mosviz Documentation

Release 0.0.2.dev429+g68fa8db

JDADF Developers

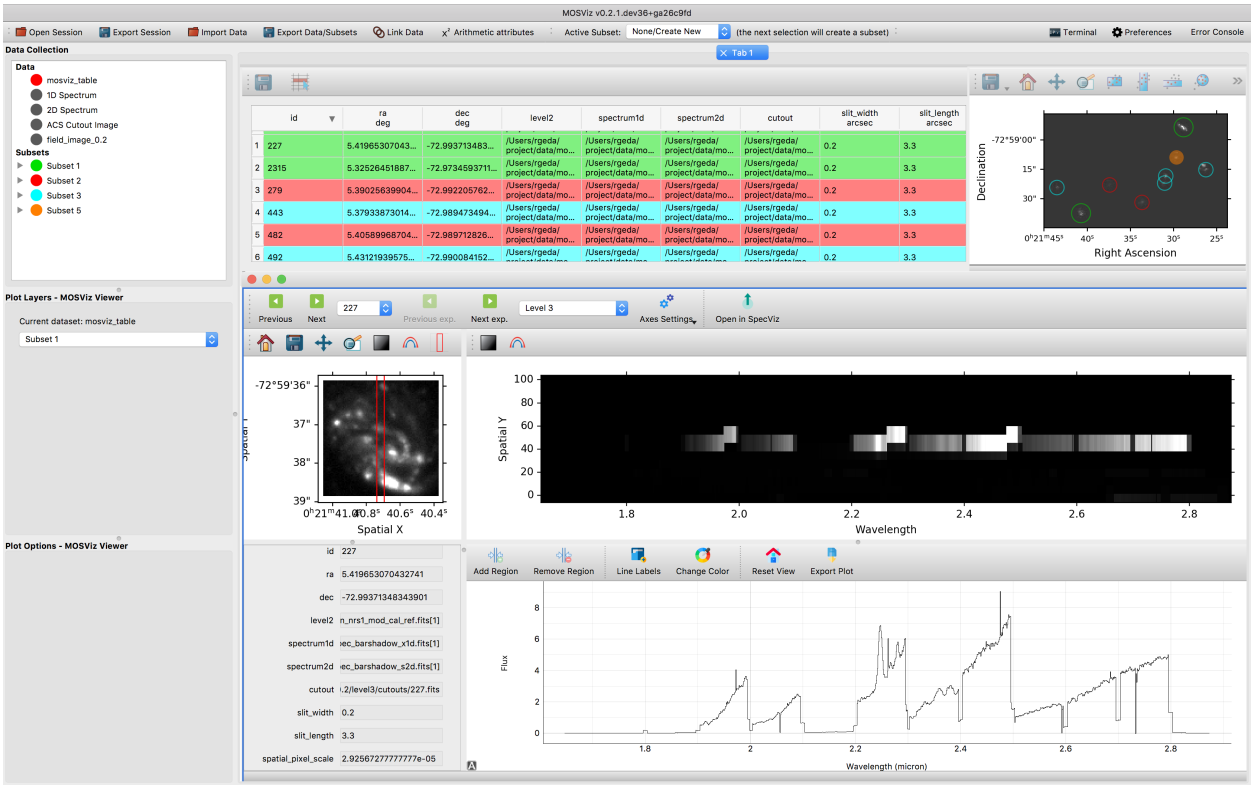
Nov 24, 2019

Contents

MOSViz is a quick-look analysis and visualization tool for multi-object spectroscopy (MOS). It is designed to work with pipeline output: spectra and associated images, or just with spectra. MOSViz is created to work with data from any telescope/instrument, but is built with the micro-shutter array (MSA) on the JWST/NIRSpec spectrograph and the JWST/NIRCam imager in mind. As such, MOSViz has some features specific to NIRSpec and NIRCam data.

The NIRSpec micro-shutter array (MSA) will produce ~100 spectra per pointing. Many users will perform surveys with the MSA that will result in data sets of thousands to tens of thousands of spectra. Especially in the early days of JWST, careful inspection of all of the data will be critical to achieving scientific goals, and it is important to make this task efficient. Inspection involves looking at the locations of astronomical sources within shutters, the location of background apertures in the observed field, the quality of the 2D spectra, and the quality of the 1D extracted spectra. It also often involves simple interactive measurements of quantities such as wavelengths, velocities, line fluxes, widths.

MOSViz is built using the PyQt programming language that allows for flexible and efficient development of a Graphical User Interface (GUI) using the Python programming language. MOSViz is incorporated into the Glue visualization tool (Beaumont, Robitaille, & Borkin 2014), and as such, takes advantage of Glue's "linked views," as described in Section 4. It uses the specutils core data object from Astropy (Astropy Collaboration 2013).



Part I

Installation

MOSViz is packaged and distributed through both Anaconda via the GlueViz channel, as well as through the Python Package Index (PyPI). It is also available from source.

CHAPTER 1

Install via Conda

In order to use the Anaconda package, you must first install an Anaconda distribution. If you do not have Anaconda, please follow the instructions here to install it.

Once you have Anaconda installed, all you have to do to install MOSViz is simply type the following at any Bash terminal prompt:

```
$ conda create -n <environment_name> -c glueviz mosviz
```

Next activate your MOSViz conda environment:

```
$ conda activate <environment_name>
```

To launch MOSViz now you enter:

```
$ mosviz
```

CHAPTER 2

Install via PyPI

To install MOSViz using PyPI type the following at any terminal prompt:

```
$ pip install mosviz
```

You will also need to install the PyQt package as well by typing the following:

```
$ pip install pyqt5
```

CHAPTER 3

Install via source

Warning: Using the Conda install is the preferred method for installing MOSViz to ensure that all dependencies are met. If you install via source/pip be aware that some dependencies will need to be installed manually!

MosViz can also be installed manually using the source code and requires the following dependencies to be installed on your system. Most of these will be handled automatically by the setup functions.

- Python 3
- Astropy
- Numpy
- Scipy
- Matplotlib
- Specutils
- Glue

At your terminal, you may either clone the repository directly and then install:

```
$ git clone https://github.com/spacetelescope/mosviz.git
$ cd mosviz
$ python setup.py install
```

Or, have the pip package manager do everything for you:

```
$ pip install git+https://github.com/spacetelescope/mosviz.git
```

Either way, the MosView Viewer should show up in the list of available Glue viewers.

Alternatively, you can use conda and install a nightly build using:

```
$ conda create -n mosviz-nightly -c glueviz/label/dev python=3.6 mosviz
```

CHAPTER 4

SpecViz functionality

MOSViz comes with the ability to open 1D spectra inside a SpecViz instance within Glue. However, this functionality will only be enabled if you also have SpecViz installed on your system.

If you are interested in this functionality, please follow the SpecViz installation instructions.

Part II

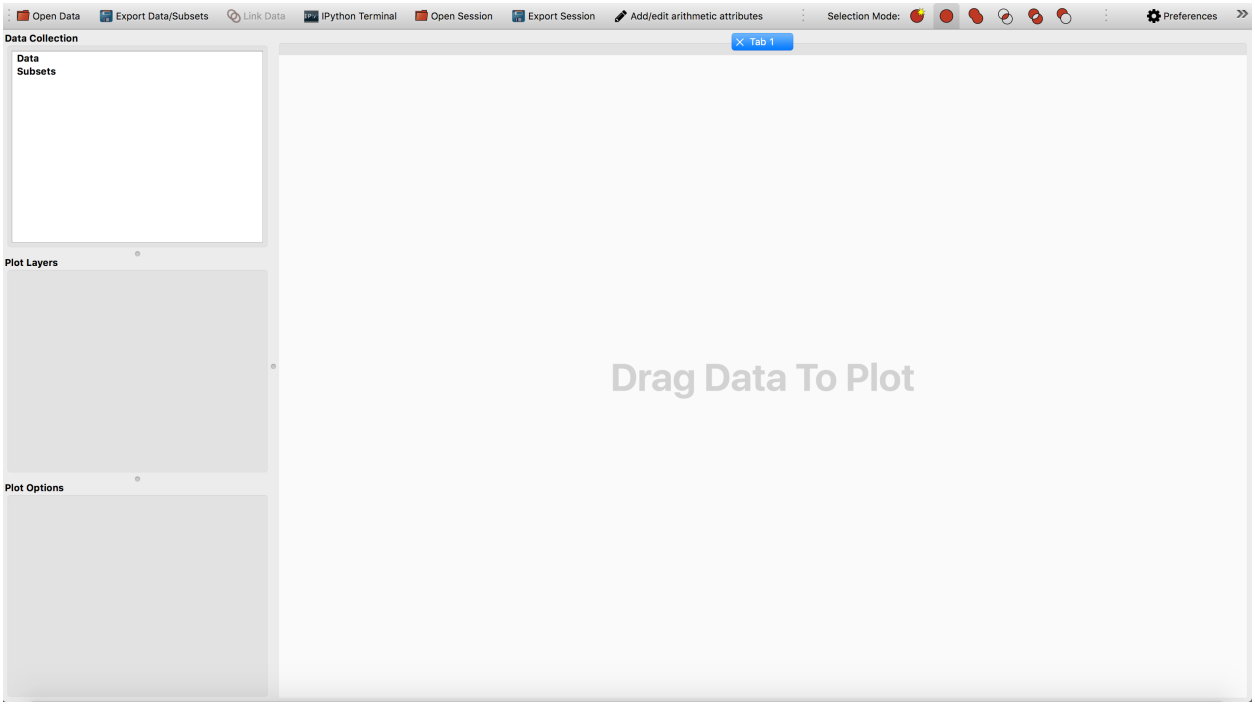
Getting Started with MOSViz

This page will walk you through the basics of the MOSViz GUI. The data we will use for this tutorial is located here:

- Click to download data: deimos.zip

Once you have MOSViz installed you can launch the GUI by typing:

```
$ mosviz
```



CHAPTER 5

Opening Data

There are many ways to open a dataset with MOSViz:

- By selecting the **Open Data Set** item under the **File** menu or using the equivalent shortcut (e.g. **Ctrl+O** on Linux, **Cmd+O** on Mac).
- Dragging and dropping data files on the main window.
- Through the terminal `$ mosviz deimos_mosviz.tbl`

Find and open `deimos_mosviz.tbl` which should be in the `deimos.zip` file you downloaded above. We will use a dataset from the Deimos instrument at the Keck Observatory. The main components of a MOSViz dataset are:

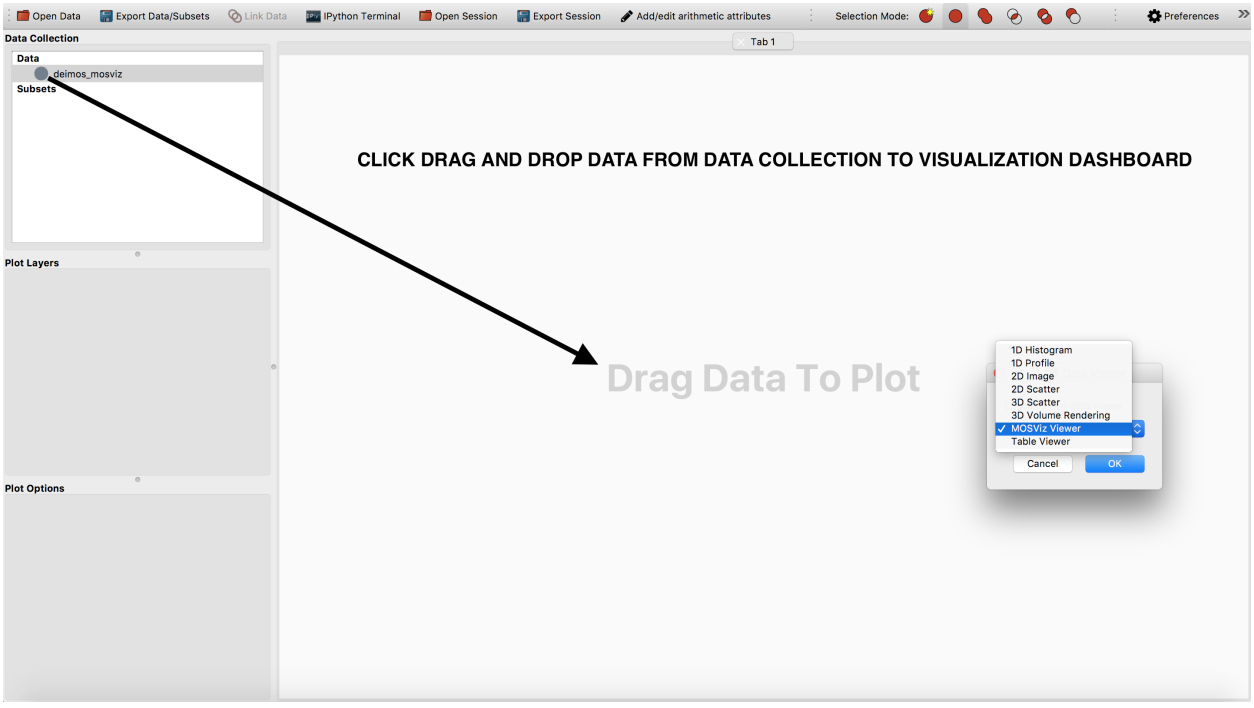
- A catalog of targets in tabular form
- A 2D image cutout
- A 1D spectrum
- A 2D spectrum

Note: MOSViz data tables are in the ECSV (Enhanced Character Separated Values) format. For more detail on ECSV tables go here.

CHAPTER 6

Visualizing Data

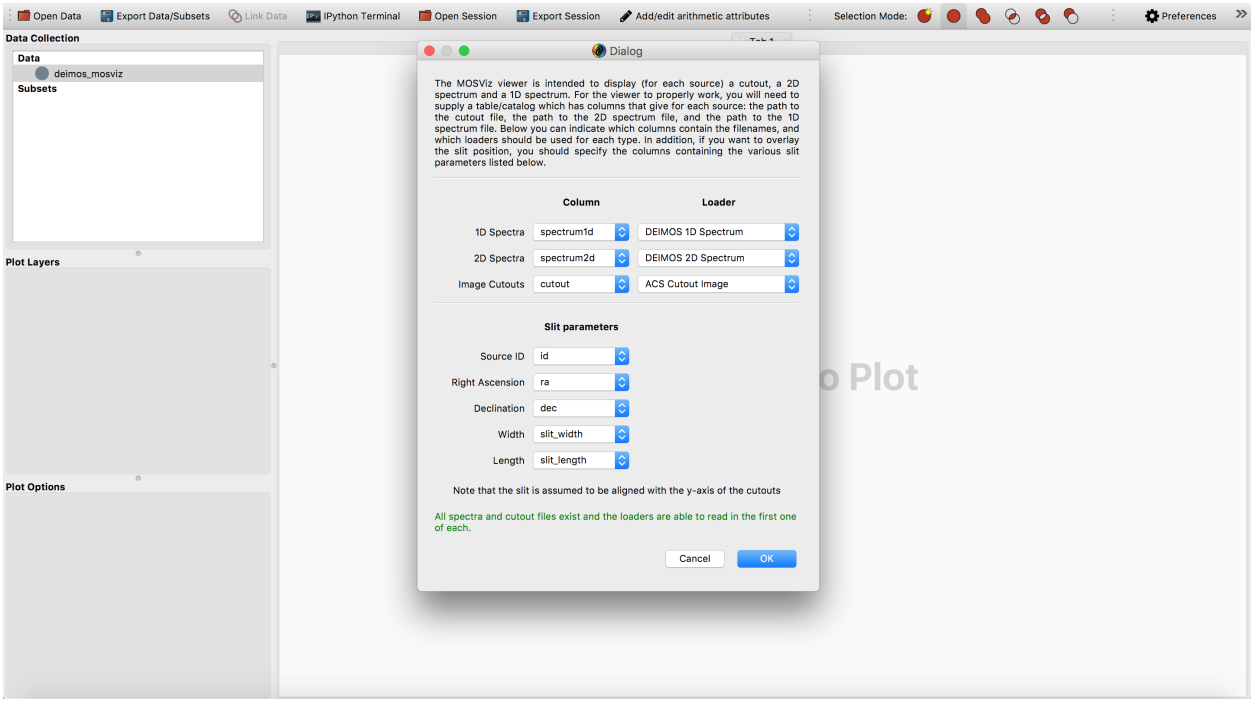
When the dataset is loaded in the data collection, drag and drop the dataset from the data collection to the visualization dashboard. A dialog will appear asking you to select a data viewer, select the **MOSViz viewer**.



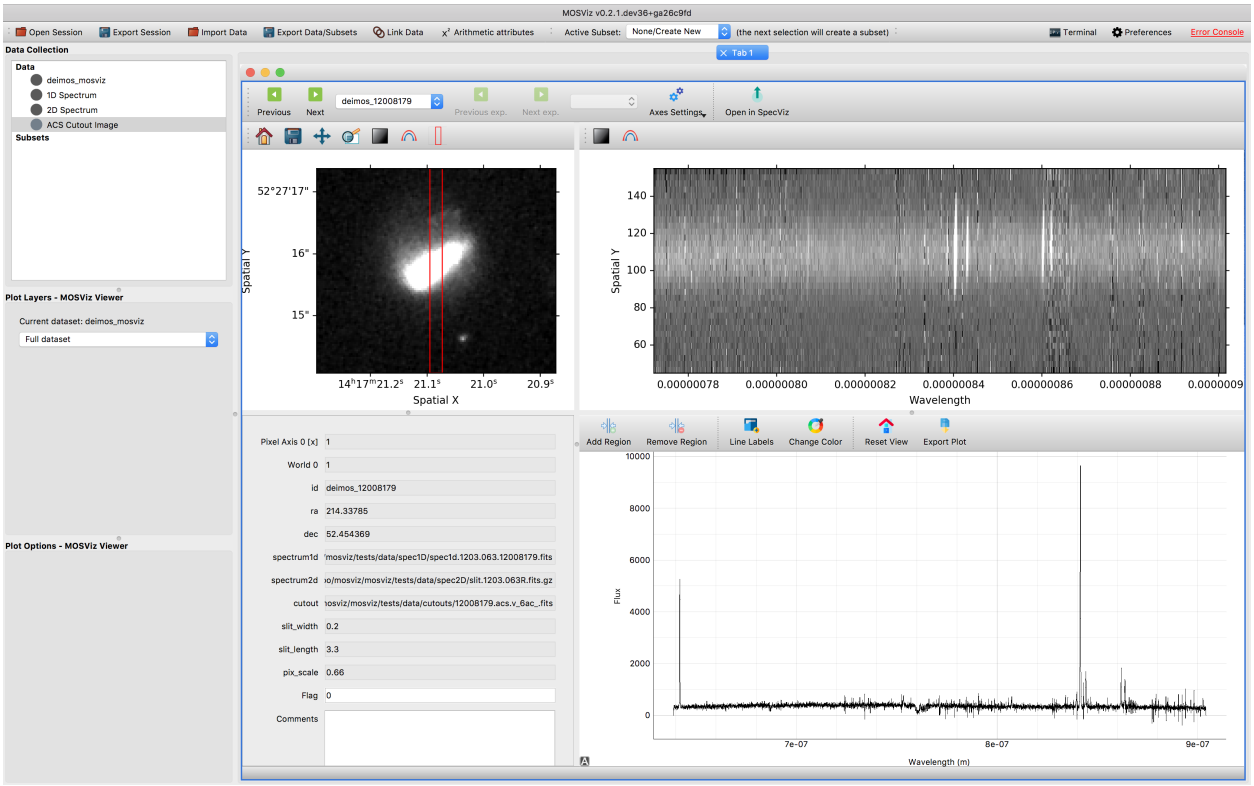
Next you will be prompted with another dialog asking you to specify the data readers for each component of your dataset. This dialog will also contains options to overlay slit positions. For this example we are going to use the default settings.

Note: If you open your MOSViz data table via the command line, you will immediately be prompted with the diaglog

below. You won't have to click drag and drop the dataset from the data collection to the visualization dashboard.



Once your configuration is set, click OK. Now the visualization dashboard should contain a MOSViz viewer with views of all of the different dataset components.



To view the other spectra and cutouts from the data table, use the Next/Previous buttons or select by name using the

drop down located in the upper left hand corner of the MOSViz Viewer.

Part III

Overview of Basic MOSViz Functionality

MOSViz is composed of separate components that interact via "linked views." As described below, and as shown in Figures 1, MOSViz includes a:

- 1D spectral viewer
- 2D spectral viewer
- Image cutout viewer
- Data table
- 2D scatter plot
- 3D scatter plot, and a
- mosaic viewer where the targets are indicated.

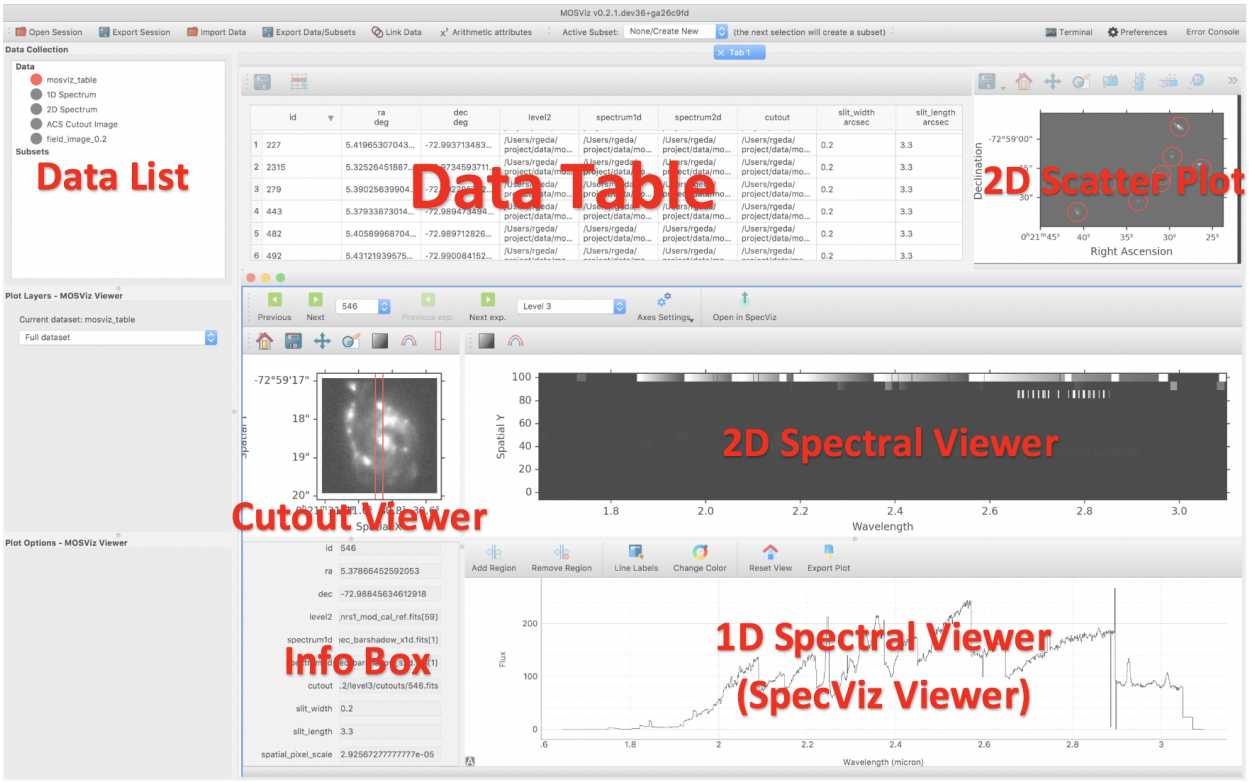


Fig. 1: Figure 1: A screen shot of MOSViz with its components labeled is shown. MOSViz is embedded in the Glue visualization tool. In the use case shown, the user has selected targets in the scatter plot on the upper right. The corresponding rows in the table are highlighted, and the first object's 1D and 2D spectra are shown. The cutout image has the spectral aperture indicated on it in red. The user advances to the next target in the spectra/image viewers by hitting the arrow key in the upper right. Shown in the MOS viewer is a simulated NIRCam image and an "engineering" NIRSpec MSA spectrum with faked errors.

Each of these components is described in more detail below. Once a MOS data set is read in, users can use any combination of these tools. A typical use case/workflow is the following:

- Read in a table of: target identifications, coordinates (RA, DEC), paths and file names of cutout images, paths and file names of 2D spectra, paths and file names of 1D spectra, and optional quantities (such as star-formation rate magnitudes, stellar masses, sizes, etc.) for 1000 galaxies at a redshift of 2
- Sort galaxies in table according to star-formation rate
- Highlight the 3 galaxies with the highest star-formation rates

- This causes the 1D and 2D spectra and cut-out image of the first galaxy of these three galaxies to display
- Measure the equivalent width of an emission line
- Hit the arrow button to move to the next galaxy
- Do the same for the second and third objects

MOSViz's main purpose is to help users to visually inspect and perform a "first-look" analysis of multiple spectra. To this end, MOSViz's main component is its 1D and 2D spectral viewers. The user can choose to use both 1D and 2D viewers, or either one. Users can interact further with the 1D spectra by opening the SpecViz 1D spectral analysis tool. SpecViz allows users to make measurements of e.g, emission and absorption line strengths and sky continuum levels. For the 2D spectra, a basic set of view adjustments will is available: zoom, pan, colormap, and intensity scaling.

MOSViz also includes an optional cutout viewer to show an image of each target alongside their 1D and 2D spectra. The cutout is aligned so that the cross-dispersion direction has the same vertical orientation as for the 2D spectrum. We provide documentation on how to create such cutouts for single-waveband images and multi-band color images. For the single-waveband cutouts, a basic set of view adjustments will be available in future versions of MOSViz: zoom, pan, colormap, and intensity scaling. The UI will read back the row, column and flux under in the pixel under the cursor. In addition, we provide documentation on how to show overlayed spectral apertures on each cutout image (at above link). For most MOS's, the spectral aperture is a thin slit. For the NIRSpec MSA, it is a single or multi-shutter "slitlet".

MOSViz's default is to lock the wavelength axis of the 1D and 2D spectra so that they are aligned spatially and zoom and pan together. It also defaults to have the cross-dispersion spatial axis locked to the cutout vertical axis so that they zoom and pan together. All of the data must have valid WCSs for this to work.

The data table allows users to select targets to display in the spectral and/or cutout viewers. Optionally, they can also display the selected targets in the 2D and/or 3D scatter plots and/or on the large image viewer (the latter available in future versionsof MOSViz). Columns in the data table must include at minimum target identification (id), right ascension (RA), and declination (DEC). The table can include any other quantities the user wishes to include. Its columns can be sorted. The user can select as many or as few objects as they wish to view.

The optional 2D and 3D scatter plots can be used to plot the quantities in the tables. The user can click on each point/target in the plots to display the selected target's spectra and images. The user can select multiple contiguous points/targets at once.

Finally, the information box displays the information from the table on the specific target displayed.

Part IV

Reading-in Data

MOSViz will be able to read in NIRSpec MSA data straight from the pipeline or archive. This will include 1D and 2D spectra, images, data tables, and associated images. This page documents how to read in MOS data from another observatory. An example is shown at the bottom (*Defining custom readers for spectra and cutouts*).

The general workflow can be summarized as follows:

1. Start Glue by typing `glue` in the command line
2. In Glue, click on the folder icon and open the file containing the Data Table for your data set.
3. Drag the file from the Data Collection window into the main canvas.
4. Select 'MOSViz Viewer' from the pop-up menu.
5. Use the green arrow buttons at the top of the viewer to switch between objects.

CHAPTER 7

Overview

MOSViz requires at minimum the following input:

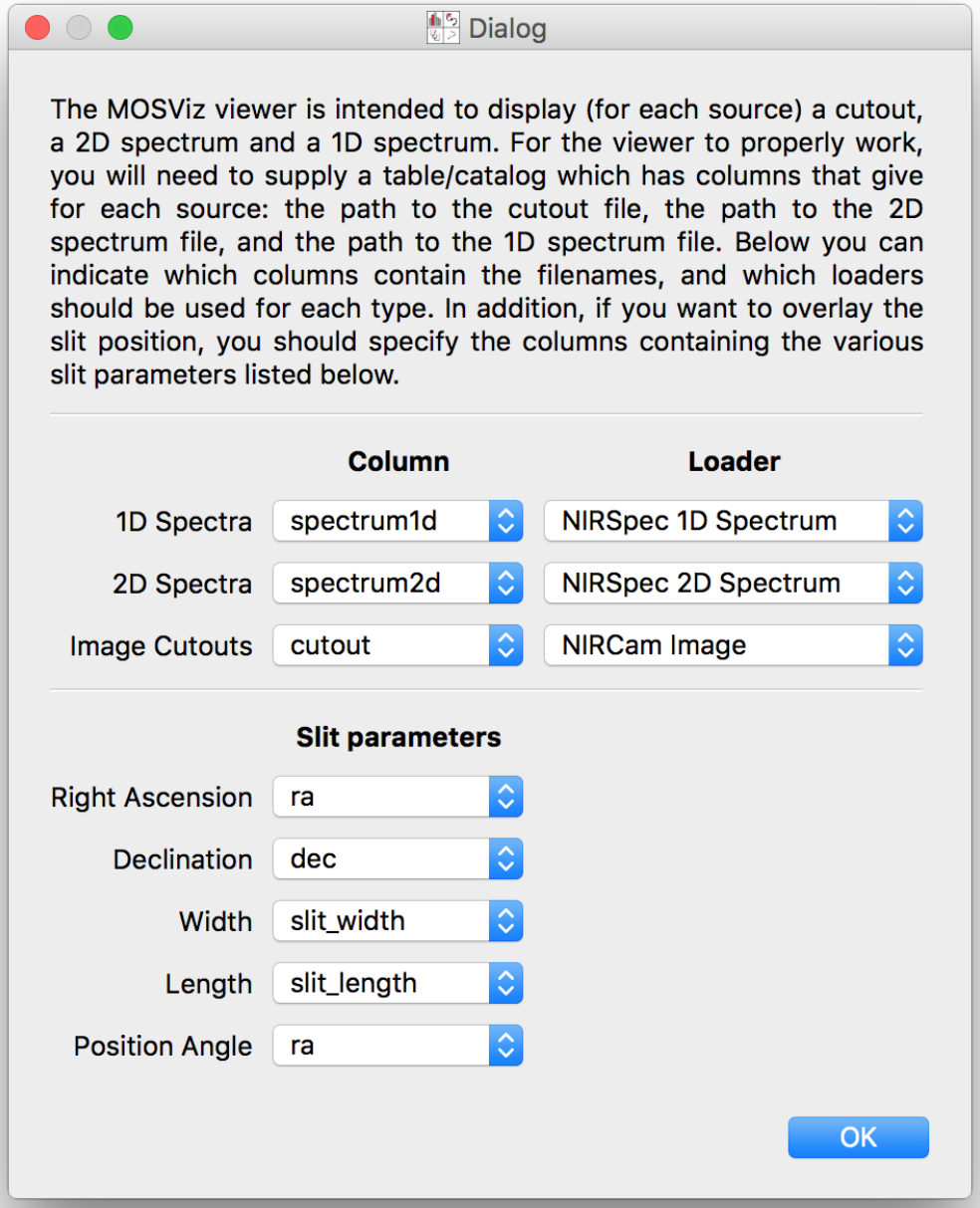
- A catalog of targets in tabular form
- A 1D and 2D spectrum for each target
- A cutout image for each target

The main data file that will be read in and used by the MOSViz viewer is the catalog. The catalog can be in any format that Glue understands (such as a FITS table, VO table, ASCII table, and so on). The main constraint is that the table should include columns with the following information for each target:

Column	Default name	Units
File path to the 1D spectrum	`spectrum1d`	
File path to the 2D spectrum	`spectrum2d`	
File path to the image cutout	`cutout`	
Right ascension of the slit center (FK5 J2000)	`ra`	degrees
Declination of the slit center (FK5 J2000)	`dec`	degrees
Width of the slit	`slit_width`	arcseconds
Length of the slit	`slit_length`	arcseconds

The columns can have any name, but if they are given the name in the **Default name** column above, they will be automatically recognized. In addition, the catalog can contain any other useful information such as source names, magnitudes, stellar masses, star-formation rates, etc. These are not required, but including them will allow selections to be done on those parameters.

When adding a dataset to the MOSViz viewer for the first time in glue, the following dialog will appear:



This dialog asks to confirm the column names and also asks to select the appropriate readers for the 1D and 2D spectra as well as the cutouts. In *Defining custom readers for spectra and cutouts* we show how to define custom readers for spectra and cutouts in case there isn't an appropriate built-in reader available, and in *Writing the Data Table* we show how to tell MOSViz which readers/columns to use by default.

CHAPTER 8

Defining custom readers for spectra and cutouts

This example uses spectra and cutout images of galaxies at 0.2<z<1.2 which have spectra from the DEIMOS multi-object spectrograph on Keck and color cut-out images already created from Hubble/ACS. These galaxies are from the DEEP2 Survey. The spectra can be obtained from the DEEP2 Data Release page.

MOSViz is a Glue plugin so we'll write our own Custom Data Loaders to read data. All we need to do is write a function which takes a filename as input and returns a glue `Data` object. For this example, we have three different types of data: a 1D Spectrum, a 2D spectrum, and a cutout image.

8.1 1D Spectrum Reader

MOSViz expects the 1D Spectrum Data object to have three components: Wavelength, Flux, and Uncertainty. The final reader we are going to write is as follows:

```
from astropy.io import fits
from glue.core import Data
from mosviz.loaders.mos_loaders import mosviz_spectrum1d_loader

@mosviz_spectrum1d_loader('DEIMOS 1D Spectrum')
def deimos_spectrum1D_reader(filename):
    """
    Data loader for Keck/DEIMOS 1D spectra.

    This loads the 'Bxspf-B' (extension 1) and 'Bxspf-R' (extension 2) and
    appends them together to proudce the combined Red/Blue Spectrum along
    with their Wavelength and Inverse Variance arrays.
    """

    hdulist = fits.open(filename)
    data = Data(label='1D Spectrum')
    data.header = hdulist[1].header

    full_wl = np.append(hdulist[1].data['LAMBDA'][0], hdulist[2].data['LAMBDA'][0])
```

(continues on next page)

(continued from previous page)

```
    full_spec = np.append(hdulist[1].data['SPEC'][0], hdulist[2].data['SPEC'][0])
    full_ivar = np.append(hdulist[1].data['IVAR'][0], hdulist[2].data['IVAR'][0])

    data.add_component(full_wl, 'Wavelength')
    data.add_component(full_spec, 'Flux')
    data.add_component(1/np.sqrt(full_ivar), 'Uncertainty')

    return data
```

Let's take a look at how to write this step by step. We first take a look at the contents of our example FITS file to see which parts we need to pass to MOSViz:

```
>>> from astropy.io import fits
>>> hdulist = fits.open('spec1d.1355.134.13040873.fits')
>>> hdulist.info()
Filename: spec1d.1355.134.13040873.fits
No.    Name         Type      Cards   Dimensions   Format
  0  PRIMARY     PrimaryHDU       4   ()
  1  Bxspf-B     BinTableHDU    131   1R x 15C   [4096E, 4096E, 4096E, 4096I, 4096I, 4096I, 4096I,␣
↪4096I, E, E, E, J, J, 4096E, E]
  2  Bxspf-R     BinTableHDU    131   1R x 15C   [4096E, 4096E, 4096E, 4096I, 4096I, 4096I, 4096I,␣
↪4096I, E, E, E, J, J, 4096E, E]
  3  Horne-B     BinTableHDU    140   1R x 15C   [4096E, 4096E, 4096E, 4096I, 4096I, 4096I, 4096I,␣
↪4096I, E, E, E, J, J, 4096E, E]
  4  Horne-R     BinTableHDU    140   1R x 15C   [4096E, 4096E, 4096E, 4096I, 4096I, 4096I, 4096I,␣
↪4096I, E, E, E, J, J, 4096E, E]
  5  Bxspf-NL-B  BinTableHDU    131   1R x 15C   [4096E, 4096E, 4096E, 4096I, 4096I, 4096I, 4096I,␣
↪4096I, E, E, E, J, J, 4096E, E]
  6  Bxspf-NL-R  BinTableHDU    131   1R x 15C   [4096E, 4096E, 4096E, 4096I, 4096I, 4096I, 4096I,␣
↪4096I, E, E, E, J, J, 4096E, E]
  7  Horne-NL-B  BinTableHDU    140   1R x 15C   [4096E, 4096E, 4096E, 4096I, 4096I, 4096I, 4096I,␣
↪4096I, E, E, E, J, J, 4096E, E]
  8  Horne-NL-R  BinTableHDU    140   1R x 15C   [4096E, 4096E, 4096E, 4096I, 4096I, 4096I, 4096I,␣
↪4096I, E, E, E, J, J, 4096E, E]
```

The file contains pairs of red and blue spectra which have been filtered in various ways. For the sake of this example we'll choose the `Bxspf` spectra. Let's take a closer look at the relevant extension:

```
>>> hdulist['Bxspf-R'].columns
ColDefs(
    name = 'SPEC'; format = '4096E'
    name = 'LAMBDA'; format = '4096E'
    name = 'IVAR'; format = '4096E'
    name = 'CRMASK'; format = '4096I'
    name = 'BITMASK'; format = '4096I'
    name = 'ORMASK'; format = '4096I'
    name = 'NBADPIX'; format = '4096I'
    name = 'INFOMASK'; format = '4096I'
    name = 'OBJPOS'; format = 'E'
    name = 'FWHM'; format = 'E'
    name = 'NSIGMA'; format = 'E'
    name = 'R1'; format = 'J'
    name = 'R2'; format = 'J'
    name = 'SKYSPEC'; format = '4096E'
    name = 'IVARFUDGE'; format = 'E'
)
```

(continues on next page)

(continued from previous page)

```
>>> hdulist['Bxspf-B'].columns
ColDefs(
    name = 'SPEC'; format = '4096E'
    name = 'LAMBDA'; format = '4096E'
    name = 'IVAR'; format = '4096E'
    name = 'CRMASK'; format = '4096I'
    name = 'BITMASK'; format = '4096I'
    name = 'ORMASK'; format = '4096I'
    name = 'NBADPIX'; format = '4096I'
    name = 'INFOMASK'; format = '4096I'
    name = 'OBJPOS'; format = 'E'
    name = 'FWHM'; format = 'E'
    name = 'NSIGMA'; format = 'E'
    name = 'R1'; format = 'J'
    name = 'R2'; format = 'J'
    name = 'SKYSPEC'; format = '4096E'
    name = 'IVARFUDGE'; format = 'E'
)
```

Again, there are a lot of options but for MOSViz we're only interested in three columns: `SPEC`, `LAMBDA`, `IVAR`. Further, MOSViz expects each of the arrays to be 1 dimensional and of the same size:

```
>>> hdulist['Bxspf-R'].data['SPEC'].shape
(1, 4096)
>>> hdulist['Bxspf-R'].data['LAMBDA'].shape
(1, 4096)
>>> hdulist['Bxspf-R'].data['IVAR'].shape
(1, 4096)
```

All of our arrays are the same size but they are stored in 2 dimensional arrays (with the first axis of size 1). So we'll just take the first (and only) element.

Now that we know what data we want from our FITS files let's look at how to write the data loader function. The basic structure for a data loader for a 1D spectrum is:

```
from glue.core import Data
from mosviz.loaders.mos_loaders import mosviz_spectrum1d_loader

@mosviz_spectrum1d_loader('DEIMOS 1D Spectrum')
def deimos_spectrum1D_reader(filename):
    # code to read in data here
    return data
```

`'DEIMOS 1D Spectrum'` is the label which is how we will identify this loader. For users familiar with defining glue data factories, `@mosviz_spectrum1d_loader` is equivalent to `@data_factory` but additionaly tells MOSViz that the loader is specifically for a 1D spectrum.

Let's now focus on what is needed inside the function. The function itself takes a filename to open as its only argument, so we open the file and instantiate a Glue `Data` object:

```
hdulist = fits.open(filename)
data = Data(label='1D Spectrum')
```

Now as above we're going to open the FITS file. We add the header from the FITS file to the data object:

```
data.header = hdulist[1].header
```

As stated above, MOSViz expects the Wavelength, Flux, and Uncertainty to be each be a single 1D array. We saw that the red and blue ends of the spectrum are stored in different extensions and that there are stored as 2D arrays. We take the first component of the each of the red and blue ends of the spectrum and combine them together. Then we take the full 1D array for each component and pass them to the ~glue.core.data.Data object using the add_component() method:

```
full_wl = np.append(hdulist[1].data['LAMBDA'][0], hdulist[2].data['LAMBDA'][0])
full_spec = np.append(hdulist[1].data['SPEC'][0], hdulist[2].data['SPEC'][0])
full_ivar = np.append(hdulist[1].data['IVAR'][0], hdulist[2].data['IVAR'][0])

data.add_component(full_wl, 'Wavelength')
data.add_component(full_spec, 'Flux')
data.add_component(1/np.sqrt(full_ivar), 'Uncertainty')

return data
```

8.2 2D Spectrum Reader

The basic structure for the 2D spectrum reader is similar to that for the 1D spectrum reader:

```
from astropy.io import fits
from glue.core import Data
from mosviz.loaders.mos_loaders import mosviz_spectrum1d_loader

@mosviz_spectrum2d_loader('DEIMOS 2D Spectrum')
def deimos_spectrum2D_reader(filename):
    """
    Data loader for Keck/DEIMOS 2D spectra.

    This loads only the Flux and Inverse variance. Wavelength information
    comes from the WCS.
    """

    hdulist = fits.open(filename)
    data = Data(label='2D Spectrum')
    data.coords = coordinates_from_header(hdulist[1].header)
    data.header = hdulist[1].header
    data.add_component(hdulist[1].data['FLUX'][0], 'Flux')
    data.add_component(1/np.sqrt(hdulist[1].data['IVAR'][0]), 'Uncertainty')
    return data
```

MOSViz expects the 2D Spectrum Data object to have two components: Flux and Uncertainty. Since a 2D spectrum is an image it also expects a World Coordinate System (WCS) which tells it how to transform from pixels to Wavelength. Let's take a look at the contents of our example FITS file to see which parts we need to pass to MOSViz:

```
>>> from astropy.io import fits
>>> hdulist = fits.open('slit.1355.134B.fits.gz')
>>> hdulist.info()
Filename: slit.1153.147B.fits.gz
No.    Name         Type      Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU       4   ()
1    slit        BinTableHDU    106   1R x 11C     [241664E, 241664E, 241664B, 241664B, 4096E, 241664E,␣
↪6D, 3D, 59E, 177E, 241664J]
2    slit        BinTableHDU     98   531R x 5C    [E, E, E, E, B]
>>> hdulist[1].data.columns
```

(continues on next page)

(continued from previous page)

```
ColDefs(
    name = 'FLUX'; format = '241664E'; dim = '( 4096, 59)'
    name = 'IVAR'; format = '241664E'; dim = '( 4096, 59)'
    name = 'MASK'; format = '241664B'; dim = '( 4096, 59)'
    name = 'CRMASK'; format = '241664B'; dim = '( 4096, 59)'
    name = 'LAMBDA0'; format = '4096E'
    name = 'DLAMBDA'; format = '241664E'; dim = '( 4096, 59)'
    name = 'LAMBDAX'; format = '6D'
    name = 'TILTX'; format = '3D'
    name = 'SLITFN'; format = '59E'
    name = 'DLAM'; format = '177E'; dim = '( 59, 3)'
    name = 'INFOMASK'; format = '241664J'; dim = '( 4096, 59)'
)
>>> hdulist[2].data.columns
ColDefs(
    name = 'AMP'; format = 'E'
    name = 'CEN'; format = 'E'
    name = 'SIG'; format = 'E'
    name = 'BASE'; format = 'E'
    name = 'MASK'; format = 'B'
)
```

MOSViz needs Flux and Uncertainty so the relevant columns are `FLUX` and `IVAR` in the the first `slit` extension:

```
>>> hdulist[1].data['FLUX'].shape
(1, 59, 4096)
>>> hdulist[1].data['IVAR'].shape
(1, 59, 4096)
>>>
```

All of our arrays are the same size but they are stored in 3 dimensional arrays (with the first axis of size 1). So we'll just take the first (and only) element which will give a 2D array.

We also need a WCS which should be in the header of the same extension as the data:

```
>>> from astropy.wcs import WCS
>>> WCS(hdulist[1].header)

Number of WCS axes: 2
CTYPE : 'LAMBDA'  'LAMBDA'
CRVAL : 6450.6538154  0.0
CRPIX : 0.0  0.0
CD1_1 CD1_2  : 0.32103118300400002  0.0
CD2_1 CD2_2  : 0.0  1.0
NAXIS    : 4367352 1
```

The WCS is here; however, the two axes both have name 'LAMBDA' and if we look at look at the second coordinate we can see that it isn't actually transformed. Glue expects that all of a `Data` object's components (including WCS axes) have unique names. We can take care of this easily in the data loader function.

Now that we know what data we want from our FITS files let's look at how to write the data loader function. As before, we use the following decorator to tell glue that this is a data loader, and MOSViz that it can read in 2D spectra:

```
@mosviz_spectrum2d_loader('DEIMOS 2D Spectrum')
def deimos_spectrum2D_reader(filename):
```

The function itself takes a filename to open as its only argument. We open the data file and instantiate a `Data` object:

```
hdulist = fits.open(filename)
data = Data(label='2D Spectrum')
```

As we noted above, the WCS axes should have different names. Since the second axis is not transformed we'll just change the header keyword which specifies its name to 'Spatial Y' Then we set the `coords` attribute of the `Data` object with `glue.core.coordinates.coordinates_from_wcs()`. We also pass the FITS header to the data so that useful information can be displayed in the MOSViz:

```
hdulist[1].header['CTYPE2'] = 'Spatial Y'
data.coords = coordinates_from_wcs(WCS(hdulist[1].header))
data.header = hdulist[1].header
```

As stated above, MOSViz expects the Flux and Uncertainty to be each be a single 2D array. We take the first component of each array (a 2D array) pass them to the `~glue.core.data.Data` object using the `add_component()` method:

```
data.add_component(hdulist[1].data['FLUX'][0], 'Flux')
data.add_component(1/np.sqrt(hdulist[1].data['IVAR'][0]), 'Uncertainty')

return data
```

8.3 Cutout Image Reader

Finally, the custom reader for the image cutouts looks like:

```
from astropy.io import fits
from glue.core import Data
from mosviz.loaders.mos_loaders import mosviz_cutout_loader

@mosviz_cutout_loader('ACS Cutout Image')
def acs_cutout_image_reader(filename):
    """
    Data loader for the ACS cut-outs for the DEIMOS spectra.

    The cutouts contain only the image.
    """

    hdulist = fits.open(filename)
    data = Data(label='ACS Cutout Image')
    data.coords = coordinates_from_header(hdulist[0].header)
    data.header = hdulist[0].header
    data.add_component(hdulist[0].data, 'Flux')

    return data
```

MOSViz expects the Cutout Image Data object to have one component: Flux. Since it is an image it also expects a World Coordinate System (WCS) which tells it how to transform from pixels to sky coordinates. Let's take a look at the contents of our example FITS file to see which parts we need to pass to MOSViz.

```
>>> from astropy.io import fits
>>> hdulist = fits.open('12020821.acs.i_6ac_.fits')
>>> hdulist.info()
Filename: 12020821.acs.i_6ac_.fits
No.    Name         Type      Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU      71   (201, 201)   float32
```

(continues on next page)

(continued from previous page)

```
>>> hdulist[0].data.shape
(201, 201)
```

There is only one extensions and the data in it is the cutout image (a 2D array). We also need a WCS which should be in the header of the same extension as the data:

```
>>> from astropy.wcs import WCS
>>> WCS(hdulist[0].header)
WCS Keywords

Number of WCS axes: 2
CTYPE : 'RA---TAN'  'DEC--TAN'
CRVAL : 214.40388488799999  52.630077362100003
CRPIX : 101.70472905800101  100.94206076200101
CD1_1 CD1_2  : -8.3333331279300006e-06  -4.5781947460699999e-14
CD2_1 CD2_2  : -4.5781947460699999e-14  8.3333331279300006e-06
NAXIS    : 201 201
```

The WCS looks as we would expect. Now that we know what data we want from our FITS files let's look at how to write the data loader function. We use the following decorator on the function to tell glue that this is a data factory and to tell MOSViz that it can handle cutout images:

```
@mosviz_cutout_loader('ACS Cutout Image')
def acs_cutout_image(filename):
```

The function itself takes a filename to open as its only argument. We open the data file and instantiate a `Data` object:

```
hdulist = fits.open(filename)
data = Data(label='Cutout Image')
```

We set the `coords` attribute of the `Data` object with `glue.core.coordinates.coordinates_from_wcs()`. We also pass the FITS header to the data so that useful information can be displayed in the MOSViz:

```
data.coords = coordinates_from_wcs(WCS(hdulist[0].header))
data.header = hdulist[0].header
```

We take the data in first extension data array (a 2D array) and pass it to the `~glue.core.data.Data` object using the `add_component()` method:

```
data.add_component(hdulist[0].data, 'Flux')

return data
```

8.4 Summary

The full contents of the ~/.glue/config.py is shown below:

```
import numpy as np

from astropy.io import fits
from astropy.wcs import WCS

from glue.core import Data
```

(continues on next page)

(continued from previous page)

```
from glue.core.coordinates import coordinates_from_header, coordinates_from_wcs

from mosviz.loaders.mos_loaders import (mosviz_spectrum1d_loader,
                                        mosviz_spectrum2d_loader,
                                        mosviz_cutout_loader)


@mosviz_spectrum1d_loader('DEIMOS 1D Spectrum')
def deimos_spectrum1D_reader(filename):
    """
    Data loader for Keck/DEIMOS 1D spectra.

    This loads the 'Bxspf-B' (extension 1)
    and 'Bxspf-R' (extension 2) and appends them
    together to proudce the combined Red/Blue Spectrum
    along with their Wavelength and Inverse Variance
    arrays.
    """

    hdulist = fits.open(filename)
    data = Data(label='1D Spectrum')
    data.header = hdulist[1].header

    full_wl = np.append(hdulist[1].data['LAMBDA'][0], hdulist[2].data['LAMBDA'][0])
    full_spec = np.append(hdulist[1].data['SPEC'][0], hdulist[2].data['SPEC'][0])
    full_ivar = np.append(hdulist[1].data['IVAR'][0], hdulist[2].data['IVAR'][0])

    data.add_component(full_wl, 'Wavelength')
    data.add_component(full_spec, 'Flux')
    data.add_component(1/np.sqrt(full_ivar), 'Uncertainty')

    return data

@mosviz_spectrum2d_loader('DEIMOS 2D Spectrum')
def deimos_spectrum2D_reader(filename):
    """
    Data loader for Keck/DEIMOS 2D spectra.

    This loads only the Flux and Inverse variance.
    Wavelength information comes from the WCS.
    """

    hdulist = fits.open(filename)
    data = Data(label='2D Spectrum')
    data.coords = coordinates_from_header(hdulist[1].header)
    data.header = hdulist[1].header
    data.add_component(hdulist[1].data['FLUX'][0], 'Flux')
    data.add_component(1/np.sqrt(hdulist[1].data['IVAR'][0]), 'Uncertainty')
    return data

@mosviz_cutout_loader('ACS Cutout Image')
def acs_cutout_image_reader(filename):
    """
    Data loader for the ACS cut-outs for the DEIMOS spectra.

    The cutouts contain only the image.
    """
```

(continues on next page)

(continued from previous page)

```
    hdulist = fits.open(filename)
    data = Data(label='ACS Cutout Image')
    data.coords = coordinates_from_header(hdulist[0].header)
    data.header = hdulist[0].header
    data.add_component(hdulist[0].data, 'Flux')

    return data
```

CHAPTER 9

Writing the Data Table

As mentioned above, when adding a dataset to the MOSViz viewer, you will be prompted to select column names and data loaders, but you can optionally encode these into the catalog metadata to save time. Note that not all file formats will support this kind of meta-data, so if you want to do this you will be restricted to certain formats for the catalog.

The main requirement is that when read in with the `Table` `read()` method, the `Table` `meta` attribute should be a dictionary that contains a `loaders` key and a `special_columns` key:

- `Table.meta['loaders']` should then be a dictionary that contains three keys - `spectrum1d`, `spectrum2d`, and `cutout`, and for each of these gives, as a string, the label of the reader to use.
- `Table.meta['special_columns']` should be a dictionary that contains one key/value pair for each special column listed in the table in *Overview*, where the key is the **Default name** given in the table and the value is the name of the actual column in the table.

Note that any metadata where the defaults are fine can be omitted. For example, for the special columns, if the actual name is the same as the default name, the key and value will be the same and can be ommitted from the metadata.

As an example, the following ECSV table header indicates the loaders to use, but does not list the special columns explicitly since they already have the expected names:

```
# %ECSV 0.9
# ---
# meta:
#   loaders:
#       spectrum1d: "DEIMOS 1D Spectrum"
#       spectrum2d: "DEIMOS 2D Spectrum"
#       cutout: "ACS Cutout Image"
# datatype:
# - {name: id, datatype: string}
# - {name: ra, unit: deg, datatype: float64}
# - {name: dec, unit: deg, datatype: float64}
# - {name: spectrum2d, datatype: string}
# - {name: spectrum1d, datatype: string}
# - {name: cutout, datatype: string}
# - {name: slit_width, unit: arcsec, datatype: float64}
```

(continues on next page)

(continued from previous page)

```
# - {name: slit_length, unit: arcsec, datatype: float64}
# - {name: pix_scale, datatype: float64}
id ra dec spectrum2d spectrum1d cutout slit_width slit_length pix_scale
deimos_12004808 214.21968 52.410386 Spectra/slit.1153.151R.fits.gz Spectra/spec1d.1153.151.12004808.
↪fits Cutouts/12004808.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_12008179 214.33785 52.454369 Spectra/slit.1203.063R.fits.gz Spectra/spec1d.1203.063.12008179.
↪fits Cutouts/12008179.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_12012573 214.34313 52.53112  Spectra/slit.1205.091R.fits.gz Spectra/spec1d.1205.091.12012573.
↪fits Cutouts/12012573.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_12016058 214.52242 52.580972 Spectra/slit.1208.055R.fits.gz Spectra/spec1d.1208.055.12016058.
↪fits Cutouts/12016058.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_12020734 214.49056 52.632246 Spectra/slit.1209.080R.fits.gz Spectra/spec1d.1209.080.12020734.
↪fits Cutouts/12020734.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_12020387 214.57266 52.642585 Spectra/slit.1210.072R.fits.gz Spectra/spec1d.1210.072.12020387.
↪fits Cutouts/12020387.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_12020049 214.62085 52.646039 Spectra/slit.1211.061R.fits.gz Spectra/spec1d.1211.061.12020049.
↪fits Cutouts/12020049.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_12019995 214.69602 52.631649 Spectra/slit.1212.038R.fits.gz Spectra/spec1d.1212.038.12019995.
↪fits Cutouts/12019995.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_12019653 214.77361 52.662353 Spectra/slit.1214.026R.fits.gz Spectra/spec1d.1214.026.12019653.
↪fits Cutouts/12019653.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_12008349 214.249   52.460424 Spectra/slit.1243.030R.fits.gz Spectra/spec1d.1243.030.12008349.
↪fits Cutouts/12008349.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_12012586 214.37004 52.52134  Spectra/slit.1243.079R.fits.gz Spectra/spec1d.1243.079.12012586.
↪fits Cutouts/12012586.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_12004455 214.27608 52.408039 Spectra/slit.1244.010R.fits.gz Spectra/spec1d.1244.010.12004455.
↪fits Cutouts/12004455.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_11051203 214.33513 52.381078 Spectra/slit.1246.011R.fits.gz Spectra/spec1d.1246.011.11051203.
↪fits Cutouts/11051203.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_12011504 214.61256 52.551567 Spectra/slit.1246.152R.fits.gz Spectra/spec1d.1246.152.12011504.
↪fits Cutouts/12011504.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_12024856 214.5929  52.718354 Spectra/slit.1252.066R.fits.gz Spectra/spec1d.1252.066.12024856.
↪fits Cutouts/12024856.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13004306 214.77715 52.814133 Spectra/slit.1253.152R.fits.gz Spectra/spec1d.1253.152.13004306.
↪fits Cutouts/13004306.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_12024118 214.73955 52.697049 Spectra/slit.1254.094R.fits.gz Spectra/spec1d.1254.094.12024118.
↪fits Cutouts/12024118.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_12020067 214.64333 52.632145 Spectra/slit.1255.041R.fits.gz Spectra/spec1d.1255.041.12020067.
↪fits Cutouts/12020067.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13019968 214.77751 52.910775 Spectra/slit.1302.115R.fits.gz Spectra/spec1d.1302.115.13019968.
↪fits Cutouts/13019968.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13026888 215.01438 52.949334 Spectra/slit.1306.072R.fits.gz Spectra/spec1d.1306.072.13026888.
↪fits Cutouts/13026888.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13026873 215.0064  52.95921  Spectra/slit.1306.077R.fits.gz Spectra/spec1d.1306.077.13026873.
↪fits Cutouts/13026873.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13026857 214.95442 52.969926 Spectra/slit.1306.094R.fits.gz Spectra/spec1d.1306.094.13026857.
↪fits Cutouts/13026857.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13026107 215.10585 53.003483 Spectra/slit.1308.070R.fits.gz Spectra/spec1d.1308.070.13026107.
↪fits Cutouts/13026107.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13025290 215.19495 52.963721 Spectra/slit.1309.034R.fits.gz Spectra/spec1d.1309.034.13025290.
↪fits Cutouts/13025290.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13043017 215.10605 53.116245 Spectra/slit.1311.114R.fits.gz Spectra/spec1d.1311.114.13043017.
↪fits Cutouts/13043017.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13051276 215.10065 53.128093 Spectra/slit.1311.121R.fits.gz Spectra/spec1d.1311.121.13051276.
↪fits Cutouts/13051276.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13041627 215.31852 53.104803 Spectra/slit.1313.048R.fits.gz Spectra/spec1d.1313.048.13041627.
↪fits Cutouts/13041627.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
```

(continues on next page)

(continued from previous page)

```
deimos_13050572 215.17647 53.154515 Spectra/slit.1313.104R.fits.gz Spectra/spec1d.1313.104.13050572.
↪fits Cutouts/13050572.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13050507 215.14259 53.169163 Spectra/slit.1313.120R.fits.gz Spectra/spec1d.1313.120.13050507.
↪fits Cutouts/13050507.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13058235 215.23847 53.184374 Spectra/slit.1314.098R.fits.gz Spectra/spec1d.1314.098.13058235.
↪fits Cutouts/13058235.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13049212 215.38783 53.136419 Spectra/slit.1315.047R.fits.gz Spectra/spec1d.1315.047.13049212.
↪fits Cutouts/13049212.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13049133 215.3953  53.156244 Spectra/slit.1315.052R.fits.gz Spectra/spec1d.1315.052.13049133.
↪fits Cutouts/13049133.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13058203 215.27553 53.210001 Spectra/slit.1315.105R.fits.gz Spectra/spec1d.1315.105.13058203.
↪fits Cutouts/13058203.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13018671 214.95738 52.921481 Spectra/slit.1343.084R.fits.gz Spectra/spec1d.1343.084.13018671.
↪fits Cutouts/13018671.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13026879 215.00536 52.95371  Spectra/slit.1343.108R.fits.gz Spectra/spec1d.1343.108.13026879.
↪fits Cutouts/13026879.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13034580 215.08674 53.055397 Spectra/slit.1352.022R.fits.gz Spectra/spec1d.1352.022.13034580.
↪fits Cutouts/13034580.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13058164 215.26445 53.18501  Spectra/slit.1352.117R.fits.gz Spectra/spec1d.1352.117.13058164.
↪fits Cutouts/13058164.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13040952 215.32582 53.068148 Spectra/slit.1355.091R.fits.gz Spectra/spec1d.1355.091.13040952.
↪fits Cutouts/13040952.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
deimos_13040873 215.40401 53.11767  Spectra/slit.1355.134R.fits.gz Spectra/spec1d.1355.134.13040873.
↪fits Cutouts/13040873.acs.v_6ac_.fits 0.2 3.3 0.0 0.66
```

If the ra/dec columns had a different name in the table, the header should instead look like e.g:

```
# %ECSV 0.9
# ---
# meta:
#   loaders:
#       spectrum1d: "DEIMOS 1D Spectrum"
#       spectrum2d: "DEIMOS 2D Spectrum"
#       cutout: "ACS Cutout Image"
#   special_columns:
#       ra: ra_j2000
#       dec: dec_j2000
# datatype:
# - {name: id, datatype: string}
# - {name: ra_j2000, unit: deg, datatype: float64}
# - {name: dec_j2000, unit: deg, datatype: float64}
...
```

Part V

API

CHAPTER 10

mosviz.loaders Package

Data loaders for MOSViz

10.1 Functions

acs_cutout_image_reader(file_name)	Data loader for the ACS cut-outs for the DEIMOS spectra.
deimos_spectrum1D_reader(file_name)	Data loader for Keck/DEIMOS 1D spectra.
deimos_spectrum2D_reader(file_name)	Data loader for Keck/DEIMOS 2D spectra.
nirspec_level2_reader(file_name)	Data Loader for level2 products.
nirspec_spectrum1d_reader(file_name)	
nirspec_spectrum2d_reader(file_name)	Data loader for simulated NIRSpec 2D spectrum.
pre_nircam_image_reader(file_name)	Data loader for simulated NIRCam image.
pre_nirspec_level2_reader(file_name)	THIS IS A TEST!
pre_nirspec_spectrum1d_reader(file_name)	Data loader for MOSViz 1D spectrum.
pre_nirspec_spectrum2d_reader(file_name)	Data loader for simulated NIRSpec 2D spectrum.

10.1.1 acs_cutout_image_reader

mosviz.loaders.**acs_cutout_image_reader**(*file_name*)

Data loader for the ACS cut-outs for the DEIMOS spectra.

The cutouts contain only the image.

10.1.2 deimos_spectrum1D_reader

mosviz.loaders.**deimos_spectrum1D_reader**(*file_name*)

Data loader for Keck/DEIMOS 1D spectra.

This loads the 'Bxspf-B' (extension 1) and 'Bxspf-R' (extension 2) and appends them together to proudce the combined Red/Blue Spectrum along with their Wavelength and Inverse Variance arrays.

10.1.3 deimos_spectrum2D_reader

mosviz.loaders.**deimos_spectrum2D_reader**(*file_name*)
Data loader for Keck/DEIMOS 2D spectra.

This loads only the Flux and Inverse variance. Wavelength information comes from the WCS.

10.1.4 nirspec_level2_reader

mosviz.loaders.**nirspec_level2_reader**(*file_name*)
Data Loader for level2 products. Uses extension information to index fits hdu list. The ext info is included in the file_name as follows: <file_path>[<ext>]

10.1.5 nirspec_spectrum1d_reader

mosviz.loaders.**nirspec_spectrum1d_reader**(*file_name*)

10.1.6 nirspec_spectrum2d_reader

mosviz.loaders.**nirspec_spectrum2d_reader**(*file_name*)
Data loader for simulated NIRSpec 2D spectrum.

This function extracts the DATA, QUALITY, and VAR extensions and returns them as a glue Data object.

It then uses the header keywords of the DATA extension to detemine the wavelengths.

10.1.7 pre_nircam_image_reader

mosviz.loaders.**pre_nircam_image_reader**(*file_name*)
Data loader for simulated NIRCam image. This is for the full image, where cut-outs will be created on the fly.

From the header: If ISWFS is T, structure is:

- Plane 1: Signal [frame3 - frame1] in ADU
- Plane 2: Signal uncertainty [sqrt(2*RN/g + |frame3|)]

If ISWFS is F, structure is:

- Plane 1: Signal from linear fit to ramp [ADU/sec]
- Plane 2: Signal uncertainty [ADU/sec]

Note that in the later case, the uncertainty is simply the formal uncertainty in the fit parameter (eg. uncorrelated, WRONG!). Noise model to be implemented at a later date. In the case of WFS, error is computed as SQRT(2*sigma_read + |frame3|) which should be a bit more correct - ~Fowler sampling.

The FITS file has a single extension with a data cube. The data is the first slice of the cube and the uncertainty is the second slice.

10.1.8 pre_nirspec_level2_reader

mosviz.loaders.**pre_nirspec_level2_reader**(*file_name*)
THIS IS A TEST!

10.1.9 pre_nirspec_spectrum1d_reader

mosviz.loaders.**pre_nirspec_spectrum1d_reader**(*file_name*)
Data loader for MOSViz 1D spectrum.

This function extracts the DATA, QUALITY, and VAR extensions and returns them as a glue Data object.

It then uses the header keywords of the DATA extension to detemine the wavelengths.

10.1.10 pre_nirspec_spectrum2d_reader

mosviz.loaders.**pre_nirspec_spectrum2d_reader**(*file_name*)
Data loader for simulated NIRSpec 2D spectrum.

This function extracts the DATA, QUALITY, and VAR extensions and returns them as a glue Data object.

It then uses the header keywords of the DATA extension to detemine the wavelengths.

CHAPTER 11

mosviz.viewers Package

11.1 Classes

MOSVizViewer(session[, parent])	

11.1.1 MOSVizViewer

class mosviz.viewers.**MOSVizViewer**(*session*, *parent=None*)

Bases: glue.viewers.common.qt.data_viewer.DataViewer

Attributes Summary

LABEL	
subtools	
tools	
window_closed()	

Methods Summary

add_data(self, data)	Processes data message from the central communication hub.
add_data_for_testing(self, data)	Processes data message from the central communication hub.
add_slit(self[, row, width, length])	
add_subset(self, subset)	Processes subset messages from the central communication hub.

Continued on next page

Table 3 – continued from previous page

closeEvent(self, event)	Clean up the extraneous data components created when opening the MOSViz viewer by overriding the parent class's close event.
get_comment(self)	
get_flag(self)	
get_slit_dimensions_from_file(self)	
get_slit_units_from_file(self)	
initialize_toolbar(self)	Initialize the custom toolbar for the MOSViz viewer.
layer_view(self)	
load_exposure(self, index)	Loads the level 2 exposure into the 2D spectrum plot widget.
load_selection(self, row)	Processes a row in the MOS catalog by first loading the data set, updating the stored data components, and then rendering the data on the visible MOSViz viewer plots.
load_ui(self)	Setup the MOSView viewer interface.
options_widget(self)	
refresh_comments(self)	
register_to_hub(self, hub)	
render_data(self, row[, spec1d_data, ...])	Render the updated data sets in the individual plot widgets within the MOSViz viewer.
send_NumericalDataChangedMessage(self)	
set_locked_axes(self[, x, y])	
setup_connections(self)	Connects gui elements to event calls.
show(self)	
update_comments(self[, pastSelection])	Process comment and flag changes and save to file.
write_comments(self)	Setup save file.

Attributes Documentation

LABEL = 'MOSViz Viewer'

subtools = []

tools = []

window_closed

Methods Documentation

add_data(*self*, *data*)

Processes data message from the central communication hub.

Parameters

data

[glue.core.data.Data] Data object.

add_data_for_testing(*self*, *data*)
Processes data message from the central communication hub.

Parameters

data
[glue.core.data.Data] Data object.

add_slit(*self*, *row=None*, *width=None*, *length=None*)

add_subset(*self*, *subset*)
Processes subset messages from the central communication hub.

Parameters

subset :
Subset object.

closeEvent(*self*, *event*)
Clean up the extraneous data components created when opening the MOSViz viewer by overriding the parent class's close event.

get_comment(*self*)

get_flag(*self*)

get_slit_dimensions_from_file(*self*)

get_slit_units_from_file(*self*)

initialize_toolbar(*self*)
Initialize the custom toolbar for the MOSViz viewer.

layer_view(*self*)

load_exposure(*self*, *index*)
Loads the level 2 exposure into the 2D spectrum plot widget.

It can also load back the level 3 spectrum.

load_selection(*self*, *row*)
Processes a row in the MOS catalog by first loading the data set, updating the stored data components, and then rendering the data on the visible MOSViz viewer plots.

Parameters

row
[astropy.table.Row] A row object representing a row in the MOS catalog. Each key should be a column name.

load_ui(*self*)
Setup the MOSView viewer interface.

options_widget(*self*)

refresh_comments(*self*)

register_to_hub(*self*, *hub*)

render_data(*self*, *row*, *spec1d_data=None*, *spec2d_data=None*, *image_data=None*, *level2_data=None*)
Render the updated data sets in the individual plot widgets within the MOSViz viewer.

send_NumericalDataChangedMessage(*self*)

set_locked_axes(*self*, *x=None*, *y=None*)

setup_connections(*self*)
Connects gui elements to event calls.

show(*self*)

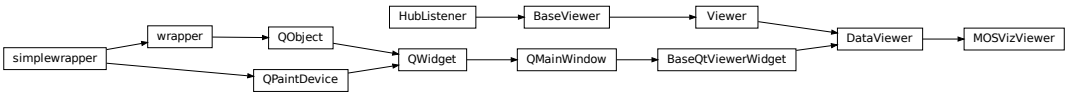
update_comments(*self*, *pastSelection=False*)
Process comment and flag changes and save to file.

Parameters

pastSelection
[bool] True when updating past selections. Used when user forgets to save.

write_comments(*self*)
Setup save file. Write comments and flags to file

11.2 Class Inheritance Diagram



CHAPTER 12

mosviz.widgets Package

12.1 Classes

CycleForwardTool(viewer[, toolbar])	
CyclePreviousTool(viewer[, toolbar])	
DrawableImageWidget(*args[, slit_controller])	
Line1DWidget(parent, flags, ...)	
MOSImageWidget(*args, **kwargs)	
MOSViewerToolbar(str, parent)	
OptionsWidget(parent, flags, ...)	

12.1.1 CycleForwardTool

class mosviz.widgets.**CycleForwardTool**(*viewer*, *toolbar=None*)

Bases: glue.viewers.common.tool.Tool

Methods Summary

activate(self)	Fired when the toolbar button is activated

Methods Documentation

activate(*self*)

Fired when the toolbar button is activated

12.1.2 CyclePreviousTool

class mosviz.widgets.**CyclePreviousTool**(*viewer*, *toolbar=None*)
Bases: glue.viewers.common.tool.Tool

Methods Summary

activate(self)	Fired when the toolbar button is activated

Methods Documentation

activate(*self*)
Fired when the toolbar button is activated

12.1.3 DrawableImageWidget

class mosviz.widgets.**DrawableImageWidget**(**args*, *slit_controller=None*, ***kwargs*)
Bases: mosviz.widgets.MOSImageWidget

Attributes Summary

slit_controller
tools

Methods Summary

draw_rectangle(self[, x, y, width, height])	
draw_slit(self)	Draw the slit patch stored in the slit controller.
launch_slit_ui(self)	
reset_limits(self)	Auto set the limits of the axes.
set_limits(self[, x_min, x_max, y_min, y_max])	Manually set the limits of the axes.
set_slit_limits(self)	Set y limits of plot according to slit length

Attributes Documentation

slit_controller = None

tools = ['mpl:home', 'mpl:save', 'mpl:pan', 'mpl:zoom', 'image:contrast', 'image:colormap', 'mosviz:slit

Methods Documentation

draw_rectangle(*self*, *x=None*, *y=None*, *width=None*, *height=None*)

draw_slit(*self*)
Draw the slit patch stored in the slit controller.

launch_slit_ui(*self*)

reset_limits(*self*)
Auto set the limits of the axes.

set_limits(*self*, *x_min=None*, *x_max=None*, *y_min=None*, *y_max=None*)
Manually set the limits of the axes.

set_slit_limits(*self*)
Set y limits of plot according to slit length

12.1.4 Line1DWidget

class mosviz.widgets.**Line1DWidget**(*parent: QWidget = None*, *flags: Union[Qt.WindowFlags, Qt.WindowType] = Qt.WindowFlags()*)
Bases: PyQt5.QtWidgets.QMainWindow

Attributes Summary

axes
tools
window_closed()

Methods Summary

initialize_toolbar(self)
no_data(self)
set_data(self, x, y[, yerr])
set_status(self, message)

Attributes Documentation

axes

tools = ['mpl:home', 'mpl:save', 'mpl:pan', 'mpl:zoom']

window_closed

Methods Documentation

initialize_toolbar(*self*)

no_data(*self*)

set_data(*self*, *x*, *y*, *yerr=None*)

set_status(*self*, *message*)

12.1.5 MOSImageWidget

class mosviz.widgets.**MOSImageWidget**(**args*, ***kwargs*)

Bases: glue.viewers.image.qt.StandaloneImageViewer

Methods Summary

set_status(self, status)	

Methods Documentation

set_status(*self*, *status*)

12.1.6 MOSViewerToolbar

class mosviz.widgets.**MOSViewerToolbar**(*str*, *parent: QWidget = None) QToolBar(parent: QWidget = None*)

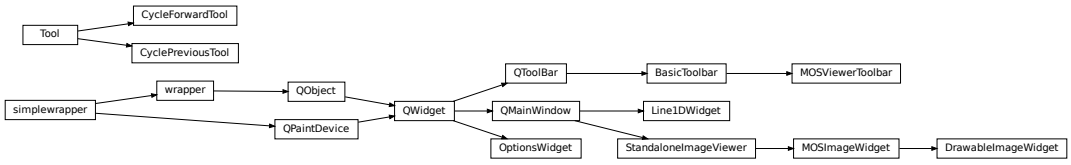
Bases: glue.viewers.common.qt.toolbar.BasicToolbar

12.1.7 OptionsWidget

class mosviz.widgets.**OptionsWidget**(*parent: QWidget = None*, *flags: Union[Qt.WindowFlags, Qt.WindowType] = Qt.WindowFlags()*)

Bases: PyQt5.QtWidgets.QWidget

12.2 Class Inheritance Diagram



Python Module Index

m

Index