

---

# **specviz Documentation**

*Release 0.4.4*

**JDADF Developers**

**Nov 14, 2017**



---

## Contents

---

<b>I Demo</b>	<b>3</b>
<b>II Installation and Setup</b>	<b>7</b>
1 Installation	9
2 Launching SpecViz	13
<b>III Using SpecViz</b>	<b>15</b>
3 Viewer	17
4 Model Fitting	21
5 Custom Loaders	25
<b>IV References/API</b>	<b>31</b>
6 SpecViz API	33
<b>V Indices and tables</b>	<b>65</b>
Python Module Index	69



SpecViz is a tool for visualization and quick-look analysis of 1D astronomical spectra. It is written in the Python programming language, and therefore can be run anywhere Python is supported (see *Installation*). SpecViz is capable of reading data from FITS and ASCII tables (see *Custom Loaders*).

SpecViz allows spectra to be easily plotted and examined. It supports instrument-specific data quality handling, flexible spectral units conversions, custom plotting attributes, plot annotations, tiled plots, and other features.

SpecViz notably includes a measurement tool for spectral lines which enables the user, with a few mouse actions, to perform and record measurements. It has a model fitting capability that enables the user to create simple (e.g., single Gaussian) or multi-component models (e.g., multiple Gaussians for emission and absorption lines in addition to regions of flat continua). SpecViz incorporates various methods for fitting such models to data. For more details, see *Model Fitting*.

Furthermore, SpecViz allows for overplotting or combining of spectra.

**SpecViz will soon include the ability to**

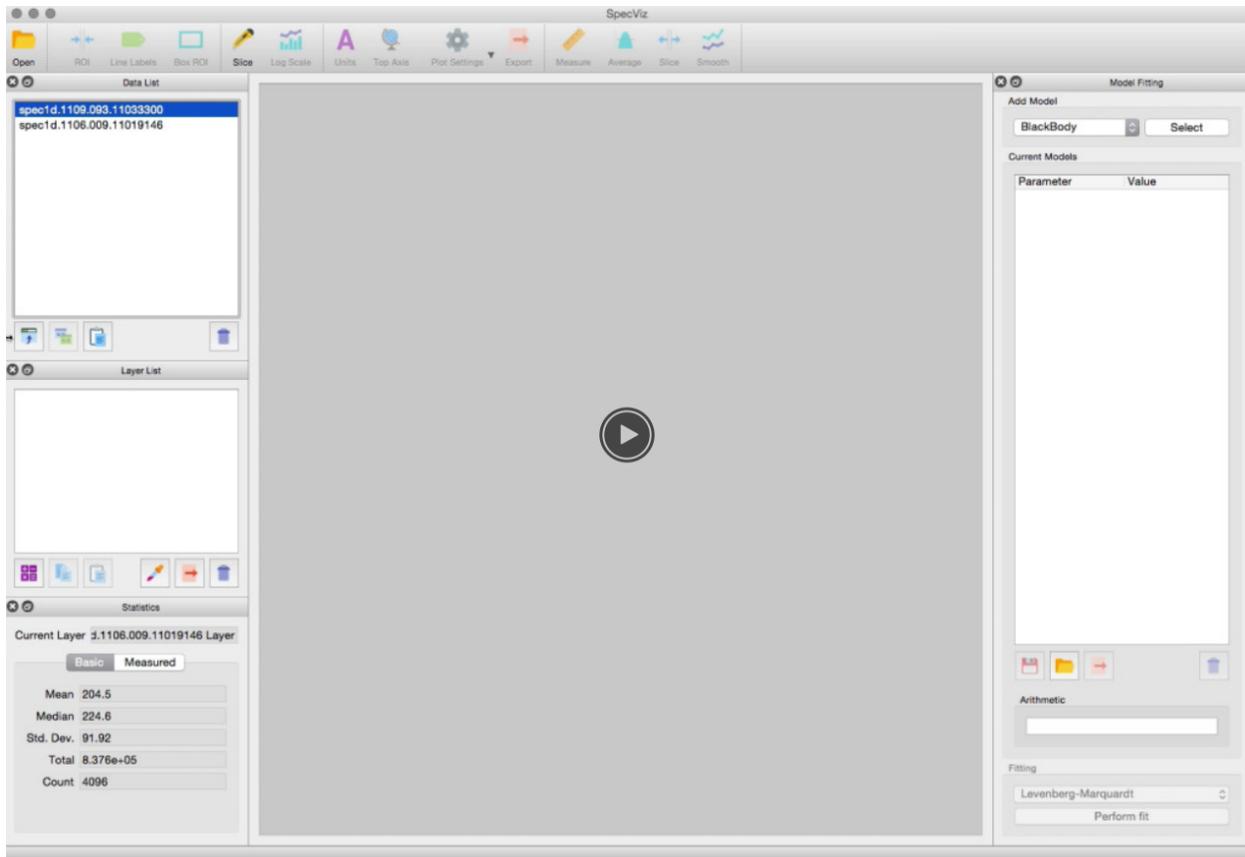
- Measure the average of multiple spectra, detrending, and apply Fourier filters.
- Interactively renormalize data from spectral templates.
- Overplot spectral line lists.
- And more...



**Part I**

**Demo**







## **Part II**

# **Installation and Setup**



SpecViz is distributed through the [Anaconda](#) package manager. Specifically, it lives within Space Telescope Science Institute's [AstroConda](#) channel.

If you do not have Anaconda, please follow the [instructions here](#) to install it, or scroll down for manual installation of SpecViz.

### 1.1 Install via Anaconda

If you have AstroConda setup, then all you have to do to install SpecViz is simply type the following at any Bash terminal prompt:

```
$ conda install specviz
```

If you do not have AstroConda setup, then you can install SpecViz by specifying the channel in your install command:

```
$ conda install --channel http://ssb.stsci.edu/astroconda specviz
```

At this point, you're done! You can launch SpecViz by typing the following at any terminal:

```
$ specviz
```

#### 1.1.1 Uninstalling

To uninstall via Anaconda, simply type the following at a command line:

```
$ conda uninstall specviz
```

## 1.2 Install via source

SpecViz 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, with the exception of PyQt/PySide.

- Python 3 (recommended) or Python 2
- PyQt5 (recommended), PyQt4, or PySide
- Astropy
- Numpy
- Scipy
- PyQtGraph
- qtpy

### 1.2.1 By using pip

Clone the SpecViz repository somewhere on your system, and install locally using pip. If you are using an Anaconda virtual environment, please be sure to activate it first before installing: `$ source activate <environment_name>`.

```
$ pip install git+http://github.com/spacetelescope/specviz.git@v0.4.4
```

This uses the pip installation system, so please note that

1. You need to have pip installed (included in most Python installations).
2. You do **not** need to run `python setup.py install`.
3. You do **not** need to install the dependencies by hand (except for PyQt).

Likewise, the pip command will use your default Python to install. You can specify by using pip2 or pip3, if you're not using a virtual environment.

### 1.2.2 By cloning

You may also install by cloning the repository directly

```
$ git clone https://github.com/spacetelescope/specviz.git
$ cd specviz
$ git checkout tags/v0.3.0
$ python setup.py install
```

### 1.2.3 PyQt/PySide bindings

SpecViz requires PyQt. Currently, only python environments with 3.5 or higher installed can use pip to install PyQt5, in which case simply type:

```
$ pip install pyqt5
```

to install it on your system.

In any other case, PyQt can be installed via anaconda:

```
$ conda install pyqt
```

SpecViz works with with PyQt4 and PySide, but it is recommended that users use PyQt5 if available.

## 1.2.4 Uninstalling

To uninstall via pip, simply type the following at a command line:

```
$ pip uninstall specviz
```

## 1.3 Known Issues

On a Mac with Qt5, depending on exactly how you have set up Anaconda, you might see the following error after following the above instructions:

```
This application failed to start because it could not find or load the Qt platform plugin "cocoa".
```

```
Reinstalling the application may fix this problem.
```

If you see this message, you have encountered an incompatibility between Anaconda's packaging of Qt4 and Qt5. The workaround is to uninstall Qt4 with the following command:

```
$ conda uninstall pyqt qt
```

and SpecViz should now happily run.

Conversely, if you've had PyQt5 installed previously and you wish to run the PyQt4 version, you may run into a similar error:

```
$ RuntimeError: the PyQt4.QtCore and PyQt5.QtCore modules both wrap the
QObject class
```

This issue can be solved with the following command:

```
$ conda uninstall pyqt5 qt5
```



---

### Launching SpecViz

---

Once you've installed SpecViz, you can launch it via the command line:

```
$ specviz
```

If you only wish to inspect a single FITS or ASCII file using the default *Custom Loaders* file formatting, you can also pass in the filename as a command line argument, as follows:

```
$ specviz filename
```

You may also include the name of a custom loader as second optional argument:

```
$ specviz filename --format="my-custom-format"
```



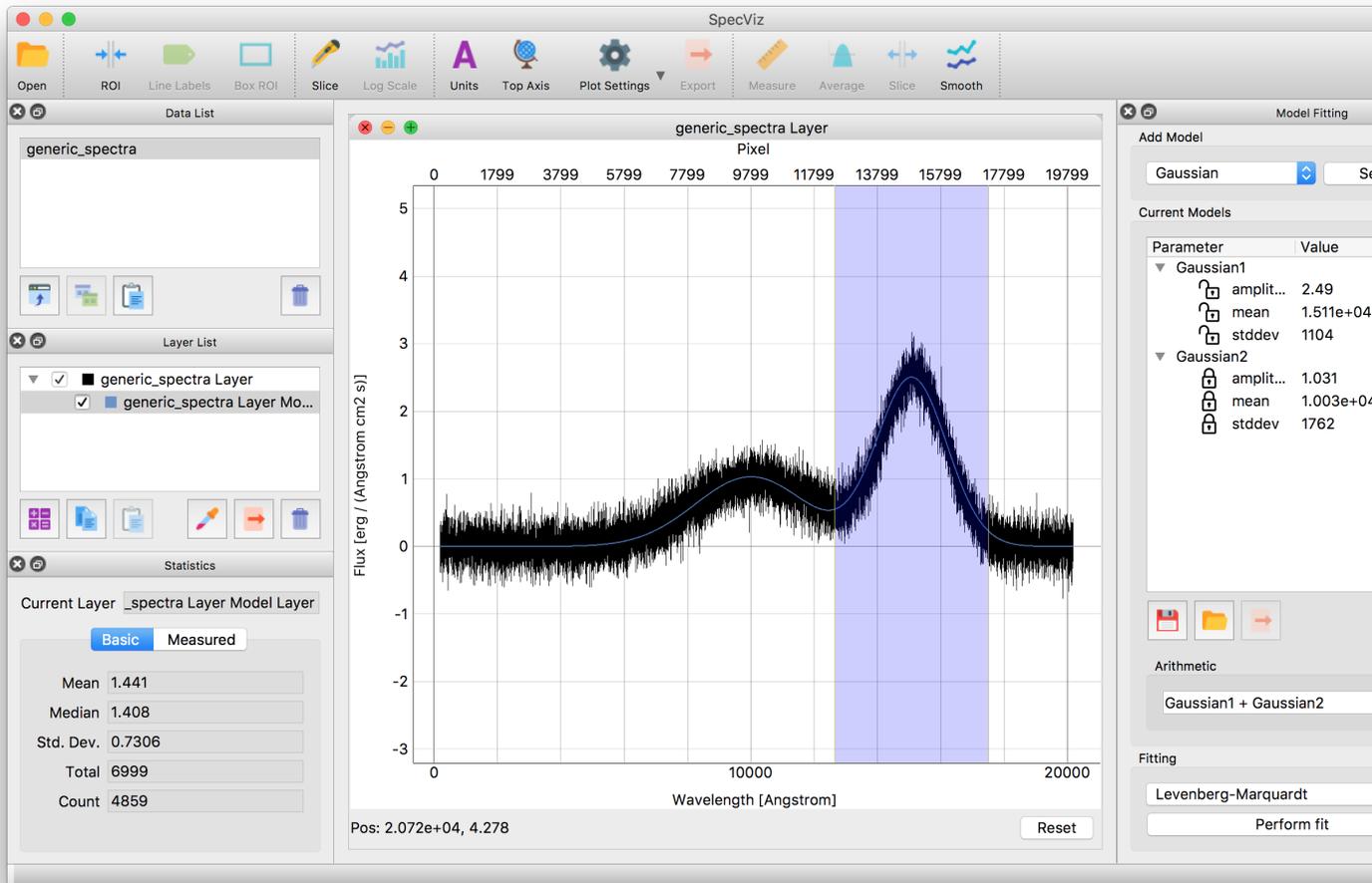
**Part III**

**Using SpecViz**



# CHAPTER 3

## Viewer



---

**Note:** Some buttons and menu options are not functional yet, as this is a work in progress.

---

To open a file:

1. Click on the folder icon on top left corner or select File -> Open from menu.
2. Select the desired file and data type in the file dialog and click “Open”.
3. The file will be listed in the “Data List” window (top half of left panel).
4. To remove the file from “Data”, click on “trash can” icon at lower right corner of the “Data List” window.

To plot the spectrum that is now loaded, **click on the plot icon** (lower left corner of the Data window). It will be plotted in the central window. Basic statistics for the displayed spectrum will be shown in the “Statistics” window at the lower left.

Each spectrum loaded will have its own layer, as shown the “Layer List” window at the lower left. New layers are created each time a spectrum is modified or a fit is performed, and the original spectrum is untouched. To work on the file of interest, click on its plot window to bring it in focus.

To adjust plot display:

1. If your mouse has center wheel, it can be used to zoom in and out while the cursor is over the plot.
2. Click on the “big wheel” icon on top of the plot to change top axis parameters or displayed units.
3. Left click on the plot to bring up a context menu.
4. Select “View All” to view all available data (if zoomed in).
5. Select “X Axis” or “Y Axis” to adjust respective axis behaviors.
6. Select “Mouse Mode” to toggle between 3-button and 1-button mouse.
7. More plot options are available under “Plot Options”.
8. Select “Export...” to save plot out as an image.

To select a region of interest (ROI):

1. Select the layer from “Layers” list (if you have multiple).
2. Click on the “rectangle” icon on the top left of the plot. An adjustable rectangle will appear on plot display.
3. Click and drag the edges to adjust wavelength coverage. As the region changes, basic statistics in the “Statistics” window on bottom left of the viewer will also update accordingly.
4. Click inside the rectangle and drag to reposition it without resizing.
5. Click the left-most “knife” button under “Layers” (bottom left of the left panel) to create a new layer slice from the ROI.
6. To remove a layer slice, select the layer and click on “trash can” button under “Layers”.
7. To remove ROI, left click while it is highlighted and select “Remove ROI” from the context menu.

---

**Note:** Layer arithmetic (the “calculator” button under “Layers”) is work in progress.

---

To measure equivalent width and related properties:

1. Select the layer from “Layers” list (if you have multiple).
2. Add three ROIs.

3. Position the orange ROI over the emission or absorption line of interest and adjust accordingly.
4. Position each of the blue ROIs over two different continuum areas, one on each side of the emission/absorption line.
5. As you adjust the regions, values for equivalent width etc. will be updated under the “Measured” tab under “Statistics” on top right of the viewer.

---

**Note:** When multiple ROIs are present, the statistics in “Basic” tab reflect all data points contained in the ROIs.

---

To fit a model to the selected ROI, see *Model Fitting*.

To quit SpecViz, select File -> Exit from menu.



---

Model Fitting

---

SpecViz utilizes [Astropy Models and Fitting](#) to fit models to its spectra. For example, you can fit one model to the continuum, another to an emission line of interest, and yet another to an absorption line.

Currently, the following models are available:

SpecViz Model Name	Astropy Model Class
BrokenPowerLaw	BrokenPowerLaw1D
Const	Const1D
ExponentialCutoffPowerLaw	ExponentialCutoffPowerLaw1D
Gaussian	Gaussian1D
GaussianAbsorption	Gaussian1D with negative amplitude
Linear	Linear1D
LogParabola	LogParabola1D
Lorentz	Lorentz1D
MexicanHat	MexicanHat1D
Trapezoid	Trapezoid1D
PowerLaw	PowerLaw1D
Scale	Scale
Shift	Shift
Sine	Sine1D
Spline	UnivariateSpline
Voigt	Voigt1D

The models can be fitted with the following fitters:

SpecViz Fitter Name	Astropy Fitter Class
Levenberg-Marquardt	LevMarLSQFitter
Simplex	SimplexLSQFitter
SLSQP	SLSQPLSQFitter

To use a model:

1. Select the layer you wish to operate on from the “Layer List” window in the bottom left. For example, you can choose the layer containing your emission or absorption line.
2. Create and position a region of interest (ROI) as described in the *Viewer* section of the documentation.
3. Select the desired model from the Add Model drop-down box and click Select to add it to Current Models.
4. If desired, repeat the above step to add additional models.
5. A new model layer will be created under your data layer in the “Layer List” window.

To edit model parameters or enter a better first estimate of the model parameters:

1. Select the model layer in the “Layer List” window that contains the desired model.
2. If desired, double-click on the model name to rename it. When you see a blinking cursor, enter its new name and press “Enter”.
3. Expand the model listing under Current Models on the right of the viewer.
4. Double-click on the desired model parameter value in the listing. When you see a blinking cursor, enter the new value and press Enter.

To fit a model:

1. Select the model layer under Layers that contains the model(s) you wish to fit to your data.
2. Click the lock icon next to any parameter to choose whether it should be kept fixed (closed lock) or allowed to vary (open lock) during fitting.
3. Select the desired fitter from Fitting using its drop-down menu.
4. Click Perform Fit. This may take up to a few seconds, depending on the complexity of the fit.
5. The associated model parameters will be adjusted accordingly.

The Arithmetic text box is used to define the relationship between different models for the same layer. If nothing is defined, the default is to add all the models together. To describe a non-default model relationship, adjust the math operators, as shown in the examples below and then press Enter to produce the compound model:

```
Linear1 + Gaussian1
```

```
Linear1 * Gaussian1
```

```
Gaussian1 - Gaussian2
```

The entity that results from lumping together all the models, and combining them either using the arithmetic behavior expression, or just adding them all together, is called a “compound model”.

## 4.1 Model names

When added to the Current Models list, a model will receive a default name that is generated from the model type (as listed in the drop down model selector) plus a running numerical suffix.

These names can be changed by clicking on the default name and entering a new name. Note that changing model names will require that any expression in the Arithmetic text box be edited accordingly.

For now, we are limited to only alphanumeric characters (and no white spaces) when re-naming models.

## 4.2 Spline model

Note that the Spline model is of an intrinsically different nature than the other models included in the drop down list of models. The Spline model, when added to a pre-existing list of models, or when added by itself to an empty list, will immediately be fit to the data within the currently defined Regions Of Interest (ROIs). That is, being a linear model, there is no need to iterate in search of a “best fit” spline. It is just computed once and for all, and kept as part of the compound model that is built from the models in the list and the arithmetic behavior expression.

This implies that, to change the regions of interest that define the spline, one must remove the spline from the list of models. Then, the user must redefine the ROIs and add a new spline to the list of models to be fit. To change a spline parameter, there is no need do discard the spline. Just do it in the same way as with other models: just type in the new value for the parameter.

Subsequently, when the fitter iterates the compound model in search of a best solution, the spline model will act as a constant. That is, it will be used to compute the global result of the compound model, but its parameters won't be accessed, and varied, by the fitter. Thus, the spline parameters are not fitted, they are just a convenient mechanism that enables user access to the parameter's values.

The documentation for the spline model can be found here:

<http://docs.scipy.org/doc/scipy-0.16.0/reference/generated/scipy.interpolate.UnivariateSpline.html>

Note that SpecViz provides access, at this point, to just two of the parameters in the scipy implementation of the spline function. Pay special attention to the smooth parameter. SpecViz initializes it to a ‘best guess’ ( $1/\text{len}(\text{wavelength})$ ). Too small of a value may cause the spline to enter an infinite loop. Change the smooth value with care, trying to stay close to the default value.

---

**Note:** Model arithmetic is a work in progress.

---

## 4.3 Saving and exporting models to a file

Selecting a model layer under “Layers” will enable the *Save* (the “floppy disk” icon) and *Export* (the “out the door” icon) buttons under “Current Models” on the right of the viewer. When a model is saved as a file on disk, it can later be read into SpecViz as described below. Exporting a model to a file will create a Python script in a .py file. This file can be directly imported by Python in a command-line session.

Click on either button to get a file dialog window. Provide a name for the model file, if this file name does not end with the correct suffix, the suffix will automatically be appended. Click “Save”, or just press the Return/Enter key. The correct suffix for saved and exported files are .yaml and .py, respectively.

## 4.4 Save and load

Saving the model to a file works in the same way as *Export*. The difference is that a saved model can be later read back into SpecViz via the “Load” button (the “folder” icon), also under “Current Models”, whereas the exported model cannot.

For the “Load” button to be enabled, a data (spectrum) layer (not a model layer) must be selected in the “Layer List” window. The selected .yaml model file will generate a model that will be attached to a new model layer associated with the selected data layer.

The file is written in the YAML format. Being a plain text file with a self-explanatory structure, it can be edited at will by the user, e.g., to add bounds, fixed flags, and ties to the model parameters. Note that these extra, user-defined

attributes, won't be accessible from SpecViz's user interface. They will however, be used by the fitter when a fit is run on the model. They will also be written out correctly, either when saving or exporting the model.

---

**Note:** YAML format for saved models and usage of advanced features like bounds and fixed flags are work in progress.

---

## 4.5 Export

This will save the model in the currently selected model layer to a file that can be directly imported by Python. The file is just a plain text file with the model given as a Python expression. The model is associated a variable named 'model1'.

The following example uses the 'test3.py' file name, and a model comprised of a constant and a gaussian:

```
>>> import test3
>>> test3
<module 'test3' from '/my/saved/models/test3.py'>
>>> test3.model1
<CompoundModel0(amplitude_0=0.297160787184, amplitude_1=2.25396100263, mean_1=15117.1710847, stddev_
↪1=948.493577186)>
>>> print(test3.model1)
Model: CompoundModel0
Inputs: ('x',)
Outputs: ('y',)
Model set size: 1
Parameters:
      amplitude_0      amplitude_1      mean_1      stddev_1
-----
0.297160787184 2.25396100263 15117.1710847 948.493577186
```

The file can be edited by the user, e.g., to add bounds, fixed flags, and ties to the model parameters.

**Warning:** Security issues with importing model this way into Python and usage of advanced features like bounds and fixed flags are work in progress.

---

## Custom Loaders

---

SpecViz utilizes [Astropy I/O registry](#) and [YAML data serialization language](#) to enable flexible support for a variety of data formats both in FITS and ASCII.

When `specviz` is called with a filename as argument (see [Launching SpecViz](#)), the default formats below are assumed based on file extension:

- `.fits` or `.mits` – *Generic FITS Loader*
- `.txt` or `.dat` – *ASCII Loader*

By examining the [YAML definitions](#) in the following sub-sections and their associated [example data files](#), you will be able to create your own YAML file to define most custom data formats (see [Creating a Custom Loader](#)). In addition, there are also other custom loaders that come with SpecViz that follow the same rules but are modified to load JWST DADF test data, which you can also use as a reference.

While using YAML is very flexible, it is also very sensitive to slight changes in your file format. For instance, if you have two files from the same instrument but processed differently (say, one was extracted using IRAF and another one using your own IDL program), they might have different formats and will need separate YAML files.

### 5.1 Generic FITS Loader

```
--- !CustomLoader
name: Generic Fits
extension: [fits, mits]
wcs:
  hdu: 0
data:
  hdu: 1
  col: 0
uncertainty:
  hdu: 1
  col: 1
type: 'std'
```

```
meta:
  author: Nicholas Earl
```

This is `generic.yaml`, which is a built-in YAML definition for a “generic” FITS file.

```
--- !CustomLoader
```

The first line states that this YAML file defines our custom loader. This is always the same no matter what kind of format you are defining.

```
name: Generic Fits
extension: [fits, mits]
```

These two lines define the format name and accepted extensions, respectively. In SpecViz GUI, this will translate to “Generic Fits (\*.fits \*.mits)” in the file type drop-down menu.

```
wcs:
  hdu: 0
```

This instructs the loader to look for WCS information in the PRIMARY (Extension 0) header. SpecViz also uses this header for other look-ups, e.g., flux unit from BUNIT (see below). Therefore, even if there are no WCS information in your file, you always define this block and point the HDU value to the PRIMARY header. If WCS information are available and supported by [Astropy WCS](#), they will be used to establish dispersion values and unit.

In the absence of WCS or the presence of explicit dispersion column, an additional `dispersion:` block (not shown) can be defined similarly as `data:` (see below). Unlike flux, TNULL masking is ignored. If its column does not have TUNIT and `unit: 'unitname'` is defined, SpecViz will fall back to WCS unit. If all these unit look-ups failed, it defaults to null unit. In the presence of `unit:` definition, it overrides both TUNIT and WCS.

```
data:
  hdu: 1
  col: 0
```

This instructs the loader to look for flux values (data) in Extension 1, the first column (column index starts from 0). If the column complies to FITS standards (see [Astropy FITS Table](#)), flux unit (inferred from TUNIT) and data mask (inferred from TNULL) are also extracted from the same column.

If TUNIT is not defined, loader will look for `unit: 'unitname'` definition within this block, the unit name must be one that is accepted by [Astropy Units](#) (case sensitive). If that is undefined as well, flux unit is extracted from BUNIT keyword in the same header that contains WCS information (see `wcs:` block for details). If all unit look-ups failed, flux unit is assumed to be  $\text{erg}^{-1} \text{cm}^{-2} \text{s}^{-1}$ .

Note that TNULL is not the same as DQ arrays, which can be similarly defined with `mask:` block (not shown). If both TNULL and DQ are defined, the masks will be combined.

```
uncertainty:
  hdu: 1
  col: 1
  type: 'std'
```

This instructs the loader to look for flux uncertainty values in Extension 1, the second column. Uncertainty type `'std'` states that the values are standard deviation (as opposed to inverse variance, `'ivar'`). Unlike flux data, its TNULL masking is ignored and `unit:` tag is not supported. If TUNIT is present, loader will attempt to convert the values to flux unit first. Otherwise, its unit is assumed to be the same as flux unit. If inverse variance is given, square-root is applied to the inversed values before being converted to `StdDevUncertainty`.

```
meta:
  author: Nicholas Earl
```

The `meta:` block can contain any metadata tags you wish to include. They do not affect how SpecViz works. In this example, the `author:` tag identifies Nicholas Earl as the origin author of this YAML file.

## 5.2 ASCII Loader

```
--- !CustomLoader
name: ASCII
extension: [txt, dat]
dispersion:
  col: 0
  unit: 'Angstrom'
data:
  col: 1
  unit: 'erg / (Angstrom cm2 s)'
uncertainty:
  col: 2
  type: 'std'
meta:
  author: STScI
```

This is `ascii.yaml`, which is a built-in YAML definition for a “generic” ASCII file.

```
--- !CustomLoader
```

The first line states that this YAML file defines our custom loader. This is always the same no matter what kind of format you are defining.

```
name: ASCII
extension: [txt, dat]
```

These two lines define the format name and accepted extensions, respectively. In SpecViz GUI, this will translate to “ASCII (\*.txt \*.dat)” in the file type drop-down menu. All ASCII files must comply to [Astropy ASCII Table](#) standards.

Any header comments with `KEY = VALUE` format will be extracted as header metadata information (currently not used by SpecViz).

```
dispersion:
  col: 0
  unit: 'Angstrom'
```

Unlike *Generic FITS Loader*, ASCII table does not contain WCS. Therefore, the `dispersion:` block is necessary to define the actual dispersion (e.g., wavelength) values. This instructs the loader to look for dispersion values in the first column (column index starts from 0). Its unit, if not defined in the table itself (e.g., via [IPAC table format](#)), will be taken from the `unit:` tag. The given unit name must be one that is accepted by [Astropy Units](#) (case sensitive). If unit is defined in both table and tag, the latter is ignore. If unit is not defined anywhere, it defaults to Angstrom.

```
data:
  col: 1
  unit: 'erg / (Angstrom cm2 s)'
```

This instructs the loader to look for flux values (`data`) in the second column. Flux unit handling is similar to dispersion unit (see above), except that the default unit would be  $\text{erg}^{-1} \text{cm}^{-2} \text{s}^{-1}$  if undefined.

If there is an associated DQ column, it can be extracted in a similar fashion using a `mask:` block specifying the column index (unit is not applicable). Like *Generic FITS Loader*, zero mask values signify good data.

```
uncertainty:
  col: 2
  type: 'std'
```

This instructs the loader to look for flux uncertainty values in the third column. Uncertainty type `'std'` states that the values are standard deviation (as opposed to inverse variance, `'ivar'`). Its unit must be the same as flux unit. If inverse variance is given, square-root is applied to the inversed values before being converted to `StdDevUncertainty`.

```
meta:
  author: STScI
```

The `meta:` block can contain any metadata tags you wish to include. They do not affect how SpecViz works. In this example, the `author:` tag identifies STScI as the origin author of this YAML file.

## 5.3 Creating a Custom Loader

In addition to loaders that come pre-packaged with the software, SpecViz also looks for custom loaders that you created and saved in your `~/specviz` directory, which can be created with the following Unix command:

```
$ mkdir ~/.specviz
```

To create your own loader, you can use either *Generic FITS Loader* or *ASCII Loader* as a template. Your YAML file can have any name of your choosing but must end with a `.yaml` extension.

In this section, we use a **MOSFIRE** spectrum named `spec1d.gds1312_H0.003.emp26177.fits` as an example of a custom FITS format for which we must create our own custom YAML definition file from the *Generic FITS Loader* template.

First, we inspect the file format that we have, as follow.

```
>>> from astropy.io import fits
>>> pf = fits.open('spec1d.gds1312_H0.003.emp26177.fits')
>>> pf.info()
Filename: spec1d.gds1312_H0.003.emp26177.fits
No.    Name          Type          Cards  Dimensions  Format
0     PRIMARY        PrimaryHDU    4       ()
1                      BinTableHDU  23      1R x 3C     [2287E, 2287E, 2287E]
```

This opens the FITS file and prints out the overall file structure. From this, it is obvious that the table is in Extension 1.

```
>>> pf[0].header
SIMPLE =                               T /Dummy Created by MWFITS v1.4a
BITPIX =                               8 /Dummy primary header created by MWFITS
NAXIS  =                               0 /No data is associated with this header
EXTEND =                               T /Extensions may (will!) be present
>>> pf[1].header
XTENSION= 'BINTABLE'                   /Binary table written by MWFITS v1.4a
BITPIX  =                               8 /Required value
...
TFORM3  = '2287E'                       /
```

This prints all the headers and we find no WCS information in either of the extensions.

```
>>> from astropy.table import Table
>>> tab = Table.read(pf[1], format='fits')
>>> print(tab)
FLUX [2287]      LAMBDA [2287]      IVAR [2287]
-----
0.0 .. 0.0989667 14500.0 .. 18223.8 1e-06 .. 422.54
```

This shows that there are three columns in the table in Extension 1, namely flux, wavelength, and inverse variance. The table has 2287 rows. Knowing the wavelength regime that the instrument is sensitive to and looking at the wavelength values, we can safely assume that the wavelength unit is Angstrom.

```
>>> from astropy import units as u
>>> u.electron / u.s / u.pix
Unit("electron / (pix s)")
```

However, the flux unit is not defined anywhere and cannot be easily inferred. So, let's just say that we already know the unit to be electrons/s/pix. The code above shows us how Astropy can ingest the flux unit that we want.

```
--- !CustomLoader
name: Keck/MOSFIRE Fits
extension: [fits, mits]
wcs:
  hdu: 0
dispersion:
  hdu: 1
  col: 1
  unit: 'Angstrom'
data:
  hdu: 1
  col: 0
  unit: 'electron / (pix s)'
uncertainty:
  hdu: 1
  col: 2
  type: 'ivar'
meta:
  author: STScI
```

Now that we have the format figured out, it is time to write our own YAML file for it. We will name it `keck_mosfire.yaml`.

```
--- !CustomLoader
```

The first line states that this YAML file defines our custom loader.

```
name: Keck/MOSFIRE Fits
extension: [fits, mits]
```

These two lines define the format name and accepted extensions, respectively. We will keep the extensions from our FITS template but change the name to identify our new format. In SpecViz GUI, this will translate to “Keck/MOSFIRE Fits (\*.fits \*.mits)” in the file type drop-down menu.

```
wcs:
  hdu: 0
```

We do not have WCS nor BUNIT defined, so we will simply leave this the same as our template.

```
dispersion:
  hdu: 1
  col: 1
  unit: 'Angstrom'
```

This instructs the loader to look for our wavelength values in Extension 1, the second column. We explicitly set its unit to Angstrom.

```
data:
  hdu: 1
  col: 0
  unit: 'electron / (pix s)'
```

This instructs the loader to look for our flux values in Extension 1, the first column, like the template. However, we also explicitly set its unit to electrons/s/pix by providing the appropriate Astropy unit name.

```
uncertainty:
  hdu: 1
  col: 2
  type: 'ivar'
```

This instructs the loader to look for flux uncertainty values in Extension 1, the third column. Unlike the template, we define it as inverse variance.

```
meta:
  author: STScI
```

Since this does not affect how SpecViz works, we do the lazy thing here by leaving it the same as our template.

Once you are done writing your YAML file, be sure to save it in `~/specviz`. Next, start SpecViz as usual. Now, in its open file dialog, you will see your new format listed in the file-type drop-down menu.

## **Part IV**

# **References/API**



## 6.1 Analysis Functions

### 6.1.1 Functions

<code>centroid(line, avg_cont[, mask])</code>	Compute centroid for the given spectrum.
<code>eq_width(cont1_stats, cont2_stats, line[, mask])</code>	Compute an equivalent width given stats for two continuum regions, and a <code>specutils.core.generic.Spectrum1DRef</code> instance with the extracted spectral line region.
<code>extract(data, x_range)</code>	Extract a region from a spectrum.
<code>fwzi(cont1_stats, cont2_stats, line)</code>	Compute full width at zero intensity (FWZI) for the given spectrum.
<code>resample(data_in, x_in, x_out, y[, ...])</code>	Resample the data of one spectrum using interpolation.
<code>stats(data[, mask])</code>	Compute basic statistics for a spectral region contained in a <code>specutils.core.generic.Spectrum1DRef</code> instance.

#### centroid

`specviz.analysis.centroid(line, avg_cont, mask=None)`

Compute centroid for the given spectrum.

```
w_cen = integral(wave*flux) / integral(flux)
```

#### Parameters

**data** : `specutils.core.generic.Spectrum1DRef`

Extracted spectrum data.

#### Returns

**wcen** : float

Centroid wavelength.

### Examples

```
>>> d = Spectrum1DRef(...)
>>> line = extract(d, (15000, 20000))
>>> wcen_em = centroid(line)
```

### eq\_width

specviz.analysis.**eq\_width**(*cont1\_stats, cont2\_stats, line, mask=None*)

Compute an equivalent width given stats for two continuum regions, and a `specutils.core.generic.Spectrum1DRef` instance with the extracted spectral line region.

This uses for now a very simple continuum subtraction method; i.e., it just subtracts a constant from the line spectrum, where the constant is  $(\text{continuum1}[\text{mean}] + \text{continuum2}[\text{mean}]) / 2$ .

#### Parameters

**cont1\_stats, cont2\_stats** : dict

This is returned by the `stats()` function.

**line** : `Spectrum1DRefLayer`

This is returned by the `extract()` function.

**mask** : ndarray

Boolean mask.

#### Returns

**ew, flux, avg\_cont** : float

Flux and equivalent width values.

### Examples

```
>>> d = Spectrum1DRefLayer(...)
>>> cont1 = extract(d, (100, 5000))
>>> cont2 = extract(d, (18000, 20000))
>>> cont1_stats = stats(cont1)
>>> cont2_stats = stats(cont2)
>>> line = extract(d, (15000, 18000))
>>> flux, ew = eq_width(cont1_stats, cont2_stats, line)
```

### extract

specviz.analysis.**extract**(*data, x\_range*)

Extract a region from a spectrum.

#### Parameters

**data** : `specutils.core.generic.Spectrum1DRef`

Contains the spectrum to be extracted.

**x\_range** : tuple

A spectral coordinate range as in (wave1, wave2).

### Returns

**result** : `specutils.core.generic.Spectrum1DRef`

Spectrum data with extracted region.

### Examples

```
>>> d = Spectrum1DRef(...)
>>> d2 = extract(d, (10000, 20000))
```

### fwzi

`specviz.analysis.fwzi(cont1_stats, cont2_stats, line)`

Compute full width at zero intensity (FWZI) for the given spectrum. Continuum calculations are similar to `eq_width()`.

#### Parameters

**cont1\_stats, cont2\_stats** : dict

This is returned by the `stats()` function.

**line** : `specutils.core.generic.Spectrum1DRef`

This is returned by the `extract()` function.

#### Returns

**fwzi\_value** : float

FWZI value.

**w\_range** : tuple

Wavelengths used to calculate FWZI.

### Examples

```
>>> d = Spectrum1DRef(...)
>>> cont1 = extract(d, (100, 5000))
>>> cont2 = extract(d, (18000, 20000))
>>> cont1_stats = stats(cont1)
>>> cont2_stats = stats(cont2)
>>> line = extract(d, (15000, 18000))
>>> fwzi_value, w_range = fwzi(cont1_stats, cont2_stats, line)
```

### resample

`specviz.analysis.resample(data_in, x_in, x_out, y, data_out=None, kind='linear')`

Resample the data of one spectrum using interpolation.

Dependent variables  $y_1, y_2, \dots$  in the input data are resampled in the independent variable  $x$  using interpolation models  $y_1(x), y_2(x), \dots$  evaluated on a new grid of  $x$  values. The independent variable will typically be a wavelength or frequency and the independent variables can be fluxes, inverse variances, etc.

Interpolation is intended for cases where the input and output grids have comparable densities. When neighboring samples are correlated, the resampling process should be essentially lossless. When the output grid is sparser than the input grid, it may be more appropriate to “downsample”, i.e., average dependent variables over consecutive ranges of input samples.

The basic usage of this function is:

```
>>> data = np.ones((5,),
... [('wlen', float), ('flux', float), ('ivar', float)])
>>> data['wlen'] = np.arange(4000, 5000, 200)
>>> wlen_out = np.arange(4100, 4700, 200)
>>> resample(data, 'wlen', wlen_out, ('flux', 'ivar'))
array([(4100, 1.0, 1.0), (4300, 1.0, 1.0), (4500, 1.0, 1.0)],
      dtype=[('wlen', '<i8'), ('flux', '<f8'), ('ivar', '<f8')])
```

The input grid can also be third\_party to the structured array of spectral data, for example:

```
>>> data = np.ones((5,), [('flux', float), ('ivar', float)])
>>> wlen_in = np.arange(4000, 5000, 200)
>>> wlen_out = np.arange(4100, 4900, 200)
>>> resample(data, wlen_in, wlen_out, ('flux', 'ivar'))
array([(1.0, 1.0), (1.0, 1.0), (1.0, 1.0), (1.0, 1.0)],
      dtype=[('flux', '<f8'), ('ivar', '<f8')])
```

If the output grid extends beyond the input grid, a [masked array](#) will be returned with any values requiring extrapolation masked:

```
>>> wlen_out = np.arange(3500, 5500, 500)
>>> resample(data, wlen_in, wlen_out, 'flux')
masked_array(data = [(--,) (1.0,) (1.0,) (--,)],
             mask = [(True,) (False,) (False,) (True,)],
             fill_value = (1e+20,),
             dtype = [('flux', '<f8')])
```

If the input data is masked, any output interpolated values that depend on an input masked value will be masked in the output:

```
>>> data = ma.ones((5,), [('flux', float), ('ivar', float)])
>>> data['flux'][2] = ma.masked
>>> wlen_out = np.arange(4100, 4900, 200)
>>> resample(data, wlen_in, wlen_out, 'flux')
masked_array(data = [(1.0,) (--,) (--,) (1.0,)],
             mask = [(False,) (True,) (True,) (False,)],
             fill_value = (1e+20,),
             dtype = [('flux', '<f8')])
```

Interpolation is performed using `scipy.interpolate.inter1pd`.

#### Parameters

**data\_in** : numpy.ndarray or numpy.ma.MaskedArray

Structured numpy array of input spectral data to resample. The input array must be one-dimensional.

**x\_in** : string or numpy.ndarray

A field name in data\_in containing the independent variable to use for interpolation, or else an array of values with the same shape as the input data.

**x\_out** : numpy.ndarray

An array of values for the independent variable where interpolation models should be evaluated to calculate the output values.

**y** : string or iterable of strings.

A field name or a list of field names present in the input data that should be resampled by interpolation and included in the output.

**data\_out** : numpy.ndarray or None

Structured numpy array where the output result should be written. If None is specified, then an appropriately sized array will be allocated and returned. Use this method to take control of the memory allocation and, for example, re-use the same array when resampling many spectra.

**kind** : string or integer

Specify the kind of interpolation models to build using any of the forms allowed by `scipy.interpolate.inter1pd`. If any input dependent values are masked, only the nearest` and ``linear values are allowed.

### Returns

numpy.ndarray or numpy.ma.MaskedArray

Structured numpy array of the resampled result containing all y fields and (if x\_in is specified as a string) the output x field. The output will be a `numpy.ma.MaskedArray` if x\_out extends beyond x\_in or if data\_in is masked.

## stats

`specviz.analysis.stats(data, mask=None)`

Compute basic statistics for a spectral region contained in a `specutils.core.generic.Spectrum1DRef` instance.

### Parameters

**data** : `specutils.core.generic.Spectrum1DRef`

Typically this is returned by the `extract()` function.

### Returns

**statistics** : dict

Statistics results.

## Examples

```
>>> d = Spectrum1DRef(...)
>>> d_stats = stats(d)
```

## 6.1.2 Classes

<code>BlackBody</code>	Produce a blackbody flux spectrum.
<code>Spline1D</code>	Perform a spline fit

## BlackBody

**class** `specviz.analysis.BlackBody`

Bases: `astropy.modeling.Fittable1DModel`

Produce a blackbody flux spectrum.

Note that the wave and flux arrays used to Quantity

### Notes

See `Fittable1DModel` for further details on modeling and all possible parameters that can be passed in.

Description of the blackbody function itself is described in `blackbody`

### Attributes Summary

---

`norm`

---

`param_names`

---

`temp`

---

### Methods Summary

---

`evaluate(x, temp, norm)`

Evaluate the blackbody for a given temperature over a wavelength range.

---

### Attributes Documentation

`norm`

`param_names = ('temp', 'norm')`

`temp`

### Methods Documentation

`evaluate(x, temp, norm)`

Evaluate the blackbody for a given temperature over a wavelength range.

#### Parameters

**x: `numpy.ndarray`**

The wavelengths to evaluate over.

**temp: `float`**

The temperature to evaluate at.

**norm: `float`**

The normalization factor.

**Returns**

`blackbody_flux`: `numpy.ndarray`

The blackbody flux.

**Spline1D**

**class** `specviz.analysis.Spline1D`

Bases: `astropy.modeling.Fittable1DModel`

Perform a spline fit

**Notes**

See `Fittable1DModel` for further details.

The spline function is based on `UnivariateSpline`

**Attributes Summary**


---

`degree`

`param_names`

`smooth`

---

**Methods Summary**


---

`evaluate(x, degree, smooth)`

Evaluate the spline

---

**Attributes Documentation**

**degree**

`param_names = ('degree', 'smooth')`

**smooth**

**Methods Documentation**

**evaluate**(*x, degree, smooth*)

Evaluate the spline

**Parameters**

**x**: `numpy.ndarray`

The wavelengths to evaluate over.

**degree**: `int`

The degree of spline to evaluate.

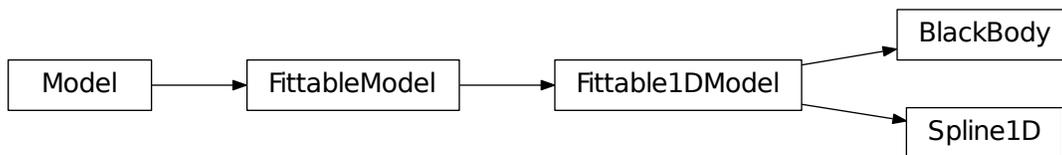
**smooth:** float or None

The smoothing factor used to choose the number of knots.

**Returns**

The evaluated spline

### 6.1.3 Class Inheritance Diagram



## 6.2 SpecViz core

### 6.2.1 Data Objects

Data Objects

### 6.2.2 Classes

<code>Spectrum1DRefLayer(data[, wcs, parent, ...])</code>	Class to handle layers in SpecViz.
<code>Spectrum1DRefModelLayer(data[, model])</code>	A layer for spectrum with a model applied.

#### Spectrum1DRefLayer

```

class specviz.core.data.Spectrum1DRefLayer(data, wcs=None, parent=None, layer_mask=None,
                                           uncertainty=None, unit=None, mask=None, *args,
                                           **kwargs)
  
```

Bases: `specutils.core.generic.Spectrum1DRef`

Class to handle layers in SpecViz.

#### Parameters

**data:** `numpy.ndarray`

The flux.

**wcs:** `~astropy.wcs.WCS`

If specified, the WCS relating pixel to wavelength.

**parent:** layer

If specified, the parent layer.

**layer\_mask: layer**

The layer defining the valid data mask.

**args, kwargs:**

Arguments passed to the Spectrum1DRef object.

**Attributes Summary**

<code>full_mask</code>	Mask for spectrum data.
<code>layer_mask</code>	Mask applied from an ROI.
<code>masked_data</code>	Flux quantity with mask applied.
<code>masked_dispersion</code>	Dispersion quantity with mask applied.
<code>raw_uncertainty</code>	Flux uncertainty with mask applied.
<code>shape</code>	
<code>unit</code>	
<code>unmasked_data</code>	Flux quantity with no layer mask applied.
<code>unmasked_dispersion</code>	Dispersion quantity with no layer mask applied.
<code>unmasked_raw_uncertainty</code>	Flux uncertainty with mask applied.

**Methods Summary**

<code>from_formula(formula, layers)</code>	Create a layer from an operation performed on other layers
<code>from_parent(parent[, layer_mask, name, copy])</code>	Create a duplicate child layer from a parent layer
<code>from_self([name, layer_mask])</code>	Create a new, parentless, layer based on this layer
<code>set_units(disp_unit, data_unit)</code>	Set the dispersion and flux units

**Attributes Documentation****full\_mask**

Mask for spectrum data.

**layer\_mask**

Mask applied from an ROI.

**masked\_data**

Flux quantity with mask applied. Returns a masked array containing a Quantity object.

**masked\_dispersion**

Dispersion quantity with mask applied. Returns a masked array containing a Quantity object.

**raw\_uncertainty**

Flux uncertainty with mask applied. Returns a masked array containing a Quantity object.

**shape****unit****unmasked\_data**

Flux quantity with no layer mask applied.

**unmasked\_dispersion**

Dispersion quantity with no layer mask applied.

**unmasked\_raw\_uncertainty**

Flux uncertainty with mask applied. Returns a masked array containing a Quantity object.

**Methods Documentation**

**classmethod from\_formula**(*formula, layers*)

Create a layer from an operation performed on other layers

**Parameters**

**formula: str**

The operation to perform on the given layers.

**layers: [layer, ...]**

The layers which are arguments to the given formula.

**Returns**

*new\_layer*:

Result of the operation

**classmethod from\_parent**(*parent, layer\_mask=None, name=None, copy=True*)

Create a duplicate child layer from a parent layer

**Parameters**

**parent: layer**

The layer to duplicate.

**layer\_mask: layer**

The layer defining the valid data mask.

**name: str**

Layer's name. If *None*, a name based on the parent layer is used.

**Returns**

*child\_layer*:

The new layer.

**from\_self**(*name="u", layer\_mask=None*)

Create a new, parentless, layer based on this layer

**Parameters**

**name: str**

Name of the new layer

**layer\_mask: layer**

The layer defining the valid data mask.

**Returns**

*new\_layer*:

The new, parentless, layer.

**set\_units**(*disp\_unit, data\_unit*)

Set the dispersion and flux units

**Parameters****disp\_unit:** ‘~astropy.units‘

The dispersion units.

**data\_unit:** ‘~astropy.units‘

The flux units.

**Spectrum1DRefModelLayer****class** specviz.core.data.Spectrum1DRefModelLayer(*data*, *model=None*, \**args*, \*\**kwargs*)

Bases: specviz.core.data.Spectrum1DRefLayer

A layer for spectrum with a model applied.

**Parameters****data:** numpy.ndarray

The flux.

**model:** ‘~astropy.modeling‘

The model

**args, kwargs:**

Arguments passed to the Spectrum1DRef object.

**Attributes Summary**

<code>model</code>	Spectrum model.
<code>parent_mask</code>	A bitwise combination of the data mask and the <code>Spectrum1DRefModelLayer</code> ’s parent’s layer mask.
<code>unmasked_data</code>	Flux quantity with no layer mask applied.
<code>unmasked_dispersion</code>	Dispersion quantity with no layer mask applied.
<code>unmasked_raw_uncertainty</code>	Flux uncertainty with mask applied.

**Methods Summary**

<code>from_formula(models, formula)</code>	Create a layer from an operation performed on other models
<code>from_parent(parent[, model, layer_mask, copy])</code>	Create a duplicate child layer from a parent layer

**Attributes Documentation****model**

Spectrum model.

**parent\_mask**

A bitwise combination of the data mask and the `Spectrum1DRefModelLayer`’s parent’s layer mask. This is useful when dealing with slices of spectra in which you want the model layer to be the visible size of the parent *in all cases* (whereas `full_mask` will always be the selected region)\*[]:

**unmasked\_data**

Flux quantity with no layer mask applied. Use the parent layer mask for cases wherein a slice of the spectrum is being used.

**unmasked\_dispersion**

Dispersion quantity with no layer mask applied. Use the parent layer mask for cases wherein a slice of the spectrum is being used.

**unmasked\_raw\_uncertainty**

Flux uncertainty with mask applied. Returns a masked array containing a Quantity object. Use the parent layer mask for cases wherein a slice of the spectrum is being used.

## Methods Documentation

**classmethod from\_formula**(*models, formula*)

Create a layer from an operation performed on other models

**Parameters**

**formula: str**

The operation to perform on the given layers.

**models: [model, ...]**

The models which are arguments to the given formula.

**Returns**

result\_model:

Result of the operation

**classmethod from\_parent**(*parent, model=None, layer\_mask=None, copy=False*)

Create a duplicate child layer from a parent layer

**Parameters**

**parent: layer**

The layer to duplicate.

**model: '~astropy.modeling'**

The model.

**layer\_mask: layer**

The layer defining the valid data mask.

**copy : bool**

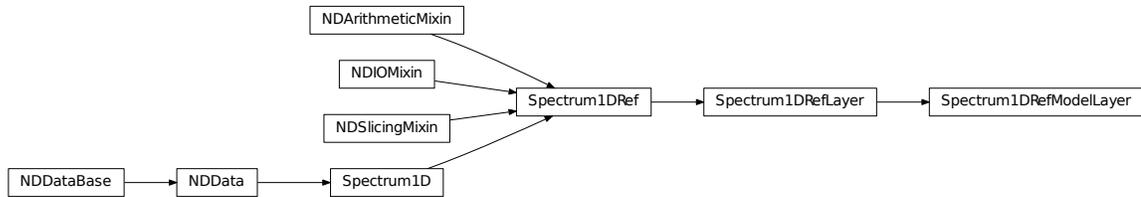
Copy the model if one is provided.

**Returns**

child\_layer:

The new layer.

### 6.2.3 Class Inheritance Diagram



### 6.2.4 Object Event Handling

### 6.2.5 Object Dispatch Machinery

#### Object Event Handling

The singleton `Dispatch` object manages the set of `EventNode` events. Handlers or **listeners** are attached to `EventNode`'s. The `DispatchHandle` decorator is used to decorate classes that handle events.

### 6.2.6 Classes

<code>Dispatch</code>	Central communications object for all events.
<code>EventNode(*args)</code>	An event

#### Dispatch

```
class specviz.core.dispatch.Dispatch
```

Bases: `object`

Central communications object for all events.

#### Methods Summary

<code>register_event(name[, args])</code>	Add an <code>EventNode</code> to the list of possible events
<code>register_listener(*args)</code>	Decorate event listeners
<code>setup(inst)</code>	Register all methods decorated by <code>register_listener</code>
<code>tear_down(inst)</code>	Remove all registered methods from their events
<code>unregister_listener(name, func)</code>	Remove a listener from an event

#### Methods Documentation

```
register_event(name, args=None)
```

Add an `EventNode` to the list of possible events

**Parameters****name: str**

The name of the event.

**args: [arg, ...]**

The list of keyword arguments this event will pass to its handlers.

**register\_listener(\*args)**

Decorate event listeners

**setup(inst)**Register all methods decorated by `register_listener`**tear\_down(inst)**

Remove all registered methods from their events

**unregister\_listener(name, func)**

Remove a listener from an event

**Parameters****name: str**

The event from which the listener should be removed.

**func: function**

The function to be removed

**EventNode****class** `specviz.core.dispatch.EventNode(*args)`Bases: `object`

An event

An event is defined by the arguments the listeners of the event expect to be given.

**Parameters****args: [arg, ...]**

The list of keyword arguments that the event will provide to its listeners

**Methods Summary**

---

<code>clear()</code>	Removes all handlers from object.
<code>emit(*args, **kwargs)</code>	Call the handlers of this event

---

**Methods Documentation****clear()**

Removes all handlers from object.

**emit(\*args, \*\*kwargs)**

Call the handlers of this event

**Parameters****args: [arg, ...]**

The keyword arguments being provided to the event.

**kwargs:** {arg: value, ...}

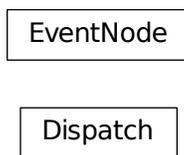
The keyword/value pairs passed to the listeners.

**Raises**

**ValueError**

An keyword is being passed that does not belong to this event.

## 6.2.7 Class Inheritance Diagram



## 6.2.8 Spectrum Layer Plotting

Spectrum Layer Plotting

## 6.2.9 Classes

---

<code>LinePlot(layer[, plot, visible, style, pen, ...])</code>	Plot representation of a layer
--	--------------------------------

---

### LinePlot

**class** `specviz.core.plots.LinePlot`(*layer*, *plot=None*, *visible=True*, *style=u'line'*, *pen=None*, *err\_pen=None*, *mask\_pen=None*, *color=(0, 0, 0)*)

Bases: `object`

Plot representation of a layer

#### Parameters

**layer:** 'Spectrum1DRefLayer'

The layer to plot

**plot:** `LinePlot`

`LinePlot` instance to reuse.

**visible:** `bool`

If `True`, the plot will be visible

**style:** `str`

The plotting style

**pen: str**

If defined, the pen style to use.

**err\_pen: str**

If defined, the pen style to use for the error/uncertainty.

### Attributes Summary

---

error_pen
layer
mask_pen
pen
plot

---

### Methods Summary

---

change_units(x[, y, z])	Change the displayed units
from_layer(layer, **kwargs)	Create a LinePlot from a layer
set_error_visibility([show])	Show the error/uncertainty
set_line_width(width)	Set the line plot width
set_mask_visibility([show])	Show masked data
set_mode(mode)	Set the line plotting mode
set_plot_visibility([show, inactive])	Set visibility and active state
update([autoscale])	Refresh the plot

---

### Attributes Documentation

**error\_pen**

**layer**

**mask\_pen**

**pen**

**plot**

### Methods Documentation

**change\_units**(x, y=None, z=None)

Change the displayed units

**Parameters**

**x:** ‘~astropy.units‘

The new units for the dispersion

**y:** ‘~astropy.units‘

The new units for the flux

**z:** ‘~astropy.units‘

The new units for the multi-spectral dimension.

**static from\_layer**(*layer*, *\*\*kwargs*)

Create a LinePlot from a layer

**Parameters**

**layer:** ‘Spectrum1DRefLayer‘

The layer to create from.

**kwargs:** dict

Other arguments for `LinePlot` class.

**Returns**

plot\_container:

The new LinePlot

**set\_error\_visibility**(*show=None*)

Show the error/uncertainty

**Parameters**

**show:** bool

If True, show the error/uncertainty info.

**set\_line\_width**(*width*)

Set the line plot width

**Parameters**

**width:** float

The width of the line

**set\_mask\_visibility**(*show=None*)

Show masked data

**Parameters**

**show:** bool

If True, display data points with mask value True.

**set\_mode**(*mode*)

Set the line plotting mode

**Parameters**

**mode:** ‘line’ | ‘scatter’ | ‘histogram’

The plot mode

**set\_plot\_visibility**(*show=None, inactive=None*)

Set visibility and active state

**Parameters**

**show:** bool

If True, show the plot

**inactive: bool**

If True, set plot style to indicate this is not the active plot.

**update**(*autoscale=False*)

Refresh the plot

**Parameters**

**autoscale: bool**

If True, rescale the plot to match the data.

## 6.2.10 Class Inheritance Diagram



## 6.2.11 Emission/Absorption Line list utilities

Emission/Absorption Line list utilities

## 6.2.12 Functions

---

<code>ingest(range)</code>	Returns a list with LineList instances.
----------------------------	---

---

### ingest

`specviz.core.linelist.ingest(range)`

Returns a list with LineList instances.

Each original list is stripped out of lines that lie outside the wavelength range.

**Parameters**

**range:**

The wavelength range of interest.

**Returns**

[LineList, ...]

The list of linelists found.

### Notes

Lets skip the file dialog business. For now, we look for line lists and their accompanying YAML files in one single place. We also restrict our search for ascii line lists whose file names end in .txt

## 6.2.13 Classes

---

<code>LineList([table, name, masked])</code>	A list of emission/absorption lines
--	-------------------------------------

---

### LineList

**class** `specviz.core.linelist.LineList`(*table=None, name=None, masked=None*)

Bases: `astropy.table.Table`

A list of emission/absorption lines

#### Parameters

**table:** ‘~`astropy.table.Table`’

If specified, a table to initialize from.

**name:** `str`

The name of the list.

**masked:** `bool`

If true, a masked table is used.

#### Methods Summary

---

<code>extract_range(wrange)</code>	Builds a <code>LineList</code> instance out of self, with
<code>extract_rows(indices)</code>	Builds a <code>LineList</code> instance out of self, with
<code>merge(lists)</code>	Executes a ‘ <code>vstack</code> ’ of all input lists, and then sorts the result by the wavelength column.

---

#### Methods Documentation

**extract\_range**(*wrange*)

Builds a `LineList` instance out of self, with the subset of lines that fall within the wavelength range defined by ‘`wmin`’ and ‘`wmax`’

#### Parameters

**wrange:** (`float, float`)

minimum and maximum wavelength of the wavelength range

#### Returns

`LineList`

line list with subset of lines

**extract\_rows**(*indices*)

Builds a `LineList` instance out of self, with the subset of lines pointed by ‘`indices`’

#### Parameters

**indices:** [`QModelIndex, ...`]

List of `QModelIndex` instances to extract from.

#### Returns

`LineList`

line list with subset of lines

**classmethod** `merge`(*lists*)

Executes a ‘vstack’ of all input lists, and then sorts the result by the wavelength column.

**Parameters**

**lists:** [`LineList`, ...]

list of `LineList` instances

**Returns**

`LineList`

merged line list

## 6.2.14 Class Inheritance Diagram



## 6.2.15 Thread Helpers

Thread Helpers

## 6.2.16 Classes

<code>FileLoadThread</code> ( <i>parent</i> )	Asynchronously read in a file
<code>FitModelThread</code> ( <i>parent</i> )	Asynchronously fit a model to a layer

### FileLoadThread

**class** `specviz.core.threads.FileLoadThread`(*parent=None*)

Bases: `PyQt5.QtCore.QThread`

Asynchronously read in a file

**Parameters**

**parent:** `QWidget`

The parent widget or None

**Attributes**

<b>file_name:</b> <code>str</code>	Name of the file to read.
<b>file_filter:</b> <code>str</code>	Type of file to read. If Auto, try all known formats.

## Attributes Summary

---

result  
status

---

## Methods Summary

<code>__call__(file_name, file_filter)</code>	Initialize the thread
<code>read_file(file_name, file_filter)</code>	Convenience method that directly reads a spectrum from a file.
<code>run()</code>	Start thread to read the file.

## Attributes Documentation

**result**

**status**

## Methods Documentation

`__call__(file_name, file_filter)`

Initialize the thread

`read_file(file_name, file_filter)`

Convenience method that directly reads a spectrum from a file.

### Parameters

**file\_name: str**

Name of the file to read.

**file\_filter: str**

Type of file to read. If Auto, try all known formats.

### Returns

data: Spectrum1DRef

The file's data or None if no known formats are found.

## Notes

This exists mostly to facilitate development workflow. In time it could be augmented to support fancier features such as wildcards, file lists, mixed file types, and the like. Note that the filter string is hard coded here; its details might depend on the intricacies of the registries, loaders, and data classes. In other words, this is brittle code.

`run()`

Start thread to read the file.

## FitModelThread

**class** specviz.core.threads.**FitModelThread**(parent=None)

Bases: PyQt5.QtCore.QThread

Asynchronously fit a model to a layer

### Parameters

**parent: QWidget**

The parent widget or None

### Attributes

<b>model_layer: Spectrum1DRefLayer</b>	The layer to fit to.
<b>fitter_name: An '~astropy.modeling' fitter</b>	The fitter to use
<b>mask: numpy.ndarray</b>	The mask to apply

### Attributes Summary

---

result

---

status

---

### Methods Summary

---

`__call__(model_layer, fitter_name[, mask, ...])`

---

`fit_model(model_layer, fitter_name[, mask, ...])` Fit the model

---

`run()` Start thread to fit the model

---

### Attributes Documentation

**result**

**status**

### Methods Documentation

`__call__(model_layer, fitter_name, mask=None, kwargs=None)`

`fit_model(model_layer, fitter_name, mask=None, kwargs=None)`

Fit the model

#### Parameters

**model\_layer: Spectrum1DRefLayer**

The layer to fit to.

**fitter\_name: An '~astropy.modeling' fitter**

The fitter to use

**mask:** `numpy.ndarray`

The mask to apply

**Returns**

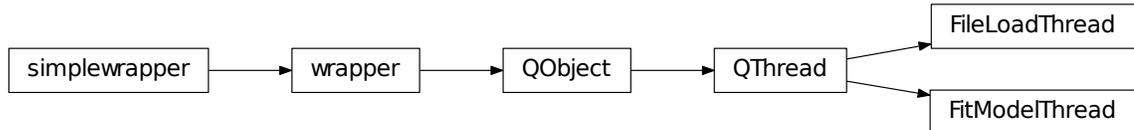
(`model_layer`, `fitter_message`): `Spectrum1DRefLayer`, `str`

The `model_layer.model` is updated with the fit parameters. The message is from the fitter itself.

**run()**

Start thread to fit the model

## 6.2.17 Class Inheritance Diagram



## 6.3 Interfaces

### 6.3.1 Functions

---

`data_loader`(`label`, `identifier`[, `priority`, ...])

A decorator that registers a function and identifies with an Astropy io registry object.

---

#### `data_loader`

`specviz.interfaces.data_loader`(`label`, `identifier`, `priority=-1`, `extensions=None`, `writer=None`, `**kwargs`)

A decorator that registers a function and identifies with an Astropy io registry object.

Priority will be assigned as an attribute to the wrapped function for use in the auto loader. Loaders with high numerical value will be tried first, followed by low numerical value in order.

**Parameters**

**label** : `str`

user-friendly name for the data loader

**identifier** : `function`

function used to determine if the loader should be used on the Input

**priority** : `int`

absolute priority to determine which loader to attempt first

## 6.3.2 Model Factories

App-wide factories

## 6.3.3 Classes

---

Factory	Responsible for creation of objects.
FitterFactory	Create a fitter
ModelFactory	Create a model

---

### Factory

**class** `specviz.interfaces.factories.Factory`

Bases: `object`

Responsible for creation of objects.

### FitterFactory

**class** `specviz.interfaces.factories.FitterFactory`

Bases: `specviz.interfaces.factories.Factory`

Create a fitter

### Attributes Summary

---

`all_fitters`

---

### Methods Summary

---

<code>create_fitter(name)</code>	Create a fitter
----------------------------------	-----------------

---

### Attributes Documentation

`all_fitters = {u'Levenberg-Marquardt': <class 'astropy.modeling.fitting.LevMarLSQFitter'>, u'Simplex': <`

### Methods Documentation

**classmethod** `create_fitter(name)`

Create a fitter

#### Parameters

**name:** `str`

The name of the fitter desired.

#### Returns

fitter: `Fitter`

The fitter class requested. None if the requested fitter does not exist.

## ModelFactory

**class** specviz.interfaces.factories.**ModelFactory**

Bases: specviz.interfaces.factories.Factory

Create a model

### Notes

Ideally we should be getting these classes from astropy directly and transparently, instead of explicitly naming them here. This is basically a maintenance issue: at each new release of astropy we should check if new models became available, and existing models got deprecated.

This might not be possible unless astropy itself somehow manages to further subclass its Fittable1DModel class into spectrum-specific and other types. Right now we have a mix of spectral models, galaxy surface brightness models, and others, all lumped into a single type. Thus we have to keep picking the spectral relevant types by hand for the time being.

### Attributes Summary

---

`all_models`

---

### Methods Summary

---

<code>create_model(name)</code>	Create a model
---------------------------------	----------------

---

### Attributes Documentation

`all_models = {u'Box': <class 'astropy.modeling.functional_models.Box1D'> Name: Box1D Inputs: (u'x',) Out:`

### Methods Documentation

**classmethod** `create_model(name)`

Create a model

#### Parameters

**name:** str

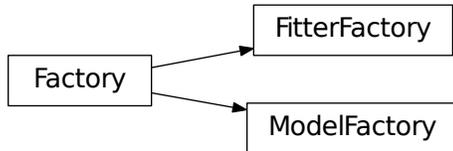
The name of the model desired.

#### Returns

model: models

The requested model. None if the requested model does not exist.

### 6.3.4 Class Inheritance Diagram



### 6.3.5 Model Initialization

This module is used to initialize spectral models to the data at hand.

This is used by model-fitting code that has to create spectral model instances with sensible parameter values such that they can be used as first guesses by the fitting algorithms.

### 6.3.6 Functions

---

<code>initialize(instance, x, y)</code>	Initialize given model.
---	-------------------------

---

#### initialize

`specviz.interfaces.initializers.initialize(instance, x, y)`

Initialize given model.

X and Y are for now Quantity arrays with the independent and dependent variables. It's assumed X values are stored in increasing order in the array.

#### Parameters

**instance:** `~astropy.modeling.models`

The model to initialize.

**x, y:** `numpy.ndarray`

The data to use to initialize from.

#### Returns

instance: `models`

The initialized model. If there are any errors, the instance is returned uninitialized.

## 6.4 I/O module

### 6.4.1 Functions

<code>apStar_identify(*args, **kwargs)</code>	Check whether given filename is FITS.
<code>apStar_loader(*args, **kwargs)</code>	Loader for APOGEE apStar files.
<code>apVisit_identify(*args, **kwargs)</code>	Check whether given filename is FITS.
<code>apVisit_loader(*args, **kwargs)</code>	Loader for APOGEE apVisit files.
<code>aspcapStar_identify(*args, **kwargs)</code>	Check whether given filename is FITS.
<code>aspcapStar_loader(*args, **kwargs)</code>	Loader for APOGEE aspcapStar files.
<code>cos_identify(*args, **kwargs)</code>	Check whether given file contains HST/COS spectral data.
<code>cos_spectrum_loader(*args, **kwargs)</code>	Load file from COS spectral data into a spectrum object
<code>fits_identify(*args, **kwargs)</code>	Check whether given filename is FITS.
<code>simple_generic_loader(*args, **kwargs)</code>	Basic FITS file loader
<code>simple_generic_writer(data, file_name, **kwargs)</code>	Basic Spectrum1DRef FITS writer.
<code>stis_identify(*args, **kwargs)</code>	Check whether given file contains HST/STIS spectral data.
<code>stis_spectrum_loader(*args, **kwargs)</code>	Load file from STIS spectral data into a spectrum object

### apStar\_identify

`specviz.io.apStar_identify(*args, **kwargs)`

Check whether given filename is FITS. This is used for Astropy I/O Registry.

### apStar\_loader

`specviz.io.apStar_loader(*args, **kwargs)`

Loader for APOGEE apStar files.

#### Parameters

**file\_name: str**

The path to the FITS file

#### Returns

data: Spectrum1DRef

The data.

### apVisit\_identify

`specviz.io.apVisit_identify(*args, **kwargs)`

Check whether given filename is FITS. This is used for Astropy I/O Registry.

### apVisit\_loader

`specviz.io.apVisit_loader(*args, **kwargs)`

Loader for APOGEE apVisit files.

#### Parameters

**file\_name: str**

The path to the FITS file

#### Returns

data: Spectrum1DRef

The data.

### aspcapStar\_identify

`specviz.io.aspcapStar_identify(*args, **kwargs)`

Check whether given filename is FITS. This is used for Astropy I/O Registry.

### aspcapStar\_loader

`specviz.io.aspcapStar_loader(*args, **kwargs)`

Loader for APOGEE aspcapStar files.

#### Parameters

**file\_name: str**

The path to the FITS file

#### Returns

data: Spectrum1DRef

The data.

### cos\_identify

`specviz.io.cos_identify(*args, **kwargs)`

Check whether given file contains HST/COS spectral data.

### cos\_spectrum\_loader

`specviz.io.cos_spectrum_loader(*args, **kwargs)`

Load file from COS spectral data into a spectrum object

#### Parameters

**file\_name: str**

The path to the FITS file

#### Returns

data: Spectrum1DRef

The data.

### fits\_identify

`specviz.io.fits_identify(*args, **kwargs)`

Check whether given filename is FITS. This is used for Astropy I/O Registry.

### simple\_generic\_loader

`specviz.io.simple_generic_loader(*args, **kwargs)`

Basic FITS file loader

Presumption is the primary data is a table with columns 'flux' and 'err'. The dispersion information is encoded in the FITS header keywords.

#### Parameters

**file\_name: str**

The path to the FITS file

**Returns**

data: Spectrum1DRef

The data.

### simple\_generic\_writer

specviz.io.**simple\_generic\_writer**(*data*, *file\_name*, **\*\*kwargs**)

Basic Spectrum1DRef FITS writer.

### stis\_identify

specviz.io.**stis\_identify**(*\*args*, **\*\*kwargs**)

Check whether given file contains HST/STIS spectral data.

### stis\_spectrum\_loader

specviz.io.**stis\_spectrum\_loader**(*\*args*, **\*\*kwargs**)

Load file from STIS spectral data into a spectrum object

**Parameters**

**file\_name: str**

The path to the FITS file

**Returns**

data: Spectrum1DRef

The data.

This module contains functions that perform the actual data parsing.

## 6.4.2 Classes

<code>AsciiYamlRegister</code> (reference)	Defines the generation of Spectrum1DRef objects by parsing ASCII files with information from YAML files.
<code>FitsYamlRegister</code> (reference)	Defines the generation of Spectrum1DRef objects by parsing FITS files with information from YAML files.
<code>LineListYamlRegister</code> (reference)	Defines the generation of Spectrum1DRef objects by parsing LineList files with information from YAML files.
<code>YamlRegister</code> (reference)	Class to encapsulate the IO registry information for a set of yaml-loaded attributes.

### AsciiYamlRegister

**class** specviz.io.yaml\_loader.**AsciiYamlRegister**(*reference*)

Bases: specviz.io.yaml\_loader.YamlRegister

Defines the generation of Spectrum1DRef objects by parsing ASCII files with information from YAML files.

Initialize this particular YamlRegister.

**Parameters**

**reference** : dict-like

The yaml reference object created by loading a yaml file.

## Methods Summary

---

<code>reader(filename, **kwargs)</code>	Like <code>fits_reader()</code> but for ASCII file.
---	---

---

## Methods Documentation

**reader** (*filename*, *\*\*kwargs*)

Like `fits_reader()` but for ASCII file.

## FitsYamlRegister

**class** `specviz.io.yaml_loader.FitsYamlRegister`(*reference*)

Bases: `specviz.io.yaml_loader.YamlRegister`

Defines the generation of `Spectrum1DRef` objects by parsing FITS files with information from YAML files.

Initialize this particular `YamlRegister`.

### Parameters

**reference** : dict-like

The yaml reference object created by loading a yaml file.

## Methods Summary

---

<code>reader(filename, **kwargs)</code>	This generic function will query the loader factory, which has already loaded the YAML configuration files, in an attempt to parse the associated FITS file.
---	--

---

## Methods Documentation

**reader** (*filename*, *\*\*kwargs*)

This generic function will query the loader factory, which has already loaded the YAML configuration files, in an attempt to parse the associated FITS file.

### Parameters

**filename** : str

Input filename.

**kwargs** : dict

Keywords for Astropy reader.

## LineListYamlRegister

**class** `specviz.io.yaml_loader.LineListYamlRegister`(*reference*)

Bases: `specviz.io.yaml_loader.YamlRegister`

Defines the generation of Spectrum1DRef objects by parsing LineList files with information from YAML files. Initialize this particular YamlRegister.

**Parameters**

**reference** : dict-like

The yaml reference object created by loading a yaml file.

**Methods Summary**

---

`reader(filename, **kwargs)`

---

**Methods Documentation**

`reader(filename, **kwargs)`

**YamlRegister**

**class** `specviz.io.yaml_loader.YamlRegister(reference)`

Bases: `object`

Class to encapsulate the IO registry information for a set of yaml-loaded attributes.

Initialize this particular YamlRegister.

**Parameters**

**reference** : dict-like

The yaml reference object created by loading a yaml file.

**Methods Summary**

---

`identify(*args, **kwargs)`

`reader(filename, **kwargs)`

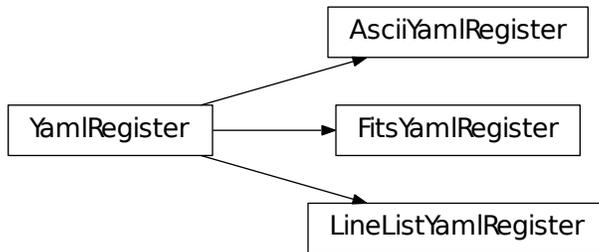
---

**Methods Documentation**

`identify(*args, **kwargs)`

`reader(filename, **kwargs)`

### 6.4.3 Class Inheritance Diagram



## **Part V**

# **Indices and tables**



- `genindex`
- `modindex`



### S

- `specviz.analysis`, 33
- `specviz.core.data`, 40
- `specviz.core.dispatch`, 45
- `specviz.core.events`, 45
- `specviz.core.linelist`, 50
- `specviz.core.plots`, 47
- `specviz.core.threads`, 52
- `specviz.interfaces`, 55
  - `specviz.interfaces.factories`, 56
  - `specviz.interfaces.initializers`, 58
- `specviz.io`, 58
  - `specviz.io.yaml_loader`, 61



## Symbols

`__call__()` (specviz.core.threads.FileLoadThread method), 53  
`__call__()` (specviz.core.threads.FitModelThread method), 54

## A

`all_fitters` (specviz.interfaces.factories.FitterFactory attribute), 56  
`all_models` (specviz.interfaces.factories.ModelFactory attribute), 57  
`apStar_identify()` (in module specviz.io), 59  
`apStar_loader()` (in module specviz.io), 59  
`apVisit_identify()` (in module specviz.io), 59  
`apVisit_loader()` (in module specviz.io), 59  
`AsciiYamlRegister` (class in specviz.io.yaml\_loader), 61  
`aspcapStar_identify()` (in module specviz.io), 60  
`aspcapStar_loader()` (in module specviz.io), 60

## B

`BlackBody` (class in specviz.analysis), 38

## C

`centroid()` (in module specviz.analysis), 33  
`change_units()` (specviz.core.plots.LinePlot method), 48  
`clear()` (specviz.core.dispatch.EventNode method), 46  
`cos_identify()` (in module specviz.io), 60  
`cos_spectrum_loader()` (in module specviz.io), 60  
`create_fitter()` (specviz.interfaces.factories.FitterFactory class method), 56  
`create_model()` (specviz.interfaces.factories.ModelFactory class method), 57

## D

`data_loader()` (in module specviz.interfaces), 55  
`degree` (specviz.analysis.Spline1D attribute), 39  
`Dispatch` (class in specviz.core.dispatch), 45

## E

`emit()` (specviz.core.dispatch.EventNode method), 46  
`eq_width()` (in module specviz.analysis), 34  
`error_pen` (specviz.core.plots.LinePlot attribute), 48  
`evaluate()` (specviz.analysis.BlackBody method), 38  
`evaluate()` (specviz.analysis.Spline1D method), 39  
`EventNode` (class in specviz.core.dispatch), 46  
`extract()` (in module specviz.analysis), 34  
`extract_range()` (specviz.core.linelist.LineList method), 51  
`extract_rows()` (specviz.core.linelist.LineList method), 51

## F

`Factory` (class in specviz.interfaces.factories), 56  
`FileLoadThread` (class in specviz.core.threads), 52  
`fit_model()` (specviz.core.threads.FitModelThread method), 54  
`FitModelThread` (class in specviz.core.threads), 54  
`fits_identify()` (in module specviz.io), 60  
`FitsYamlRegister` (class in specviz.io.yaml\_loader), 62  
`FitterFactory` (class in specviz.interfaces.factories), 56  
`from_formula()` (specviz.core.data.Spectrum1DRefLayer class method), 42  
`from_formula()` (specviz.core.data.Spectrum1DRefModelLayer class method), 44  
`from_layer()` (specviz.core.plots.LinePlot static method), 49  
`from_parent()` (specviz.core.data.Spectrum1DRefLayer class method), 42  
`from_parent()` (specviz.core.data.Spectrum1DRefModelLayer class method), 44  
`from_self()` (specviz.core.data.Spectrum1DRefLayer method), 42  
`full_mask` (specviz.core.data.Spectrum1DRefLayer attribute), 41  
`fwzi()` (in module specviz.analysis), 35

## I

`identify()` (specviz.io.yaml\_loader.YamlRegister method),

63  
 ingest() (in module specviz.core.linelist), 50  
 initialize() (in module specviz.interfaces.initializers), 58

## L

layer (specviz.core.plots.LinePlot attribute), 48  
 layer\_mask (specviz.core.data.Spectrum1DRefLayer attribute), 41  
 LineList (class in specviz.core.linelist), 51  
 LineListYamlRegister (class in specviz.io.yaml\_loader), 62  
 LinePlot (class in specviz.core.plots), 47

## M

mask\_pen (specviz.core.plots.LinePlot attribute), 48  
 masked\_data (specviz.core.data.Spectrum1DRefLayer attribute), 41  
 masked\_dispersion (specviz.core.data.Spectrum1DRefLayer attribute), 41  
 merge() (specviz.core.linelist.LineList class method), 52  
 model (specviz.core.data.Spectrum1DRefModelLayer attribute), 43  
 ModelFactory (class in specviz.interfaces.factories), 57

## N

norm (specviz.analysis.BlackBody attribute), 38

## P

param\_names (specviz.analysis.BlackBody attribute), 38  
 param\_names (specviz.analysis.Spline1D attribute), 39  
 parent\_mask (specviz.core.data.Spectrum1DRefModelLayer attribute), 43  
 pen (specviz.core.plots.LinePlot attribute), 48  
 plot (specviz.core.plots.LinePlot attribute), 48

## R

raw\_uncertainty (specviz.core.data.Spectrum1DRefLayer attribute), 41  
 read\_file() (specviz.core.threads.FileLoadThread method), 53  
 reader() (specviz.io.yaml\_loader.AsciiYamlRegister method), 62  
 reader() (specviz.io.yaml\_loader.FitsYamlRegister method), 62  
 reader() (specviz.io.yaml\_loader.LineListYamlRegister method), 63  
 reader() (specviz.io.yaml\_loader.YamlRegister method), 63  
 register\_event() (specviz.core.dispatch.Dispatch method), 45  
 register\_listener() (specviz.core.dispatch.Dispatch method), 46  
 resample() (in module specviz.analysis), 35

result (specviz.core.threads.FileLoadThread attribute), 53  
 result (specviz.core.threads.FitModelThread attribute), 54  
 run() (specviz.core.threads.FileLoadThread method), 53  
 run() (specviz.core.threads.FitModelThread method), 55

## S

set\_error\_visibility() (specviz.core.plots.LinePlot method), 49  
 set\_line\_width() (specviz.core.plots.LinePlot method), 49  
 set\_mask\_visibility() (specviz.core.plots.LinePlot method), 49  
 set\_mode() (specviz.core.plots.LinePlot method), 49  
 set\_plot\_visibility() (specviz.core.plots.LinePlot method), 49  
 set\_units() (specviz.core.data.Spectrum1DRefLayer method), 42  
 setup() (specviz.core.dispatch.Dispatch method), 46  
 shape (specviz.core.data.Spectrum1DRefLayer attribute), 41  
 simple\_generic\_loader() (in module specviz.io), 60  
 simple\_generic\_writer() (in module specviz.io), 61  
 smooth (specviz.analysis.Spline1D attribute), 39  
 Spectrum1DRefLayer (class in specviz.core.data), 40  
 Spectrum1DRefModelLayer (class in specviz.core.data), 43  
 specviz.analysis (module), 33  
 specviz.core.data (module), 40  
 specviz.core.dispatch (module), 45  
 specviz.core.events (module), 45  
 specviz.core.linelist (module), 50  
 specviz.core.plots (module), 47  
 specviz.core.threads (module), 52  
 specviz.interfaces (module), 55  
 specviz.interfaces.factories (module), 56  
 specviz.interfaces.initializers (module), 58  
 specviz.io (module), 58  
 specviz.io.yaml\_loader (module), 61  
 Spline1D (class in specviz.analysis), 39  
 stats() (in module specviz.analysis), 37  
 status (specviz.core.threads.FileLoadThread attribute), 53  
 status (specviz.core.threads.FitModelThread attribute), 54  
 stis\_identify() (in module specviz.io), 61  
 stis\_spectrum\_loader() (in module specviz.io), 61

## T

tear\_down() (specviz.core.dispatch.Dispatch method), 46  
 temp (specviz.analysis.BlackBody attribute), 38

## U

unit (specviz.core.data.Spectrum1DRefLayer attribute), 41  
 unmasked\_data (specviz.core.data.Spectrum1DRefLayer attribute), 41

`unmasked_data` (`specviz.core.data.Spectrum1DRefModelLayer` attribute), 43

`unmasked_dispersion` (`specviz.core.data.Spectrum1DRefLayer` attribute), 41

`unmasked_dispersion` (`specviz.core.data.Spectrum1DRefModelLayer` attribute), 44

`unmasked_raw_uncertainty` (`specviz.core.data.Spectrum1DRefLayer` attribute), 42

`unmasked_raw_uncertainty` (`specviz.core.data.Spectrum1DRefModelLayer` attribute), 44

`unregister_listener()` (`specviz.core.dispatch.Dispatch` method), 46

`update()` (`specviz.core.plots.LinePlot` method), 50

## Y

`YamlRegister` (class in `specviz.io.yaml_loader`), 63