wxmplot documentation

Release 0.9.7

Matthew Newville

March 03, 2012

CONTENTS

The wxmplot python package provides easy to use, richly featured plotting widgets for wxPython built on top of the wonderful matplotlib library. While matplotlib provides excellent general purpose plotting functionality, and supports a variety of GUI and non-GUI backends, it does not have a very tight integration with any particular GUI toolkit. Similarly, while wxPython has some plotting functionality, it has nothing as good as matplotlib. The wxmplot package attempts to bridge this gap, providing wx.Panels for basic 2D line plots and image display that are richly featured and provide end-users with interactivity and customization of the graphics without having to know matplotlib. While wxmplot does not expose all of matplotlib's capabilities, but does provide widgets, the plotting and imaging Panels and Frames can be used simply in wxPython applications to handle many use cases.

The wxmplot package is aimed at programmers who want high quality scientific graphics for their applications that can be manipulated by the end-user. If you're a python programmer who is comfortable writing matplotlib / pylab scripts, or plotting interactively from IPython, this package may seem to limiting for your needs.

CHAPTER

ONE

DOWNLOADING AND INSTALLATION

1.1 Prerequisites

The wxmplot package requires Python, wxPython, numpy, and matplotlib. Some of the example applications rely on the Image module as well.

1.2 Downloads

The latest version is available from PyPI or CARS (Univ of Chicago):

Download Option	Python Versions	Location
Source Kit	2.6, 2.7	• wxmplot-0.9.7.tar.gz (CARS) • wxmplot-0.9.7.tar.gz (PyPI) • wxmplot-0.9.7.zip (CARS) • wxmplot-0.9.7.zip (PyPI)
Windows Installers	2.6 2.7	• wxmplot-0.9.7win32-py2.6.exe • wxmplot-0.9.7win32-py2.7.exe
Development Version	all	use wxmplot github repository

if you have Python Setup Tools installed, you can download and install the package simply with:

```
easy_install -U wxmplot
```

1.3 Development Version

To get the latest development version, use:

```
git clone http://github.com/newville/wxmplot.git
```

1.4 Installation

This is a pure python module, so installation on all platforms can use the source kit:

```
tar xvzf wxmplot-0.9.7.tar.gz  or unzip wxmplot-0.9.7.zip
cd wxmplot-0.9.7/
python setup.py install
```

1.5 License

The wxmplot code is distribution under the following license:

CHAPTER

TWO

PLOTPANEL: A WX.PANEL FOR BASIC 2D LINE PLOTS

The `PlotPanel` class supports standard 2 dimensional plots, including line plots and scatter plots, with a simple-to-use programming interface. This is derived from a `wx.Panel` and so can be included in a wx GUI anywhere a `wx.Panel` can be. A `PlotPanel` provides the following capabilities for the end-user:

1. display x, y coordinates as the mouse move.
2. display x, y coordinates of last left-click.
3. zoom in on a particular region of the plot with left-drag in a lineplot, or draw an 'lasso' around selected points in a scatterplot.
4. customize titles, labels, legend, colors, linestyles, markers, and whether a grid and a legend is shown. A separate window is used to give control of these settings.
5. save high-quality plot images (as PNGs), or copy to system clipboard, or print.

These both classes have the basic plotting methods of `plot()` to make a new plot with a single trace, and `oplot()` to overplot another trace on top of an existing plot. These each take 2 equal-length numpy arrays (abscissa, ordinate) for each trace. The `PlotPanel` and `PlotFrame` have many additional methods to interact with the plots.

class **PlotPanel** (*parent*[, *size=(6.0, 3.7)*[, *dpi=96*[, *messenger=None*[, *show_config_popup=True*[, ***kws*]]]]])

Create a Plot Panel, a `wx.Panel`

Parameters

- **parent** – wx parent object.
- **size** – figure size in inches.
- **dpi** – dots per inch for figure.
- **messenger** (callable or `None`) – function for accepting output messages.
- **show_config_popup** (`True`/`False`) – whether to enable a popup-menu on right-click.

The *size*, and *dpi* arguments are sent to matplotlib's `Figure`. The *messenger* should should be a function that accepts text messages from the panel for informational display. The default value is to use `sys.stdout.write()`.

The *show_config_popup* arguments controls whether to bind right-click to showing a poup menu with options to zoom in or out, configure the plot, or save the image to a file.

Extra keyword parameters are sent to the wx.Panel.

2.1 PlotPanel methods

plot (*x*, *y*, ***kws*)

Draw a plot of the numpy arrays *x* and *y*, erasing any existing plot. The displayed curve for these data is called a *trace*. The plot() method has many optional parameters, all using keyword/value argument. Since most of these are shared with the oplot() method, the full set of parameters is given in *Table of Plot Arguments*

oplot (*x*, *y*, ***kws*)

Draw a plot of the numpy arrays *x* and *y*, overwriting any existing plot.

The oplot() method has many optional parameters, as listed in *Table of Plot Arguments*

Table of Plot Arguments These arguments apply for the plot(), oplot(), and scatterplot() methods. Except where noted, the arguments are available for plot() and oplot(). In addition, the scatterplot() method uses many of the same arguments for the same meaning, as indicated by the right-most column.

argument	type	default	meaning	scatterplot?
title	string	None	Plot title	yes
ylabel	string	None	abscissa label	yes
y2label	string	None	right-hand abscissa label	yes
label	string	None	trace label (defaults to 'trace N')	yes
side	left/right	left	side for y-axis and label	yes
grid	None/bool	None	to show grid lines	yes
color	string	blue	color to use for trace	yes
use_dates	bool	False	to show dates in xlabel (plot() only)	no
linewidth	int	2	linewidth for trace	no
style	string	solid	line-style for trace (solid, dashed, ...)	no
drawstyle	string	line	style connecting points of trace	no
marker	string	None	symbol to show for each point (+, o, ...)	no
markersize	int	8	size of marker shown for each point	no
dy	array	None	uncertainties for y values; error bars	no
ylog_scale	bool	False	draw y axis with log(base 10) scale	no
xmin	float	None	minimum displayed x value	yes
xmax	float	None	maximum displayed x value	yes
ymin	float	None	minimum displayed y value	yes
ymax	float	None	maximum displayed y value	yes
autoscale	bool	True	whether to automatically set plot limits	no
draw_legend	None/bool	None	whether to display legend (None: leave as is)	no
refresh	bool	True	whether to refresh display	no
	arguments that apply only for scatterplot()			
size	int	10	size of marker	yes
edgecolor	string	black	edge color of marker	yes
selectcolor	string	red	color for selected points	yes
callback	function	None	user-supplied callback to run on selection	yes

As a general note, the configuration for the plot (title, labels, grid displays) and for each trace (color, linewidth, ...) are preserved for a PlotPanel. A few specific notes:

1. The title, label, and grid arguments to plot() default to None, which means to use the previously used value.

2. The *use_dates* option is not very rich, and simply turns x-values that are Unix timestamps into x labels showing the dates.

3. While the default is to auto-scale the plot from the data ranges, specifying any of the limits will override the corresponding limit(s).

4. The *color* argument can be any color name ("blue", "red", "black", etc), standard X11 color names ("cadetblue3", "darkgreen", etc), or an RGB hex color string of the form "#RRGGBB".

5. Valid *style* arguments are 'solid', 'dashed', 'dotted', or 'dash-dot', with 'solid' as the default.

6. Valid *marker* arguments are '+', 'o', 'x', '^', 'v', '>', '<', 'l', '_', 'square', 'diamond', 'thin diamond', 'hexagon', 'pentagon', 'tripod 1', or 'tripod 2'.

7. Valid *drawstyles* are None (which connects points with a straight line), 'steps-pre', 'steps-mid', or 'steps-post', which give a step between the points, either just after a point ('steps-pre'), midway between them ('steps-mid') or just before each point ('steps-post'). Note that if displaying discrete values as a function of time, left-to-right, and want to show a transition to a new value as a sudden step, you want 'steps-post'.

All of these values, and a few more settings controlling whether and how to display a plot legend can be configured interactively (see Plot Configuration).

update_line(*trace*, *x*, *y*[, *side='left'*])
update an existing trace.

Parameters

- **trace** – integer index for the trace (0 is the first trace)
- **x** – array of x values
- **y** – array of y values
- **side** – which y axis to use ('left' or 'right').

This function is particularly useful for data that is changing and you wish to update traces from a previous `plot()` or `oplot()` with the new (x, y) data without completely redrawing the entire plot. Using this method is substantially faster than replotting, and should be used for dynamic plots such as a StripChart.

scatterplot(*x*, *y*, ***kws*)
draws a 2d scatterplot. This is a collection of points that are not meant to imply a specific order that can be connected by a continuous line. A full list of arguments are listed in *Table of Plot Arguments*.

clear()
Clear the plot.

set_xylims(*limits*[, *axes=None*[, *side=None*[, *autoscale=True*]]])
Set the x and y limits for a plot based on a 2x2 list.

Parameters

- **limits** (*a 4-element list: [xmin, xmax, ymin, ymax]*) – x and y limits
- **axes** – instance of matplotlib axes to use (i.e, for right or left side y axes)
- **side** – set to 'right' to get right-hand axes.
- **autoscale** – whether to automatically scale to data range.

That is, if *autoscale=False* is passed in, then the limits are used.

get_xylims()
return current x, y limits.

unzoom()
unzoom the plot. The x, y limits for interactive zooms are stored, and this function unzooms one level.

unzoom_all()
unzoom the plot to the full data range.

set_title (*title*)
set the plot title.

set_xlabel (*label*)
set the label for the ordinate axis.

set_ylabel (*label*)
set the label for the left-hand abscissa axis.

set_y2label (*label*)
set the label for the right-hand abscissa axis.

set_bgcol (*color*)
set the background color for the PlotPanel.

write_message (*message*)
write a message to the messenger. For a PlotPanel embedded in a PlotFrame, this will go the the StatusBar.

save_figure ()
show a FileDialog to save a PNG image of the current plot.

configure ()
show plot configuration window for customizing plot.

reset_config ()
reset the configuration to default settings.

2.2 `PlotFrame`: a wx.Frame showing a `PlotPanel`

A `PlotFrame` is a wx.Frame – a separate plot window – that contains a `PlotPanel` and is decorated with a status bar and menubar with menu items for saving, printing and configuring plots. It inherits many of the methods of a `PlotPanel`.

class PlotFrame (*parent*[, *size=(700, 450)*[, *title=None*[, ***kws*]]])
create a plot frame. This frame will have a `panel` member holding the underlying `PlotPanel`, and have menus and statusbar for plot interaction.

plot (*x*, *y*, ***kws*)
Passed to panel.plot

oplot (*x*, *y*, ***kws*)
Passed to panel.oplot

scatterplot (*x*, *y*, ***kws*)
Passed to panel.scatterplot

clear (*x*, *y*, ***kws*)
Passed to panel.clear

update_line (*x*, *y*, ***kws*)
Passed to panel.update_line

reset_config (*x*, *y*, ***kws*)
Passed to panel.reset_config

2.3 PlotApp: a wx.App showing a PlotFrame

A PlotApp is a wx.App – an application – that consists of a PlotFrame. This show a frame that is decorated with a status bar and menubar with menu items for saving, printing and configuring plots.

class PlotAppp

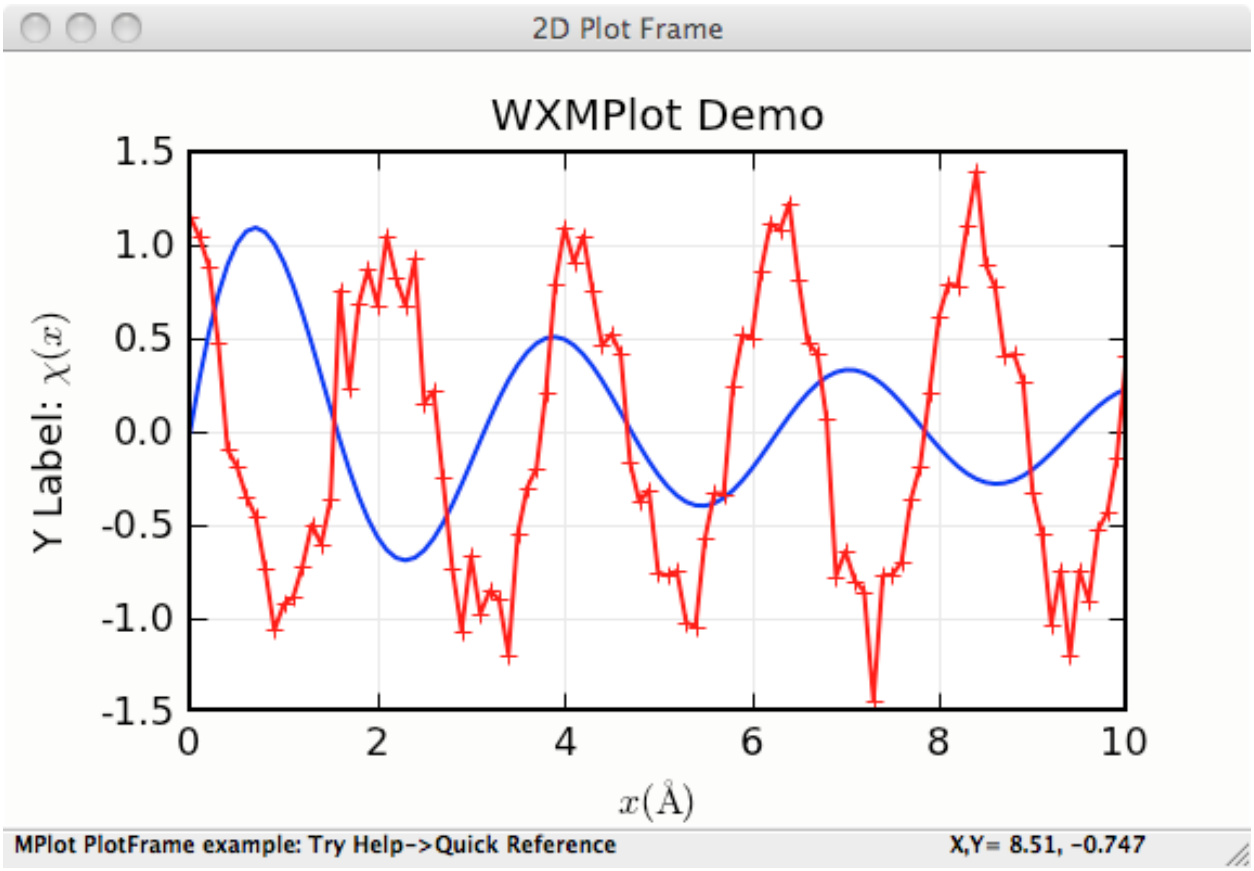
create a plot application. This has methods plot(), oplot(), and write_message(), which are sent to the underlying PlotPanel.

This allows very simple scripts which give plot interactivity and customization:

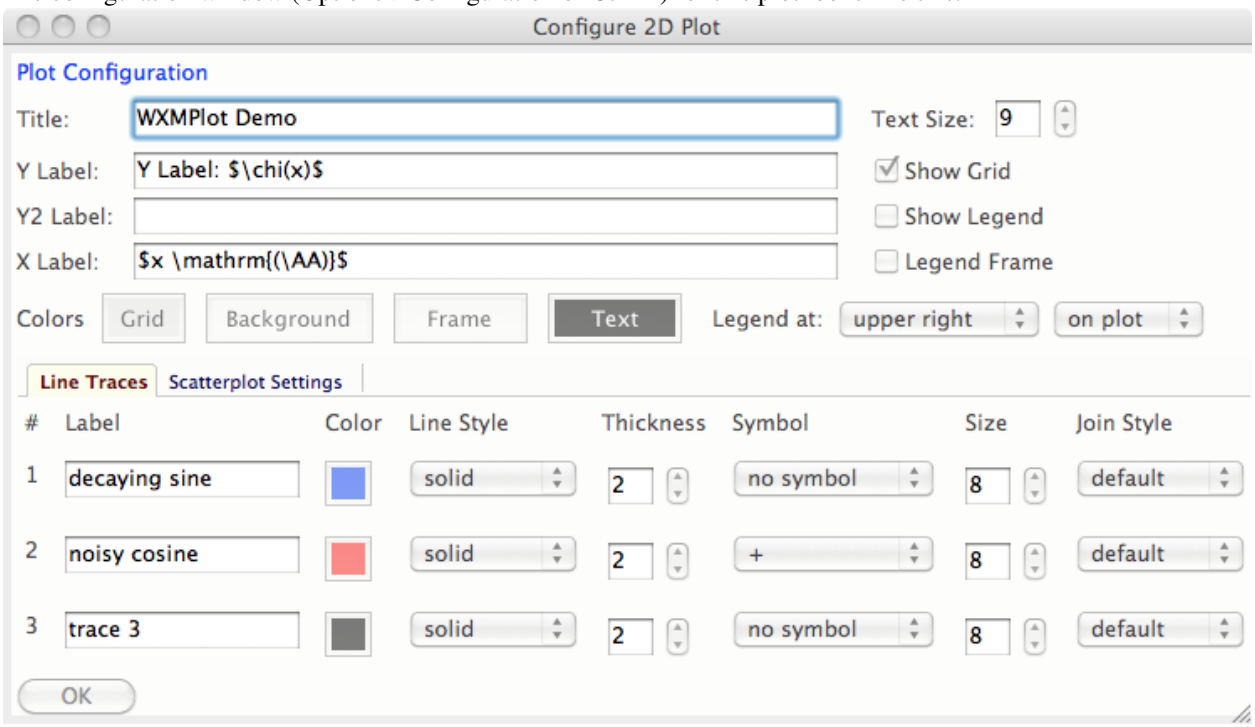
```
from wxmplot import PlotApp
from numpy import arange, sin, cos, exp, pi

xx  = arange(0.0,12.0,0.1)
y1  = 1*sin(2*pi*xx/3.0)
y2  = 4*cos(2*pi*(xx-1)/5.0)/(6+xx)
y3  = -pi + 2*(xx/10. + exp(-(xx-3)/5.0))

p = PlotApp()
p.plot(xx, y1, color='blue',  style='dashed',
       title='Example PlotApp',  label='a',
       ylabel=r'$k^2\chi(k) $',
       xlabel=r'$  k \ (\AA^{-1}) $' )

p.oplot(xx, y2,  marker='+', linewidth=0, label =r'$ x_1 $')
p.oplot(xx, y3,  style='solid',          label ='x_2')
p.write_message(Try Help->Quick Reference')
p.run()
```

2.4 Examples and Screenshots

A basic plot from a PlotFrame looks like this:

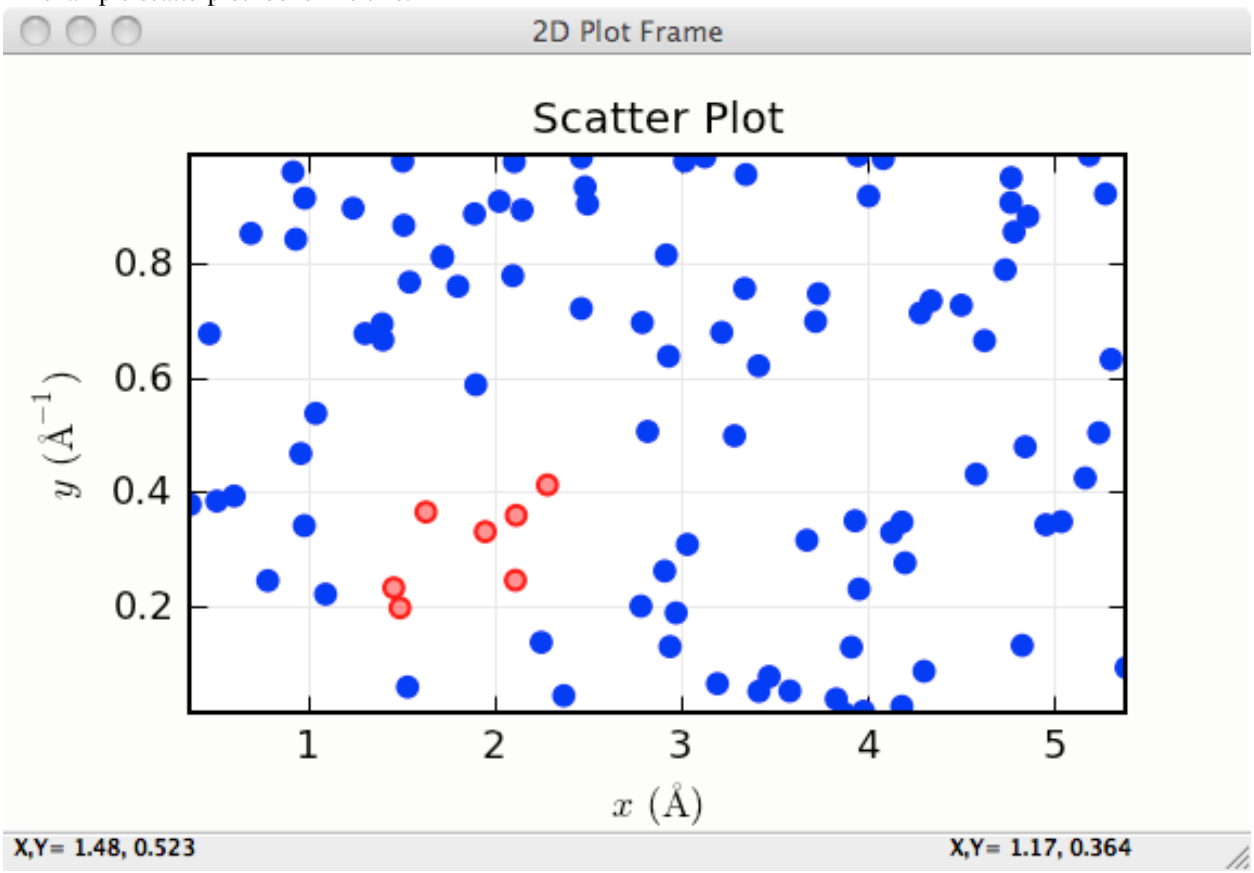


The configuration window (Options->Configuration or Ctrl-K) for this plot looks like this:



where all the options there will dynamically change the plot in the PlotPanel.

An example scatterplot looks like this:



Many more examples are given in the *examples* directory in the source distribution kit. The *demo.py* script there will show several 2D Plot panel examples, including a plot which uses a timer to simulate a dynamic plot, updating the plot as fast as it can - typically 10 to 30 times per second, depending on your machine. The *stripchart.py* example script also shows a dynamic, time-based plot.

CHAPTER

THREE

IMAGEPANEL: A WX.PANEL FOR IMAGE DISPLAY

The `ImagePanel` class supports image display (ie, gray-scale and false-color intensity maps for 2-D arrays. As with `PlotPanel`, this is derived from a `wx.Panel` and so can be included in a wx GUI anywhere a `wx.Panel` can be. While the image can be customized programmatically, the only interactivity built in to the `ImagePanel` is the ability to zoom in and out.

In contrast, an `ImageFrame` provides many more ways to manipulate an image, and will be discussed below.

class ImagePanel (*parent*[, *size=(4.5, 4.0)*[, *dpi=96*[, *messenger=None*[, *data_callback=None*[, ***kws*]]]]])

Create an Image Panel, a `wx.Panel`

Parameters

- **parent** – wx parent object.
- **size** – figure size in inches.
- **dpi** – dots per inch for figure.
- **messenger** (callable or `None`) – function for accepting output messages.
- **data_callback** (callable or `None`) – function to call with new data, on `display()`

The *size*, and *dpi* arguments are sent to matplotlib's `Figure`. The *messenger* should should be a function that accepts text messages from the panel for informational display. The default value is to use `sys.stdout.write()`.

The *data_callback* is useful if some parent frame wants to know if the data has been changed with `display()`. `ImageFrame` uses this to display the intensity max/min values.

Extra keyword parameters are sent to the wx.Panel.

The configuration settings for an image (its colormap, smoothing, orientation, and so on) are controlled through configuration attributes.

3.1 ImagePanel methods

display (*data*[, *x=None*[, *y=None*[, ***kws*]]])

display a new image from the 2-D numpy array *data*. If provided, the *x* and *y* values will be used for display purposes, as to give scales to the pixels of the data.

Additional keyword arguments will be sent to a *data_callback* function, if that has been defined.

3.2 `ImageFrame`: A wx.Frame for Image Display

In addition to providing a top-level window frame holding an `ImagePanel`, an `ImageFrame` provides the end-user with many ways to manipulate the image:

1. display x, y, intensity coordinates (left-click)
2. zoom in on a particular region of the plot (left-drag).
3. change color maps.
4. flip and rotate image.
5. select optional smoothing interpolation.
6. modify intensity scales.
7. save high-quality plot images (as PNGs), copy to system clipboard, or print.

These options are all available programmatically as well, by setting the configuration attributes and redrawing the image.

class `ImageFrame` (*parent*[, *size=(550, 450)*[, ***kws*]])
Create an Image Frame, a `wx.Frame`.

3.3 Image configuration with `ImageConfig`

To change any of the attributes of the image on an `ImagePanel`, you can set the corresponding attribute of the panel's `conf`. That is, if you create an `ImagePanel`, you can set the colormap with:

```
import matplotlib.cm as cmap
im_panel = ImagePanel(parent)
im_panel.display(data_array)

# now change colormap:
im_panel.conf.cmap = cmap.cool
im_panel.redraw()

# now rotate the image by 90 degrees (clockwise):
im_panel.conf.rot = True
im_panel.redraw()

# now flip the image (top/bottom), apply log-scaling,
# and apply gaussian interpolation
im_panel.conf.flip_ud = True
im_panel.conf.log_scale = True
im_panel.conf.interp = 'gaussian'
im_panel.redraw()
```

For a `ImageFrame`, you can access this attribute as *frame.panel.conf.cmap*.

The list of configuration attributes and their meaning are given in the *Table of Image Configuration attributes* Table of Image Configuration attributes: All of these are members of the *panel.conf* object, as shown in the example above.

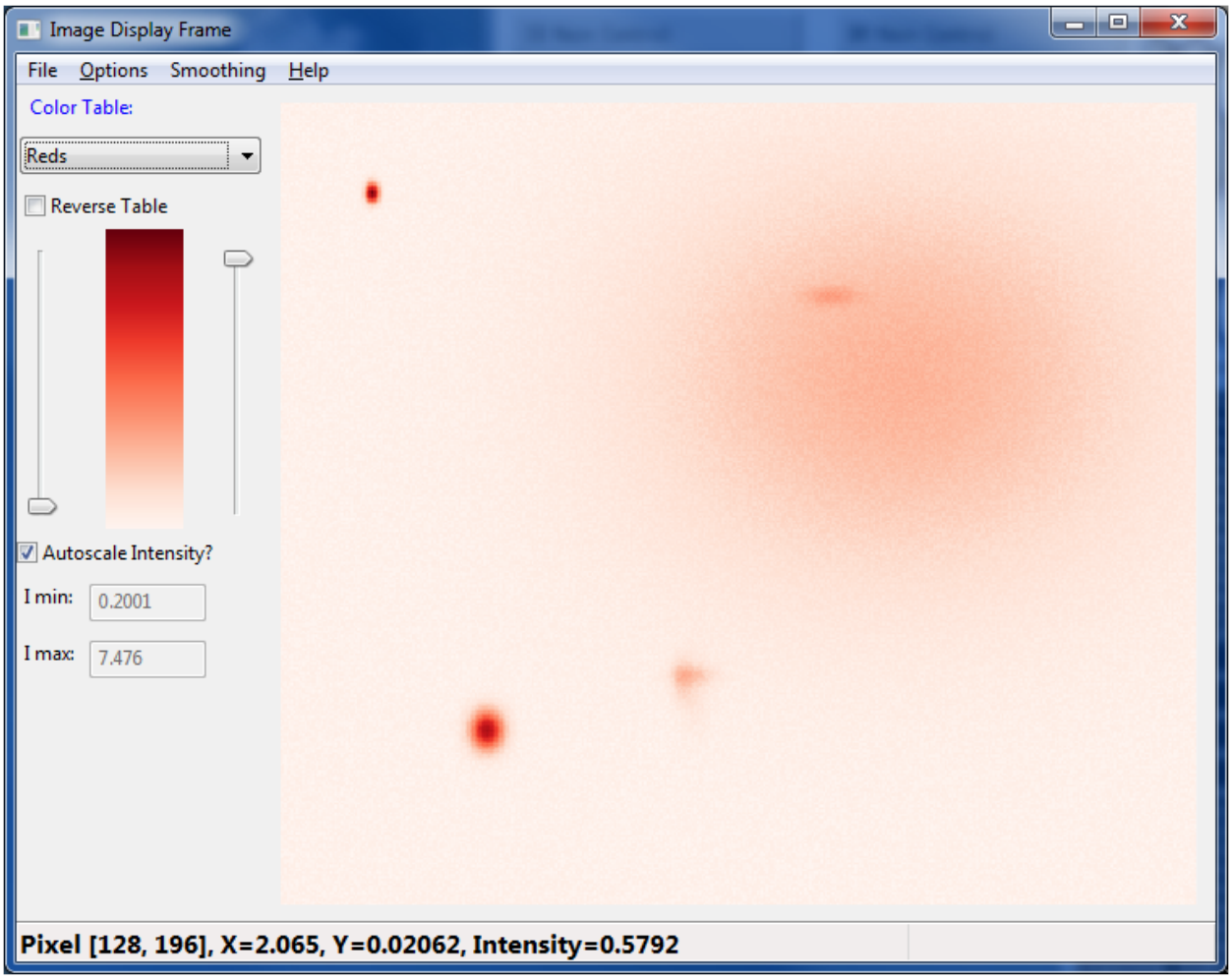
attribute	type	default	meaning
rot	bool	False	rotate image 90 degrees clockwise
flip_ud	bool	False	flip image top/bottom
flip_lr	bool	False	flip image left/right
log_scale	bool	False	display log(image)
auto_intensity	bool	True	auto-scale the intensity
cmap	colormap	gray	colormap for intensity scale
cmap_reverse	bool	False	reverse colormap
interp	string	nearest	interpolation, smoothing algorithm
xylims	list	None	xmin, xmax, ymin, ymax for display
cmap_lo	int	0	low intensity percent for colormap mapping
cmap_hi	int	100	high intensity percent for colormap mapping
int_lo	float	None	low intensity when autoscaling is off
int_hi	float	None	high intensity when autoscaling is off

Some notes:

1. *cmap* is an instance of a matplotlib colormap.
2. *cmap_lo* and *cmap_hi* set the low and high values for the sliders that compress the colormap, and are on a scale from 0 to 100.
3. In contrast, *int_lo* and *int_hi* set the map intensity values that are used when *auto_intensity* is `False`. These can be used to put two different maps on the same intensity intensity scale.

3.4 Examples and Screenshots

A basic plot from a `ImageFrame` looks like this:



This screenshot shows a long list of choices for color table, a checkbox to reverse the color table, sliders to adjust the upper and lower level, a checkbox to auto-scale the intensity, or entries to set the intensity values for minimum and maximum intensity. Clicking on the image will show its coordinates and intensity value. Click-and-Drag will select a rectangular box to zoom in on a particular feature of the image.

The File menu includes options to save an PNG file of the image (Ctrl-S), copy the image to the system clipboard (Ctrl-C), print (Ctrl-P) or print-preview the image, or quit the application. The Options menu includes Zoom Out (Ctrl-Z), applying a log-scale to the intensity (Ctrl-L), rotating the image clockwise (Ctrl-R), flipping the image top/bottom (Ctrl-T) or right/left (Ctrl-F), or saving an image of the colormap. The Smoothing menu allows you choose from one of several interpolation algorithms.

INDEX