Creating an About Box

When I create an application, I usually want to include an “About” box to let the user know more about the application, myself and to give shout outs to anyone who may have helped in the creation of my program. One cool feature wxPython provides is a custom AboutBox widget. I think it looks a little odd, so I created my own About box using the HtmlWindow widget. However, I'll show how to do it both ways in this recipe.

First, we'll create a simple application that will allow you to see how to open the dialog either via a button or a menu item. It will have three buttons, one to open the wx.AboutBox, one to open my Html version of the About box and a close button.

Using wx.AboutBox

Creating the AboutBox is pretty straightforward. Let's take a look at a simple snippet of code:

```
def onAboutDlg(self, event):
    info = wx.AboutDialogInfo()
    info.Name = "My About Box"
    info.Version = "0.0.1 Beta"
    info.Copyright = "(C) 2008 Python Geeks Everywhere"
    info.Description = wordwrap(
        "This is an example application that shows how to create "
        "different kinds of About Boxes using wxPython!",
        350, wx.ClientDC(self.panel))
    info.WebSite = ("http://www.pythonlibrary.org", "My Home Page")
    info.Developers = ["Mike Driscoll"]
    info.License = wordwrap("Completely and totally open source!", 500,
                            wx.ClientDC(self.panel))
    # Show the wx.AboutBox
    wx.AboutBox(info)
```

To begin, you instantiate an instance of wx.AboutDlgInfo. This gives you a way to set the various pieces of information you want to display in your AboutBox, such as application name, version, copyright, etc. When you have that all filled in, you create the wx.AboutBox and pass it that information. Notice that this does not require you to explicitly “show” it; that's done automatically.

When done, you should see something like the image below:

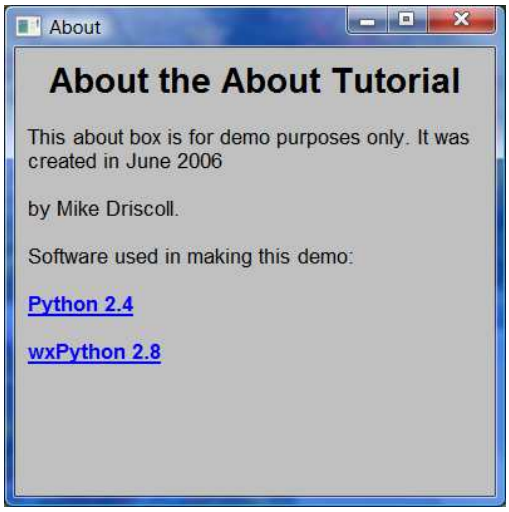
Now we can move on and learn how to create an about dialog using HTML.

Using HtmlWindow for an About Box

Creating the HTML version is a little bit more complex. I prefer splitting the code up into two classes. The two top level widgets I recommend using to base your About Box on would be wx.Frame or wx.Dialog. In this example I'll use a wx.Frame widget. The 2nd class is only to catch mouse clicks on URLs, if you have some. Let's take a look at the code:

```
class AboutDlg(wx.Frame):

    def __init__(self, parent):

        wx.Frame.__init__(self, parent, wx.ID_ANY, title="About", size=(400,400))

        html = wxHTML(self)

        html.SetPage(
            ''

            "<h2>About the About Tutorial</h2>"

            "<p>This about box is for demo purposes only. It was created in June 2006"

            "by Mike Driscoll.</p>"

            "<p><b>Software used in making this demo:</h3></p>"

            '<p><b><a href="http://www.python.org">Python 2.4</a></b></p>'

            '<p><b><a href="http://www.wxpython.org">wxPython 2.8</a></b></p>'
            )

class wxHTML(wx.html.HtmlWindow):

     def OnLinkClicked(self, link):
         webbrowser.open(link.GetHref())
```

The reason I like this so much is that it allows me to specify font sizes, use html tables, insert photos and more very easily, plus it's completely cross-platform. One definite disadvantage is that this widget doesn't allow advanced html, such as css or javascript. Anyway, when you get done, it should turn into something that looks similar to the screenshot below:



Here's the full source for my demo program I used to activate my two About Boxes:

```
import wx
import wx.html
from wx.lib.wordwrap import wordwrap

class MyForm(wx.Frame):

    def __init__(self):
        wx.Frame.__init__(self, None, wx.ID_ANY, title='My Form')

        # Add a panel so it looks correct on all platforms
        self.panel = wx.Panel(self, wx.ID_ANY)

        # Create buttons
        aboutBtn = wx.Button(self.panel, wx.ID_ANY, "Open wx.AboutBox")
        self.Bind(wx.EVT_BUTTON, self.onAboutDlg, aboutBtn)
        aboutHtmlBtn = wx.Button(self.panel, wx.ID_ANY, "Open HtmlAboutBox")
        self.Bind(wx.EVT_BUTTON, self.onAboutHtmlDlg, aboutHtmlBtn)

        closeBtn = wx.Button(self.panel, wx.ID_ANY, "Close")
        self.Bind(wx.EVT_BUTTON, self.onClose, closeBtn)

        # Create Sizers
        topSizer = wx.BoxSizer(wx.VERTICAL)

        # Add widgets to sizers
        topSizer.Add(aboutBtn, 0, wx.ALL|wx.CENTER, 5)
        topSizer.Add(aboutHtmlBtn, 0, wx.ALL|wx.CENTER, 5)
        topSizer.Add(closeBtn, 0, wx.ALL|wx.CENTER, 5)

        # Create the menu
        self.createMenu()
        self.statusBar = self.CreateStatusBar()

        self.panel.SetSizer(topSizer)
        self.SetSizeHints(250,300,500,400)
        self.Fit()
        self.Refresh()

    def createMenu(self):
        """ Create the application's menu """
        menubar = wx.MenuBar()

        # Create the file menu
        fileMenu = wx.Menu()

        # Append the close item
        # Append takes an id, the text label, and a string
        # to display in the statusbar when the item is selected
        close_menu_item = fileMenu.Append(wx.NewId(),
                                          "&amp;Close",
                                          "Closes the application")
        # Bind an event to the menu item
        self.Bind(wx.EVT_MENU, self.onClose, close_menu_item)
        # Add the fileMenu to the menu bar
```

```
        menubar.Append(fileMenu, "&amp;amp;File")

        # Create the help menu
        helpMenu = wx.Menu()
        about_menu_item = helpMenu.Append(wx.NewId(),
                                          "&amp;amp;About",
                                          "Opens the About Box")
        self.Bind(wx.EVT_MENU, self.onAboutDlg, about_menu_item)
        menubar.Append(helpMenu, "&amp;amp;Help")

        # Add the menu bar to the frame
        self.SetMenuBar(menubar)

    def onAboutHtmlDlg(self, event):
        aboutDlg = AboutDlg(None)
        aboutDlg.Show()

    def onAboutDlg(self, event):
        info = wx.AboutDialogInfo()
        info.Name = "My About Box"
        info.Version = "0.0.1 Beta"
        info.Copyright = "(C) 2008 Python Geeks Everywhere"
        info.Description = wordwrap(
            "This is an example application that shows how to create "
            "different kinds of About Boxes using wxPython!",
            350, wx.ClientDC(self.panel))
        info.WebSite = ("http://www.pythonlibrary.org", "My Home Page")
        info.Developers = ["Mike Driscoll"]
        info.License = wordwrap("Completely and totally open source!", 500,
                                wx.ClientDC(self.panel))
        # Show the wx.AboutBox
        wx.AboutBox(info)

    def onClose(self, event):
        self.Close()


class AboutDlg(wx.Frame):

    def __init__(self, parent):

        wx.Frame.__init__(self, parent, wx.ID_ANY, title="About", size=(400,400))

        html = wxHTML(self)

        html.SetPage(
            ''

            "<h2>About the About Tutorial</h2>"

            "<p>This about box is for demo purposes only. It was created in June 2006"

            "by Mike Driscoll.</p>"

            "<p><b>Software used in making this demo:</h3></p>"
```

```
            '<p><b><a href="http://www.python.org">Python 2.4</a></b></p>'

            '<p><b><a href="http://www.wxpython.org">wxPython 2.8</a></b></p>'
            )

class wxHTML(wx.html.HtmlWindow):
     def OnLinkClicked(self, link):
         webbrowser.open(link.GetHref())


# Run the program
if __name__ == '__main__':
    app = wx.PySimpleApp()
    frame = MyForm().Show()
    app.MainLoop()
```

The main things to take note of here is how I handle the setup of the menubar and the related menubar events. If you're no familiar with hooking those up, then you'll probably find this helpful. I'll be going over this process in more detail in a later post, but suffice it to say that you need to create a wx.MenuBar object and some wx.Menu objects. The wx.Menu objects are used for the headings of the menu (i.e. "File", "About". etc).

The wx.Menu objects are then appended to the menubar. Finally you do a self.SetMenuBar() command to attach the menubar to your application's wx.Frame.

Wrapping Up

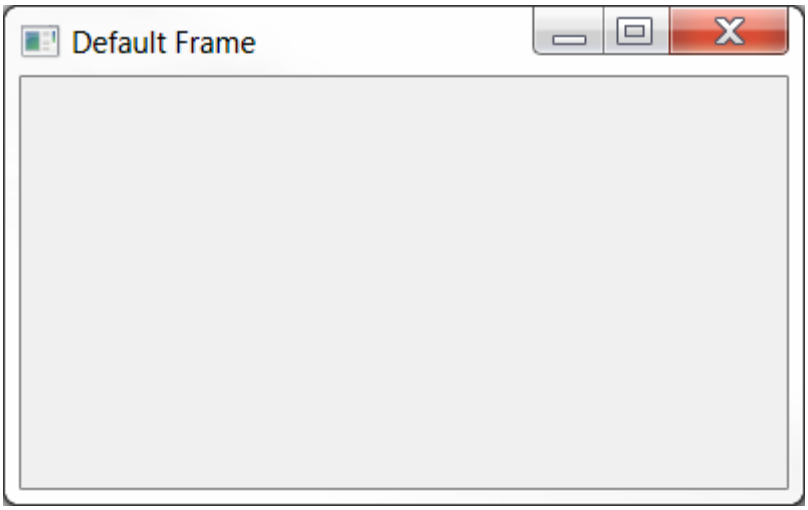
Now you have the knowledge to create your own About Box. These widgets are quite useful for communicating to the end user information about your program such as what version they are using. Some developers use the About Box for displaying information about their company or the product. They also use them to allow the user to manually check for updates. You can use yours as you see fit as they are completely customizable.

Using wx.Frame Styles

The wxPython Frame widget is used in almost all wxPython applications. It has the minimize, maximize and close buttons on it as well as the caption along the top that identifies the application. The wx.Frame allows you to modify its styles in such a way that you can remove or disable various buttons and features. In this article, we will look at some of the ways that you can change the behavior of the wx.Frame widget. Specifically, we will cover the following:

- Different ways to create a default frame
- How to create a frame without a caption (i.e. no title bar)
- How to create a frame with a disabled close button
- How to create a frame without a maximize or minimize button
- How to create a frame that cannot be resized
- How to create a frame without the system menu
- How to make your frame stay on top of other windows

Getting Started



It's always a good idea to look at how the default style works and then modify that to see what happens. So let's start with the frame's default style: wx.DEFAULT_FRAME_STYLE. You can create a frame that uses **wx.DEFAULT_FRAME_STYLE** (or its equivalent) in 3 different ways. The first an easiest is to just do something like this:

```
import wx


class DefaultFrame(wx.Frame):
    """
    The default frame
    """

    def __init__(self):
        """Constructor"""
        wx.Frame.__init__(self, None, title="Default Frame")
        panel = wx.Panel(self)
        self.Show()
```

```
if __name__ == "__main__":
    app = wx.App(False)
    frame = DefaultFrame()
    app.MainLoop()
```

This will create a normal frame with all the normal functionality any user would expect. Now let's change it slightly by passing it the wx.DEFAULT_FRAME_STYLE.

```
import wx


class DefaultFrame(wx.Frame):
    """
    The default frame
    """

    def __init__(self):
        """Constructor"""
        wx.Frame.__init__(self, None, title="Default Frame",
                          style=wx.DEFAULT_FRAME_STYLE)
        panel = wx.Panel(self)
        self.Show()


if __name__ == "__main__":
    app = wx.App(False)
    frame = DefaultFrame()
    app.MainLoop()
```

This code does EXACTLY the same thing as the previous code. Now if you do a little research, you'll find out that wx.DEFAULT_FRAME_STYLE is the equivalent of passing the following:

```
wx.MINIMIZE_BOX | wx.MAXIMIZE_BOX | wx.RESIZE_BORDER | wx.SYSTEM_MENU
| wx.CAPTION | wx.CLOSE_BOX | wx.CLIP_CHILDREN
```

So let's modify out code one more time to show how that would work.

```
import wx


class DefaultFrame(wx.Frame):
    """
    The default frame
    """

    def __init__(self):
        """Constructor"""
        default = (wx.MINIMIZE_BOX | wx.MAXIMIZE_BOX | wx.RESIZE_BORDER
                   | wx.SYSTEM_MENU | wx.CAPTION | wx.CLOSE_BOX
                   | wx.CLIP_CHILDREN)
        wx.Frame.__init__(self, None, title="Default Frame", style=default)
        panel = wx.Panel(self)
```

```
        self.Show()


if __name__ == "__main__":
    app = wx.App(False)
    frame = DefaultFrame()
    app.MainLoop()
```

That was easy. Now we're ready to start experimenting!

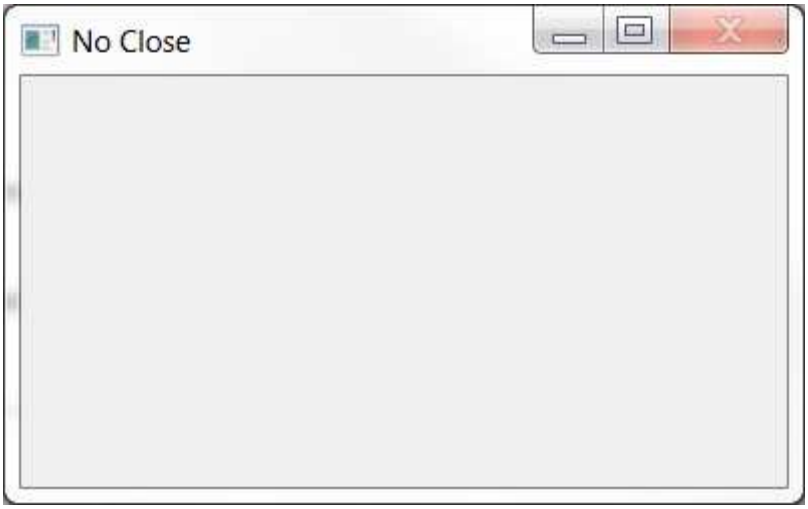
Create a Frame Without a Caption

Let's create a frame that doesn't have a caption. The caption is what holds the buttons along the top of the frame along with the title of the application.

```
import wx


class NoCaptionFrame(wx.Frame):
    """"""

    def __init__(self):
        """Constructor"""
        no_caption = (wx.MINIMIZE_BOX | wx.MAXIMIZE_BOX | wx.RESIZE_BORDER
                      | wx.SYSTEM_MENU | wx.CLOSE_BOX | wx.CLIP_CHILDREN)
        wx.Frame.__init__(self, None, title="No Caption", style=no_caption)
        panel = wx.Panel(self)
        self.Show()


if __name__ == "__main__":
    app = wx.App(False)
    frame = NoCaptionFrame()
    app.MainLoop()
```

When this code is run, the panel is squashed up in the upper left hand corner of the frame. You can resize the frame and the panel will “snap” into place, but it's kind of weird looking. You might also note that you cannot close this application since there is no close button on it. You will need to kill your Python process to close this application.

Create a Frame With a Disabled Close Button

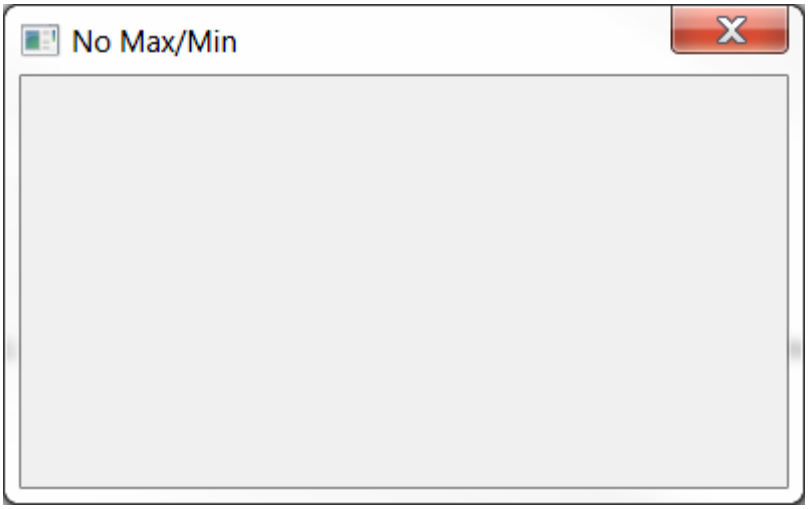


Some programmers think they need a frame where there's no close button. Well you can't really remove the close button (on Windows) and keep the other buttons at the same time, but you can disable the close button. On Linux, the close button actually does get removed. Here's the code:

```
import wx


class NoCloseFrame(wx.Frame):
    """
    This frame has no close box and the close menu is disabled
    """

    def __init__(self):
        """Constructor"""
        no_close = (wx.MINIMIZE_BOX | wx.MAXIMIZE_BOX | wx.RESIZE_BORDER
                    | wx.SYSTEM_MENU | wx.CAPTION | wx.CLIP_CHILDREN)
        wx.Frame.__init__(self, None, title="No Close", style=no_close)
        panel = wx.Panel(self)
        self.Show()


if __name__ == "__main__":
    app = wx.App(False)
    frame = NoCloseFrame()
    app.MainLoop()
```

Of course, on Windows you cannot close this application either, so this is a rather annoying piece application. You'll probably want to add a wx.Button that can close it instead. On Linux, you can close it by double-clicking the top left corner.

Create a Frame Without Maximize/Minimize

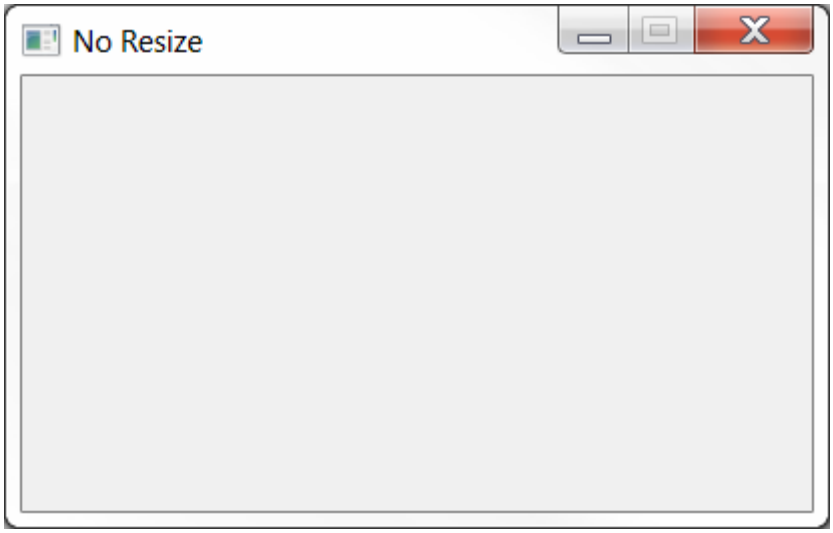


Sometimes you'll want to create an application that you cannot minimize or maximize. If you're going to go that far, let's make an application that also doesn't show up in the taskbar!

```
import wx


class NoMaxMinFrame(wx.Frame):
    """
    This frame does not have maximize or minimize buttons
    """

    def __init__(self):
        """Constructor"""
        no_caption = (wx.RESIZE_BORDER | wx.SYSTEM_MENU | wx.CAPTION
                      | wx.CLOSE_BOX | wx.CLIP_CHILDREN
                      | wx.FRAME_NO_TASKBAR)
        wx.Frame.__init__(self, None, title="No Max/Min", style=no_caption)
        panel = wx.Panel(self)
        self.Show()


if __name__ == "__main__":
    app = wx.App(False)
    frame = NoMaxMinFrame()
    app.MainLoop()
```

As you can see, we just removed the wx.MINIMIZE_BOX and wx.MAXIMIZE_BOX style flags and added the wx.FRAME_NO_TASKBAR style flag. This works just fine on Windows 7, but on Linux I noticed that the Maximize button wasn't removed.

Create a Un-Resizable Frame

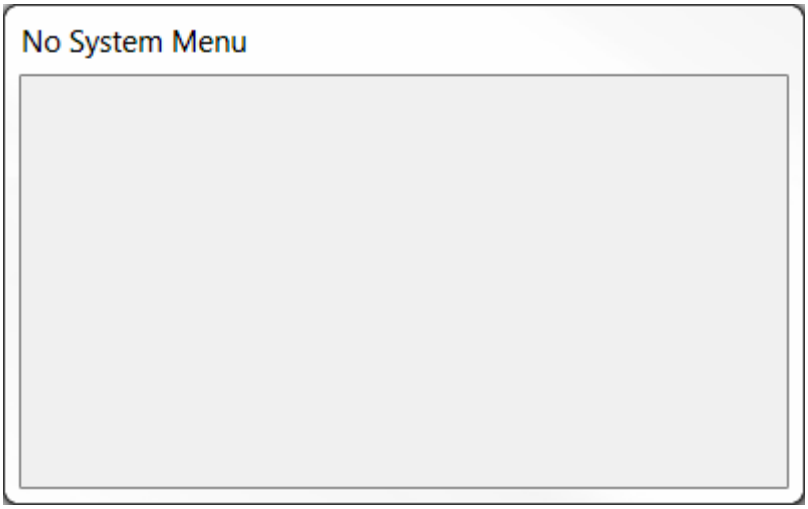


Occasionally you'll want to create a frame that cannot be resized. You could use SetSizeHints or you could just set some frame style flags. We'll be doing the latter here:

```
import wx


class NoResizeFrame(wx.Frame):
    """
    This frame cannot be resized. It can only be minimized, maximized
    and closed
    """

    def __init__(self):
        """Constructor"""
        no_resize = wx.DEFAULT_FRAME_STYLE & ~ (wx.RESIZE_BORDER |
                                                wx.RESIZE_BOX |
                                                wx.MAXIMIZE_BOX)
        wx.Frame.__init__(self, None, title="No Resize", style=no_resize)
        panel = wx.Panel(self)
        self.Show()


if __name__ == "__main__":
    app = wx.App(False)
    frame = NoResizeFrame()
    app.MainLoop()
```

Note that here we use bitwise operators to remove 3 style flags from the wx.DEFAULT_FRAME_STYLE. As you can see, this gives us a frame that we cannot resize in any way.

Create a Frame Without a System Menu



This is a rather silly requirement, but I've seen people ask for it. Basically, they want to remove ALL the buttons, but leave the title. Here's how to do that:

```
import wx


class NoSystemMenuFrame(wx.Frame):
    """
    There is no system menu, which means the title bar is there, but
    no buttons and no menu when clicking the top left hand corner
    of the frame
    """

    def __init__(self):
        """Constructor"""
        no_sys_menu = (wx.MINIMIZE_BOX | wx.MAXIMIZE_BOX | wx.RESIZE_BORDER
                       | wx.CAPTION | wx.CLIP_CHILDREN | wx.CLOSE_BOX)
        wx.Frame.__init__(self, None, title="No System Menu",
                          style=no_sys_menu)
        panel = wx.Panel(self)
        self.Show()


if __name__ == "__main__":
    app = wx.App(False)
    frame = NoSystemMenuFrame()
    app.MainLoop()
```

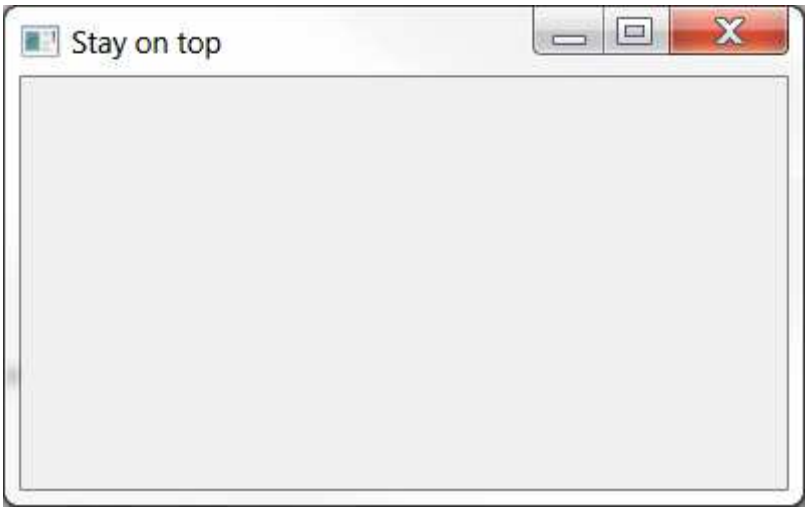
As you can see, there is a title and you can resize the frame, but you cannot maximize, minimize or close the application. Note that this worked on Windows 7. However on Linux, Windows XP and Windows 8.1, you will need to change the code to the following:

```
import wx


class NoSystemMenuFrame(wx.Frame):
    """
```

```
    There is no system menu, which means the title bar is there, but
    no buttons and no menu when clicking the top left hand corner
    of the frame
    """

    def __init__(self):
        """Constructor"""
        no_sys_menu = wx.CAPTION
        wx.Frame.__init__(self, None, title="No System Menu",
                          style=no_sys_menu)
        panel = wx.Panel(self)
        self.Show()


if __name__ == "__main__":
    app = wx.App(False)
    frame = NoSystemMenuFrame()
    app.MainLoop()
```

You will note that all we changed was to reduce the style flags down to just one: **wx.CAPTION**.

Create a Frame That Stays on Top



A lot of programmers ask about this one. They want their application to stay on top of all the others. While there isn't a completely foolproof way to accomplish this, the little recipe below will work most of the time.

```
import wx


class StayOnTopFrame(wx.Frame):
    """
    A frame that stays on top of all the others
    """

    def __init__(self):
        """Constructor"""
        on_top = wx.DEFAULT_FRAME_STYLE | wx.STAY_ON_TOP
        wx.Frame.__init__(self, None, title="Stay on top", style=on_top)
        panel = wx.Panel(self)
```

```
        self.Show()


if __name__ == "__main__":
    app = wx.App(False)
    frame = StayOnTopFrame()
    app.MainLoop()
```

Here we just use the default style flag and add on **wx.STAY_ON_TOP**.

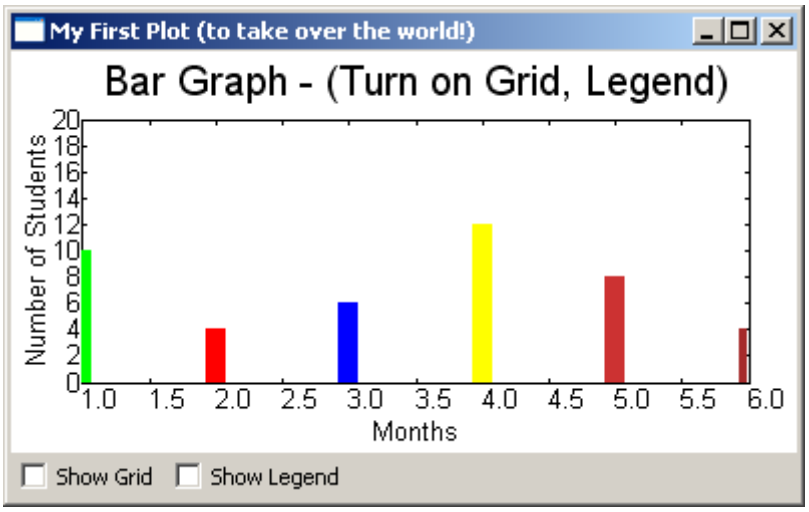
Wrapping Up

At this point, you should know how to edit almost all the frame's styles. There are a couple of other style flags that are OS dependent (like wx.ICONIZE) or just aren't that useful. If you're interested in those, check out the links below. Otherwise, go forth and use your knowledge wisely.

Creating Graphs with PyPlot

Some people learn through doing it, others are better with visual stimuli. At least, that's what we're told. So in the spirit of what we've been taught, we're going to take a look at the visual half of the equation and see how we can make graphs with wxPython. You may not know this, but wxPython includes a widget just for this purpose. It's name is PyPlot. PyPlot is great at doing simple plots and it's super fast too! If you need to weird or complicated plotting, then you'll want to use matplotlib instead. Fortunately, wxPython and matplotlib play well with each other, but we won't be looking at matplotlib in this article.

Getting Started (with a Bar Graph!)



If you look at the plot.py file in the wxPython distribution you'll discover that PyPlot requires Numeric, numarray or numpy (in reverse order) so make sure you have one of those installed to be able to use this widget. Of course, wxPython is also required, but you knew that, right?

Anyway, at the bottom of that Python file, there's a simple demo that shows how to do various graphs with PyPlot. Let's take some of that code and see if we can figure it out.

```
import wx
from wx.lib.plot import PolyLine, PlotCanvas, PlotGraphics


def drawBarGraph():
    # Bar graph
    points1=[(1,0), (1,10)]
    line1 = PolyLine(points1, colour='green', legend='Feb.', width=10)
    points1g=[(2,0), (2,4)]
    line1g = PolyLine(points1g, colour='red', legend='Mar.', width=10)
    points1b=[(3,0), (3,6)]
    line1b = PolyLine(points1b, colour='blue', legend='Apr.', width=10)

    points2=[(4,0), (4,12)]
    line2 = PolyLine(points2, colour='Yellow', legend='May', width=10)
    points2g=[(5,0), (5,8)]
    line2g = PolyLine(points2g, colour='orange', legend='June', width=10)
    points2b=[(6,0), (6,4)]
    line2b = PolyLine(points2b, colour='brown', legend='July', width=10)

    return PlotGraphics([line1, line1g, line1b, line2, line2g, line2b],
```

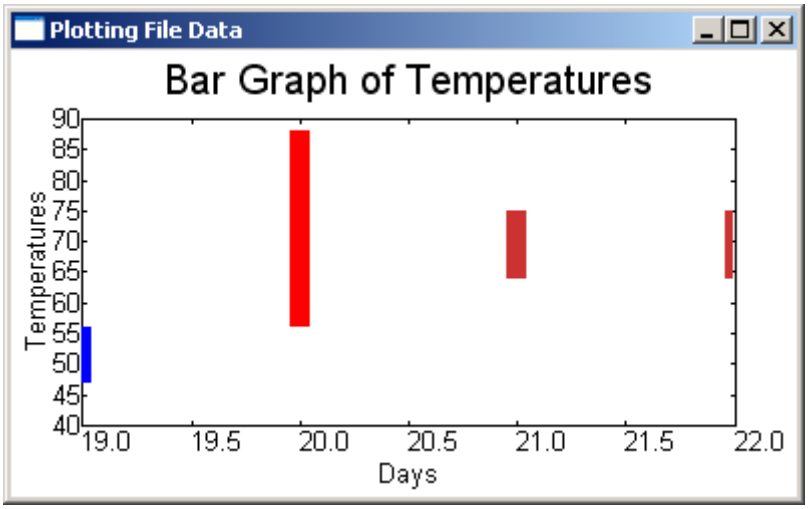
```
                        "Bar Graph - (Turn on Grid, Legend)", "Months",
                        "Number of Students")


class MyGraph(wx.Frame):


    def __init__(self):
        wx.Frame.__init__(self, None, wx.ID_ANY,
                          'My First Plot (to take over the world!)')

        # Add a panel so it looks the correct on all platforms
        panel = wx.Panel(self, wx.ID_ANY)

        # create some sizers
        mainSizer = wx.BoxSizer(wx.VERTICAL)
        checkSizer = wx.BoxSizer(wx.HORIZONTAL)

        # create the widgets
        self.canvas = PlotCanvas(panel)
        self.canvas.Draw(drawBarGraph())
        toggleGrid = wx.CheckBox(panel, label="Show Grid")
        toggleGrid.Bind(wx.EVT_CHECKBOX, self.onToggleGrid)
        toggleLegend = wx.CheckBox(panel, label="Show Legend")
        toggleLegend.Bind(wx.EVT_CHECKBOX, self.onToggleLegend)

        # layout the widgets
        mainSizer.Add(self.canvas, 1, wx.EXPAND)
        checkSizer.Add(toggleGrid, 0, wx.ALL, 5)
        checkSizer.Add(toggleLegend, 0, wx.ALL, 5)
        mainSizer.Add(checkSizer)
        panel.SetSizer(mainSizer)

    def onToggleGrid(self, event):
        """"""
        self.canvas.SetEnableGrid(event.IsChecked())

    def onToggleLegend(self, event):
        """"""
        self.canvas.SetEnableLegend(event.IsChecked())

if __name__ == '__main__':
    app = wx.App(False)
    frame = MyGraph()
    frame.Show()
    app.MainLoop()
```

The drawBarGraph function is pulled directly from the plot.py file that was mentioned earlier. For this example, the function name was changed from “_draw6Objects” to “drawBarGraph” to make the code easier to follow. Let's take a look at it. The points are the points on the graph: [(x1, y1), (x2, y2)]. They tell PyPlot where to plot via the PolyLine method. As you can see, PolyLine takes a list of tuples of graph points, and optionally, a colour, legend, width and style (not shown). We create a series of PolyLines and then add them to a PlotGraphics instance. The PlotGraphics first method is a list of PolyLines (or other PolyXXX objects), title, xLabel and yLabel. We return the PlotGraphics object back to the caller which is in our wxPython class.

Now we turn our attention to that class, which has the bland name of MyGraph. The first few lines are pretty familiar if you've used wxPython before, so let's skip those and jump right down to the widget creation section. Here we see how to create a PlotCanvas with just a plain wx.Panel as its parent. To draw the bar graph, we call our canvas object's Draw method, passing in the PlotGraphics object that was returned from the drawBarGraph function. Feel free to re-read that as many times as needed to understand what's going on before continuing.

Are you ready? Then let's continue! After we draw the bar graph, we create a couple check boxes to allow us to toggle the graph's grid and legend. Then we lay out the widgets on the frame. The check box's methods are pretty self-explanatory, so you can figure those out on your own. Hint: IsChecked() returns a Boolean.

Graphing Using Saved Data



Normally you'll want to read the data from a saved file, database or a web service rather than using hard-coded data. Here we'll look at using some saved data to create a graph. Here's the data we'll be using (you'll probably want to download the archives at the bottom of the article):

http://www.wunderground.com/history/airport/KMIW/2010/9/22/WeeklyHistory.html?format=1 CDT,Max TemperatureF,Mean TemperatureF,Min TemperatureF,Max Dew PointF,MeanDew PointF,Min DewpointF,Max Humidity, Mean Humidity, Min Humidity, Max Sea Level PressureIn, Mean Sea Level PressureIn, Min Sea Level PressureIn, Max VisibilityMiles, Mean VisibilityMiles, Min VisibilityMiles, Max Wind SpeedMPH, Mean Wind SpeedMPH, Max Gust SpeedMPH,PrecipitationIn, CloudCover, Events
 2010-9-19,56,52,47,55,49,44,100,97,93,30.21,30.17,30.11,10,5,2,14,9,20,0.34,8,Rain-Thunderstorm
 2010-9-20,88,72,56,71,62,55,100,73,46,30.10,29.94,29.77,10,6,0,25,12,32,T,4,Fog-Rain
 2010-9-21,75,70,64,66,64,63,93,83,73,29.89,29.83,29.75,10,7,0,22,7,30,1.79,5,Fog-Rain-Thunderstorm
 2010-9-22,75,70,64,68,64,63,100,93,69,30.00,29.96,29.86,10,5,1,15,4,,0.26,8,Rain
 <!-- 0.481:1 -->

The first line is the website, the second tells us what the comma delimited lines that follow are. The last four lines are plain data with some junk HTML at the end of each ling. The last line is also something we'll want to ignore. Let's create some code to actually plot this data!

```
import wx
from wx.lib.plot import PolyLine, PlotCanvas, PlotGraphics

class MyGraph(wx.Frame):

    def __init__(self):
        wx.Frame.__init__(self, None, wx.ID_ANY,
                          'Plotting File Data')
```

```
        # Add a panel so it looks the correct on all platforms
        panel = wx.Panel(self, wx.ID_ANY)
        self.canvas = PlotCanvas(panel)
        self.canvas.Draw(self.createPlotGraphics())

        sizer = wx.BoxSizer(wx.VERTICAL)
        sizer.Add(self.canvas, 1, wx.EXPAND)
        panel.SetSizer(sizer)

    def readFile(self):
        """
        Reads the hard-coded file
        """
        # normally you would want to pass a file path in, NOT hard code it!
        f = open("data.txt")
        # skip the first two lines of text in the file
        data = f.readlines()[2:-1]
        temps = []
        for line in data:
            parts = line.split(",")
            date = parts[0].split("-")
            day = date[2]
            points = [(day, parts[3]), (day, parts[1])]
            temps.append(points)
        return temps

    def createPlotGraphics(self):
        """
        Create the plot's graphics
        """
        temps = self.readFile()
        lines = []
        for temp in temps:
            tempInt = int(temp[1][1])
            if tempInt < 60:
                color = "blue"
            elif tempInt >=60 and tempInt <= 75:
                color = "orange"
            else:
                color = "red"
            lines.append(PolyLine(temp, colour=color, width=10))

        return PlotGraphics(lines, "Bar Graph of Temperatures",
                            "Days", "Temperatures")

if __name__ == '__main__':
    app = wx.App(False)
    frame = MyGraph()
    frame.Show()
    app.MainLoop()
```

Can you figure this code out? Well, if you can't (or don't want to) then you can read all about it now! Just like in our previous example, we import a few things and create a wx.Frame with a panel and a PlotCanvas. We have a simple readFile method and a createPlotGraphics method too. These two methods are what we will focus on.

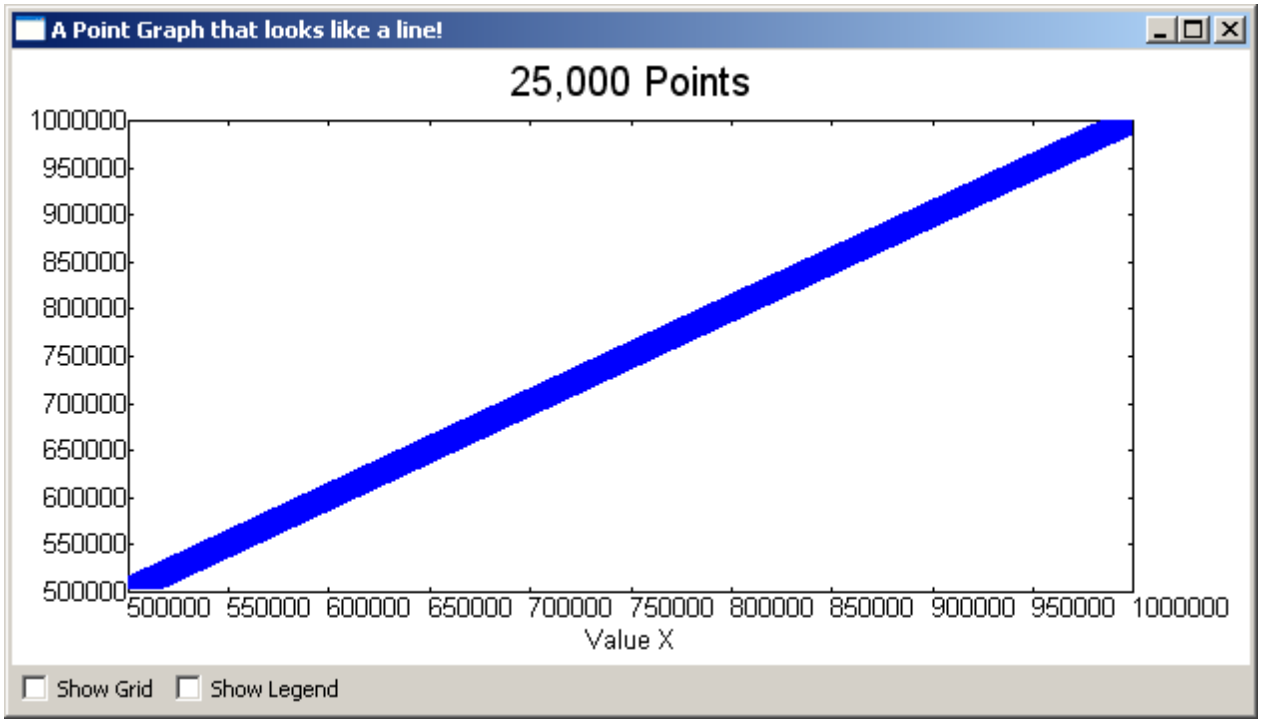
The readFile method is called by the createPlotGraphics method. All it does is read a file. For this example, we have the “path” to the file hard-coded. What you would normally want to do is use some kind of file browser to load the file, but we're going the super-simple route. When we read the lines from the file, we skip over the first two by using the following syntax:

```
data = f.readlines()[2:-1]
```

What that does is skip the first two lines in the file and read to the end, minus one line. By doing it this way, we skip the junk at the beginning and the end. Isn't Python cool? Next we create a simple “for loop” to pull out the data we want, which is just the day, the low and the high temperatures. The rest we just throw away.

In the createPlotGraphics method, we take the list of temps returned from the readFile method and loop over those, creating a new list of PolyLines. We use the some “if statements” to decide what color to make each bar in the bar graph. Finally, we put all the PolyLines into a PlotGraphics instance and return that to the called in the __init__ method. That's all there is to it!

Point Plot with Thousands of Points



Now we're going to look at how to create a point plot with 25,000 plots! This one is also from the demo. Here's the code:

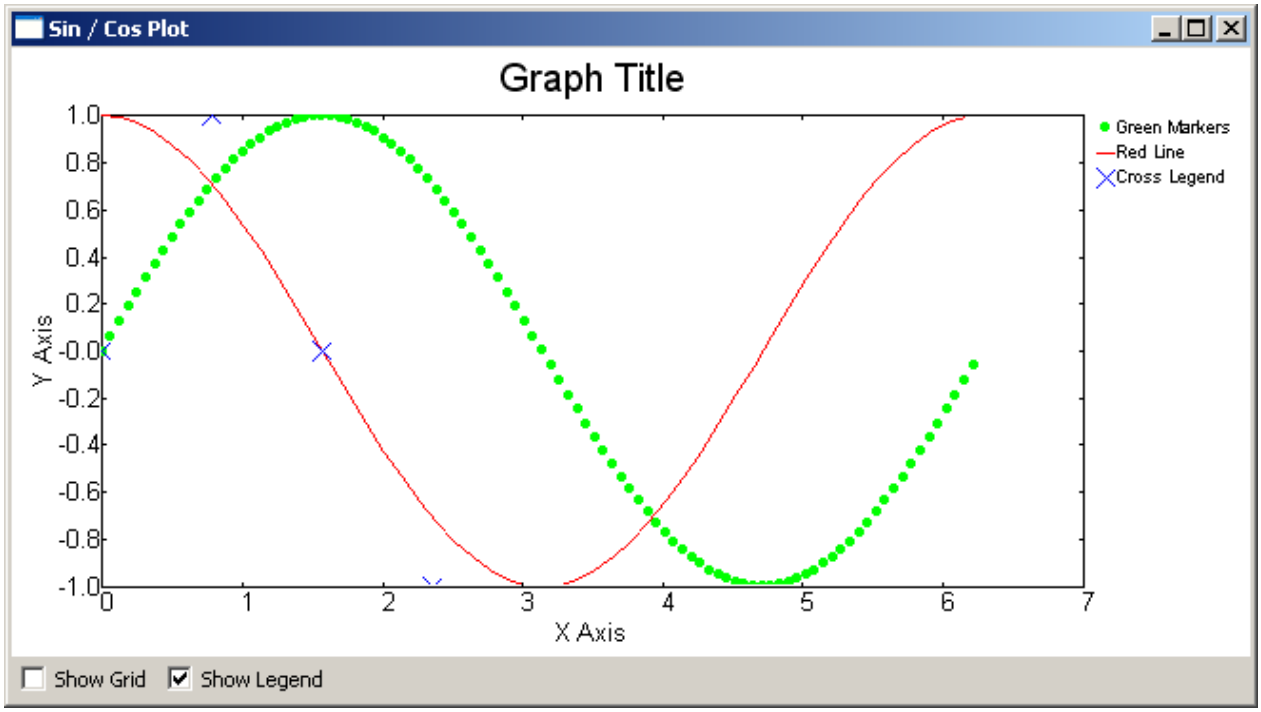
```
import numpy.oldnumeric as _Numeric
import wx
from wx.lib.plot import PlotCanvas, PlotGraphics, PolyLine, PolyMarker


def drawLinePlot():
    # 25,000 point line
    data1 = _Numeric.arange(5e5,1e6,10)
    data1.shape = (25000, 2)
    line1 = PolyLine(data1, legend='Wide Line', colour='green', width=5)
```

```
    # A few more points...
    markers2 = PolyMarker(data1, legend='Square', colour='blue',
                          marker='square')
    return PlotGraphics([line1, markers2], "25,000 Points", "Value X", "")



class MyGraph(wx.Frame):


    def __init__(self):
        wx.Frame.__init__(self, None, wx.ID_ANY,
                          'It Looks Like a Line Graph!')

        # Add a panel so it looks the correct on all platforms
        panel = wx.Panel(self, wx.ID_ANY)

        # create some sizers
        mainSizer = wx.BoxSizer(wx.VERTICAL)
        checkSizer = wx.BoxSizer(wx.HORIZONTAL)

        # create the widgets
        self.canvas = PlotCanvas(panel)
        self.canvas.Draw(drawLinePlot())
        toggleGrid = wx.CheckBox(panel, label="Show Grid")
        toggleGrid.Bind(wx.EVT_CHECKBOX, self.onToggleGrid)
        toggleLegend = wx.CheckBox(panel, label="Show Legend")
        toggleLegend.Bind(wx.EVT_CHECKBOX, self.onToggleLegend)

        # layout the widgets
        mainSizer.Add(self.canvas, 1, wx.EXPAND)
        checkSizer.Add(toggleGrid, 0, wx.ALL, 5)
        checkSizer.Add(toggleLegend, 0, wx.ALL, 5)
        mainSizer.Add(checkSizer)
        panel.SetSizer(mainSizer)


    def onToggleGrid(self, event):
        """"""
        self.canvas.SetEnableGrid(event.IsChecked())


    def onToggleLegend(self, event):
        """"""
        self.canvas.SetEnableLegend(event.IsChecked())

if __name__ == '__main__':
    app = wx.App(False)
    frame = MyGraph()
    frame.Show()
    app.MainLoop()
```

We reuse most of the wxPython code that we saw in our original example and just call a different function here. The drawLinePlot function is pretty simple. For this example, we use numpy to create the 25,000 plot points and then create a PolyLine with them. If you zoom in, you will see that some of the points are

square instead of round. That's what the PolyMarker class is for. It sets the style of the “marker”. Now we're ready to look at our next example!

Creating a Sine / Cosine Graph



This example shows you how to take a Sine and a Cosine and graph them. It kind of looks like a horizontal double-helix. Anyway, here's the code:

```
import numpy.oldnumeric as _Numeric
import wx
from wx.lib.plot import PlotCanvas, PlotGraphics, PolyLine, PolyMarker

def drawSinCosWaves():
    # 100 points sin function, plotted as green circles
    data1 = 2.*_Numeric.pi*_Numeric.arange(200)/200.
    data1.shape = (100, 2)
    data1[:,1] = _Numeric.sin(data1[:,0])
    markers1 = PolyMarker(data1, legend='Green Markers', colour='green', marker='circle'

    # 50 points cos function, plotted as red line
    data1 = 2.*_Numeric.pi*_Numeric.arange(100)/100.
    data1.shape = (50,2)
    data1[:,1] = _Numeric.cos(data1[:,0])
    lines = PolyLine(data1, legend= 'Red Line', colour='red')

    # A few more points...
    pi = _Numeric.pi
    markers2 = PolyMarker([(0., 0.), (pi/4., 1.), (pi/2, 0.),
                           (3.*pi/4., -1)], legend='Cross Legend', colour='blue',
                           marker='cross')

    return PlotGraphics([markers1, lines, markers2],"Graph Title", "X Axis", "Y Axis")
```

```
class MyGraph(wx.Frame):


    def __init__(self):
        wx.Frame.__init__(self, None, wx.ID_ANY,
                          'Sin / Cos Plot')

        # Add a panel so it looks the correct on all platforms
        panel = wx.Panel(self, wx.ID_ANY)

        # create some sizers
        mainSizer = wx.BoxSizer(wx.VERTICAL)
        checkSizer = wx.BoxSizer(wx.HORIZONTAL)

        # create the widgets
        self.canvas = PlotCanvas(panel)
        self.canvas.Draw(drawSinCosWaves())
        toggleGrid = wx.CheckBox(panel, label="Show Grid")
        toggleGrid.Bind(wx.EVT_CHECKBOX, self.onToggleGrid)
        toggleLegend = wx.CheckBox(panel, label="Show Legend")
        toggleLegend.Bind(wx.EVT_CHECKBOX, self.onToggleLegend)

        # layout the widgets
        mainSizer.Add(self.canvas, 1, wx.EXPAND)
        checkSizer.Add(toggleGrid, 0, wx.ALL, 5)
        checkSizer.Add(toggleLegend, 0, wx.ALL, 5)
        mainSizer.Add(checkSizer)
        panel.SetSizer(mainSizer)


    def onToggleGrid(self, event):
        """"""
        self.canvas.SetEnableGrid(event.IsChecked())


    def onToggleLegend(self, event):
        """"""
        self.canvas.SetEnableLegend(event.IsChecked())

if __name__ == '__main__':
    app = wx.App(False)
    frame = MyGraph()
    frame.Show()
    app.MainLoop()
```

This example is for the math geeks out there. I haven't done trigonometry or geometry in quite a while, so I won't explain the equations here. You can look up that sort of thing with your favorite search engine. This example uses one PolyLine and two PolyMarkers to create the graph. It's mostly like the other examples though, so there's really not much to say.

Wrapping Up

By now you should be more than ready to undertake doing graphs on your own with wxPython. If you get stuck, there are several other examples in the plot.py file and the wxPython mailing list members are quite friendly and will probably help you if you ask nicely. Let me know if you create anything cool!