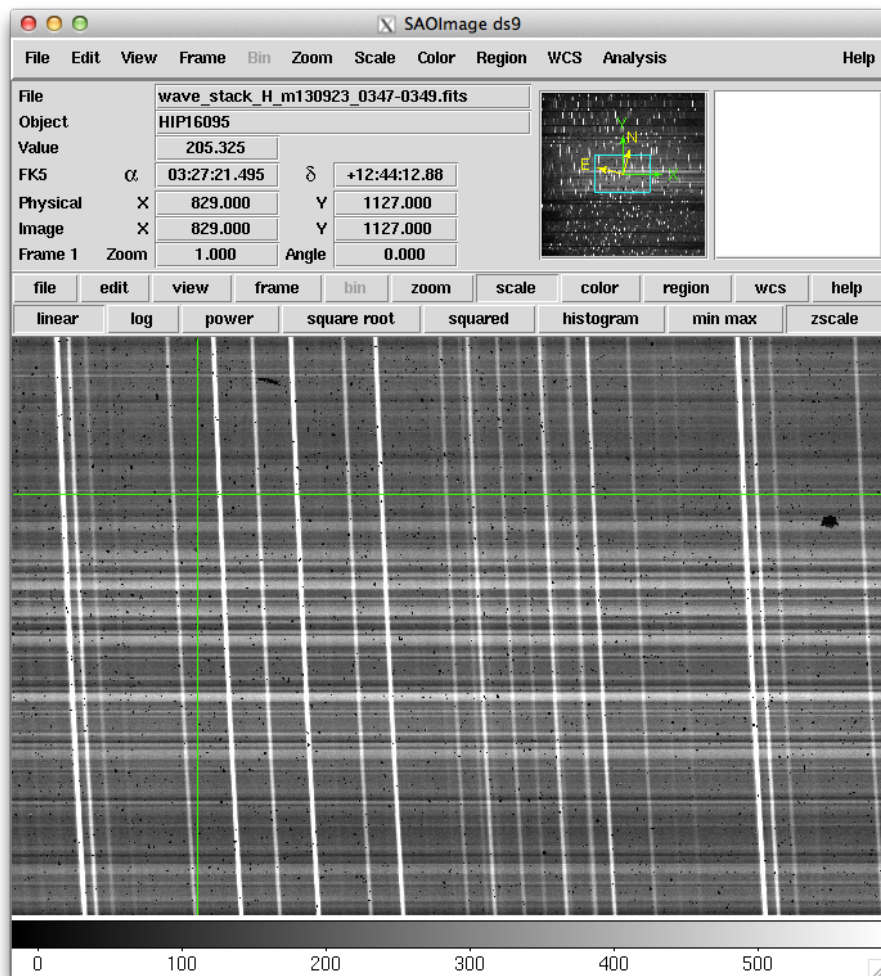
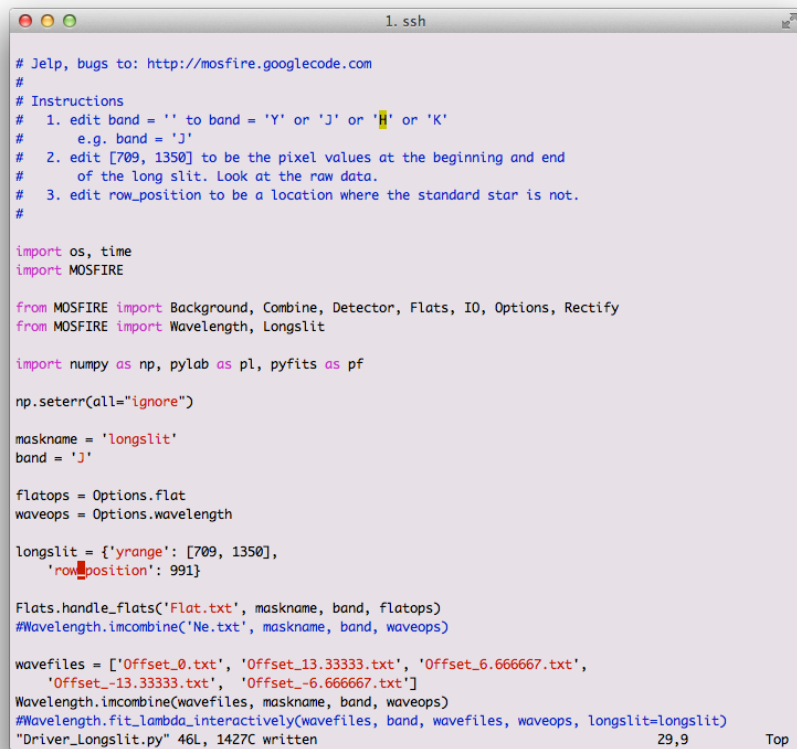


Figure 11: Screenshot



```
# Help, bugs to: http://mosfire.googlecode.com
#
# Instructions
# 1. edit band = '' to band = 'Y' or 'J' or 'H' or 'K'
#    e.g. band = 'J'
# 2. edit [709, 1350] to be the pixel values at the beginning and end
#    of the long slit. Look at the raw data.
# 3. edit row_position to be a location where the standard star is not.
#

import os, time
import MOSFIRE

from MOSFIRE import Background, Combine, Detector, Flats, IO, Options, Rectify
from MOSFIRE import Wavelength, Longslit

import numpy as np, pylab as pl, pyfits as pf

np.seterr(all="ignore")

maskname = 'longslit'
band = 'J'

flatops = Options.flat
waveops = Options.wavelength

longslit = {'yrange': [709, 1350],
            'row_position': 991}

Flats.handle_Flats('Flat.txt', maskname, band, flatops)
#Wavelength.imcombine('Ne.txt', maskname, band, waveops)

wavefiles = ['Offset_0.txt', 'Offset_13.33333.txt', 'Offset_6.66667.txt',
             'Offset_-13.33333.txt', 'Offset_-6.66667.txt']
Wavelength.imcombine(wavefiles, maskname, band, waveops)
#Wavelength.fit_lambda_interactively(wavefiles, band, waveops, longslit=longslit)
"Driver_Longslit.py" 46L, 1427C written                29,9      Top
```

Figure 12: Screenshot

‘A’ frames go into the header, the new keyword is checked against the current header. If the value of the keyword is different, a new keyword is added with the key postpended by `_img###` where `###` is the file number. A special keyword called `imfno###` is created showing the full path to the file in the data reduction set. An example is shown below:

```
CELOCAL = 0.00518363 / collimation elevation local (18.7 arcsec)
CALLOCAL = -0.00399414 / collimation azimuth local (-14.4 arcsec)
AZ = 108.89364809 / telescope azimuth (108.89 deg)
AXESTAT = 'tracking' / axes control status
AIRMASS = 1.05613137 / air mass (1.06)
UTC = '04:34:58.63' / coordinated universal time (h)
HIERARCH imfno0135 = '/scr2/mosfire/2013nov26/m131126_0135.fits' / img0135 file
HIERARCH imfno0137 = '/scr2/mosfire/2013nov26/m131126_0137.fits' / img0137 file
HIERARCH TIME-OBS_img137 = '04:42:07.515' / UT start time of exposure
HIERARCH TIME-END_img137 = '04:45:31.609' / UT end time of exposure (after last
HIERARCH DATAFILE_img137 = 'm131126_0137' / Datafile name
HIERARCH FRAMENUM_img137 = 137 / Frame number
HIERARCH REGTMP1_img137 = 76.99600220000001 / Temperature at channel A
HIERARCH OUTPTMP1_img137 = 10.10000038 / Ch A heater output percentage
HIERARCH REGTMP2_img137 = 120.01300049 / Temperature at channel B
HIERARCH OUTPTMP2_img137 = 29.10000038 / Ch B heater output percentage
HIERARCH DWRTMP1_img137 = 120.11799622 / Ch C1 temperature (K)
HIERARCH DWRTMP2_img137 = 120.28099823 / Ch C2 temperature (K)
HIERARCH DWRTMP3_img137 = 120.64299774 / Ch C3 temperature (K)
HIERARCH DWRTMP4_img137 = 120.09100342 / Ch C4 temperature (K)
HIERARCH DWRTMP5_img137 = 114.05599976 / Ch D1 temperature (K)
HIERARCH DWRTMP6_img137 = 113.72899628 / Ch D2 temperature (K)
HIERARCH DWRTMP7_img137 = 61.65299988 / Ch D3 temperature (K)
```

Figure 13: Screenshot

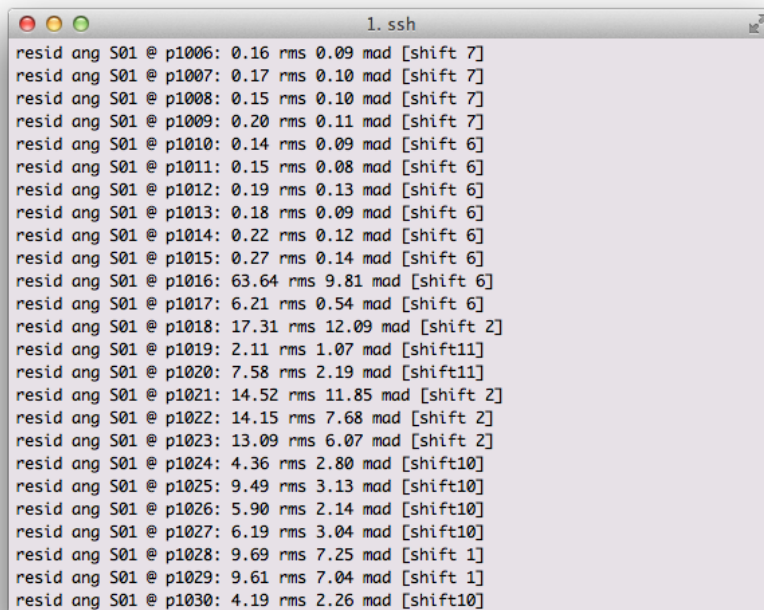
ds9 output of the FITS header. Note that the first “A” frame file is located in `/scr2/mosfire/2013nov26/m131126_0135.fits`. The second file (`#137`) has a similar path. The keywords which follow from file `#137` have different values than those in file `#135` and are thus named `KEY_img###`.

## Pay attention to the wavelength fitting output

The output above shows that up to pixel 1015 the RMS was 0.27 Angstrom level, and then dramatically jumped to 60 angstrom. Look at the image and examine pixel 1016, figure out what happened. You may have to adjust your input files or remove a file from the set.

## Look at rectified\_wave\_stack files

Look at `rectified_wave_stack*` files and make sure the night sky lines are vertical on the detector.



```
resid ang S01 @ p1006: 0.16 rms 0.09 mad [shift 7]
resid ang S01 @ p1007: 0.17 rms 0.10 mad [shift 7]
resid ang S01 @ p1008: 0.15 rms 0.10 mad [shift 7]
resid ang S01 @ p1009: 0.20 rms 0.11 mad [shift 7]
resid ang S01 @ p1010: 0.14 rms 0.09 mad [shift 6]
resid ang S01 @ p1011: 0.15 rms 0.08 mad [shift 6]
resid ang S01 @ p1012: 0.19 rms 0.13 mad [shift 6]
resid ang S01 @ p1013: 0.18 rms 0.09 mad [shift 6]
resid ang S01 @ p1014: 0.22 rms 0.12 mad [shift 6]
resid ang S01 @ p1015: 0.27 rms 0.14 mad [shift 6]
resid ang S01 @ p1016: 63.64 rms 9.81 mad [shift 6]
resid ang S01 @ p1017: 6.21 rms 0.54 mad [shift 6]
resid ang S01 @ p1018: 17.31 rms 12.09 mad [shift 2]
resid ang S01 @ p1019: 2.11 rms 1.07 mad [shift11]
resid ang S01 @ p1020: 7.58 rms 2.19 mad [shift11]
resid ang S01 @ p1021: 14.52 rms 11.85 mad [shift 2]
resid ang S01 @ p1022: 14.15 rms 7.68 mad [shift 2]
resid ang S01 @ p1023: 13.09 rms 6.07 mad [shift 2]
resid ang S01 @ p1024: 4.36 rms 2.80 mad [shift10]
resid ang S01 @ p1025: 9.49 rms 3.13 mad [shift10]
resid ang S01 @ p1026: 5.90 rms 2.14 mad [shift10]
resid ang S01 @ p1027: 6.19 rms 3.04 mad [shift10]
resid ang S01 @ p1028: 9.69 rms 7.25 mad [shift 1]
resid ang S01 @ p1029: 9.61 rms 7.04 mad [shift 1]
resid ang S01 @ p1030: 4.19 rms 2.26 mad [shift10]
```

Figure 14: Screenshot