FREE eBook

LEARNING wxpython

Free unaffiliated eBook created from
Stack Overflow contributors.

#wxpython

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: wxpython

It is an unofficial and free wxpython ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official wxpython.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with wxpython

Remarks

What Is wxPython

Simply put wxPython is a set of bindings to the **wxWidgets** C++ Cross Platform GUI library.

Ok what is wxWidgets

The wxWidgets library provides a free, gratis & open source, set of abstractions for the various GUI elements so that the native controls are still used, where available, maintaining the native look, feel & speed. As such it provides an abstraction for GUI creation and a number of other utilities in a platform that lets developers create applications for Windows, Mac OS X, Linux and other platforms using a single code base. wxWidgets was started in 1992 and you can see a detailed history here. The wxWidgets library is distributed under the wxWindows License, which is based on the **L-GPL but with an exception clause**. *The exception clause allows you to link your application either dynamically or statically to wxWidgets* ***without*** *the requirement to distribute the source for your your own application. In other words, you can use wxWidgets for either* ***free or commercial*** *projects, at* ***no cost****. The license encourages you to give back enhancements you make to the wxWidgets library itself.*

The highlights, *note that wxWidgets comprises 100s of classes for cross platform application develepment*:

- Window Layout Using Sizers
- Device Contexts (along with pens, brushes and fonts)
- Comprehensive Event Handling System
- HTML Help Viewer
- Sound and Video Playback
- Unicode and Internationalization Support
- Document/View Architecture
- Printing Archiecture
- Sockets
- Multithreading
- File and Directory Manipulation
- Online and Context-Sensitive Help
- HTML Rendering
- Basic Containers
- Image Loading, Saving, Drawing and Manipulation
- Date-Time Library and Timers
- Error Handling
- Clipboard and Drag-and-Drop

Note that some of these facilities, *e.g. threading,* are not actually GUI related but provide a useful cross platform abstraction so that, in the case of threading for example, one set of application code will work on any supported platform.

For many years the wxWidgets library, produced 4 separate builds, *in addition to debug builds* from one set of source code, static and dynamic libraries built for both ASCII and Unicode. It is usually available pre-built in the most common variants and as source code to build *with the various options* for the target environment and with the developers C++ tool chain with numerous tool chains being supported.

The python bindings for this library and some additions form wxPython.

Back to What Is wxPython, (what does it give me)?

wxPython gives a developer a way of benefiting from a cross platform GUI library, with a clear licence, while also giving the benefits of Python. Like wxWidgets and Python wxPython is free, gratis & open source, and available for use and distribution in both free and commercial projects *without a resulting requirement to distribute your source code*.

- Full GUI Suite including, (but not limited to):
  - Windows (including MDI Windows)
  - Wizards
  - Frames & MiniFrames
  - Dialogues, Standard, Advanced & Custom
  - Books, Trees, Grids & Data View Controls
  - Gauges, Sliders, Spinners, Animations, Clipboard, Drag & Drop
  - HTML, PDF & Image viewer support
  - GUI components can be absolutely positioned but it is strongly recommended to use sizer based layout which support auto sizing, etc.
- Cross Platform - Support GUIs for Windows, OS-X & Linux with a single code base *without conditional statements in your code*
- Native speed, look & feel.
- Rapid prototype, test & debug - *remember that this is python*
- Run & edit samples of just about everything in the demo package.
- Clear licence for gratis use even in commercial products.
- If necessary your python GUI can be refactored to a C++ wxWidgets GUI later as it is already using it.
- Large, active & helpful user & developer community both on StackOverflow and mailing lists.

Note that where python itself provides a cross platform mechanism for implementing the utility functions of wxWidgets, *threading again being a good example*, it is **intentionally** omitted from wxPython.

wxPython also has a very large suite of demonstrations that can be run, tested and edited from

within the Documents and Demo package.

Flavours of wxPython

ASCII vs Unicode:

For many years, *as with wxWidgets*, developers had to choose between ASCII and Unicode builds as well as needing a build for their specific version of python as well as the 32/64 bit options. As of about wxPython 2.8.9 the ASCII only build of wxPython has been dropped so Unicode support is always available.

Classic vs. Phoenix:

Since wxPython 3.0.0 there have existed the *released* "Classic" build of wxPython and a Phoenix *currently unreleased* build. The classic build tends to lag behind the wxWidgets builds of the same numbers and the documentation package is the C++ - it is available for download for various platforms, (see Installation of Classic), in the case of windows as an executable installer. The Phoenix bindings, being largely automatically generated, should follow more closely on the wxWidgets builds and also include wxPython specific documentation - it is build-able from source or nightly builds *as wheels* can be obtained using **pip**, (see Installation of Phoenix).

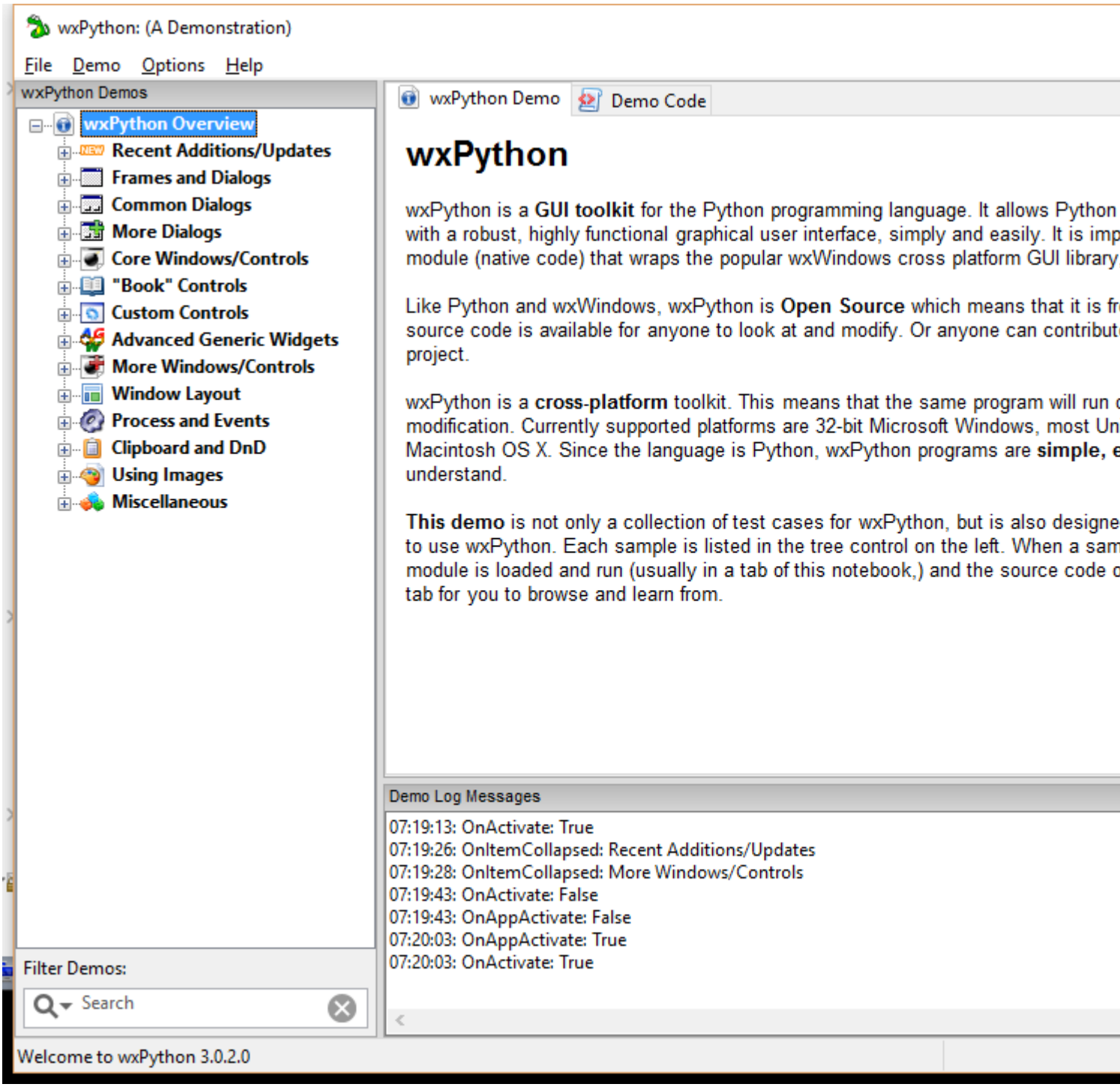
In wxPython but not wxWidgets

wxPython extends the wxWidgets library with a number of features, *the following are just a few,* that are not available in wxWidgets:

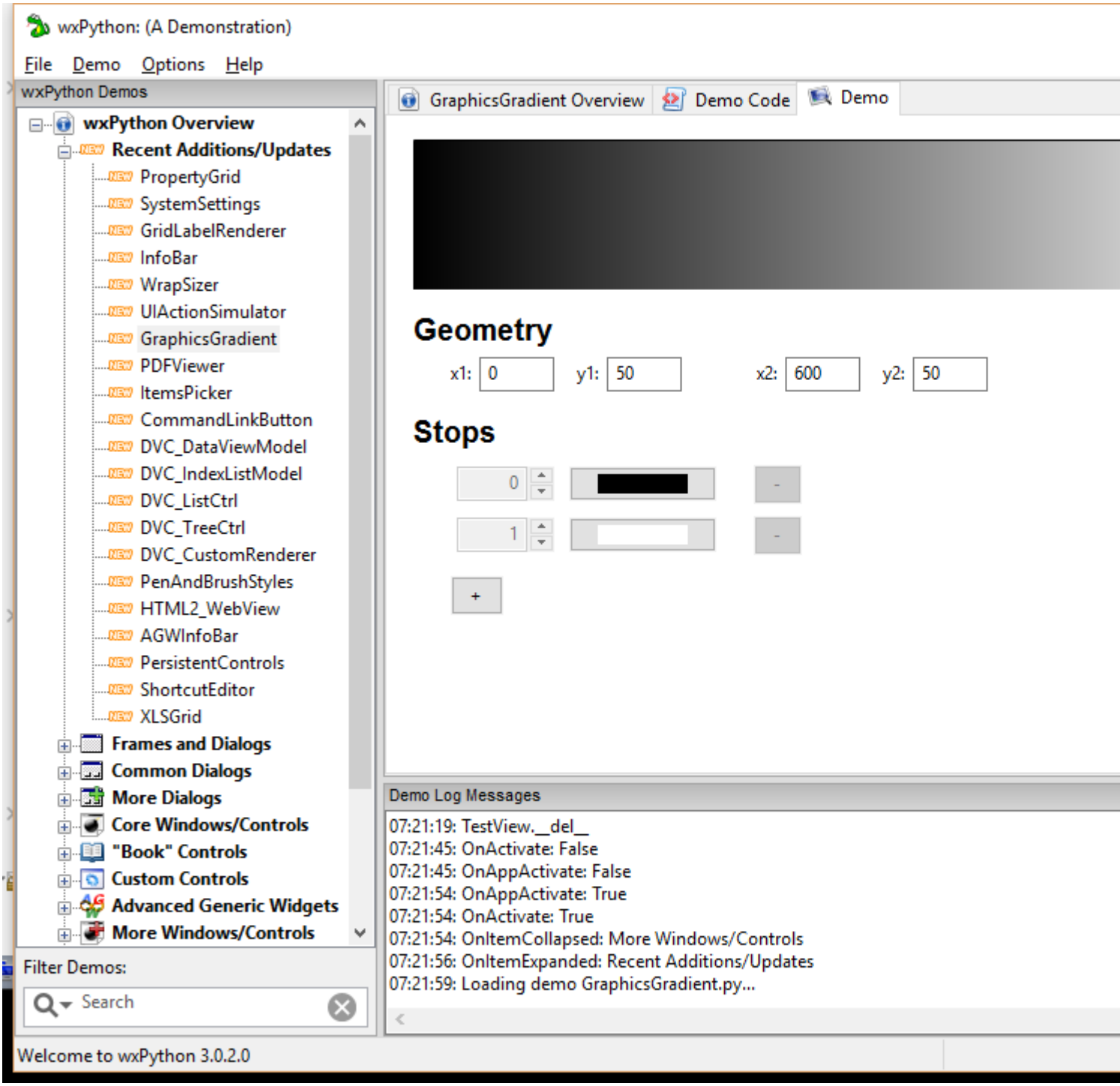
- Programmers Editors & Shells: crust, crustslices, AlaCart & AlaMode, AlaModeTest
- Interpreter & magic
- Inspection - this allows you to launch a window to browse all of your applications GUI components.
- An extensive set of Demos

Demo Screenshots *on Win10*

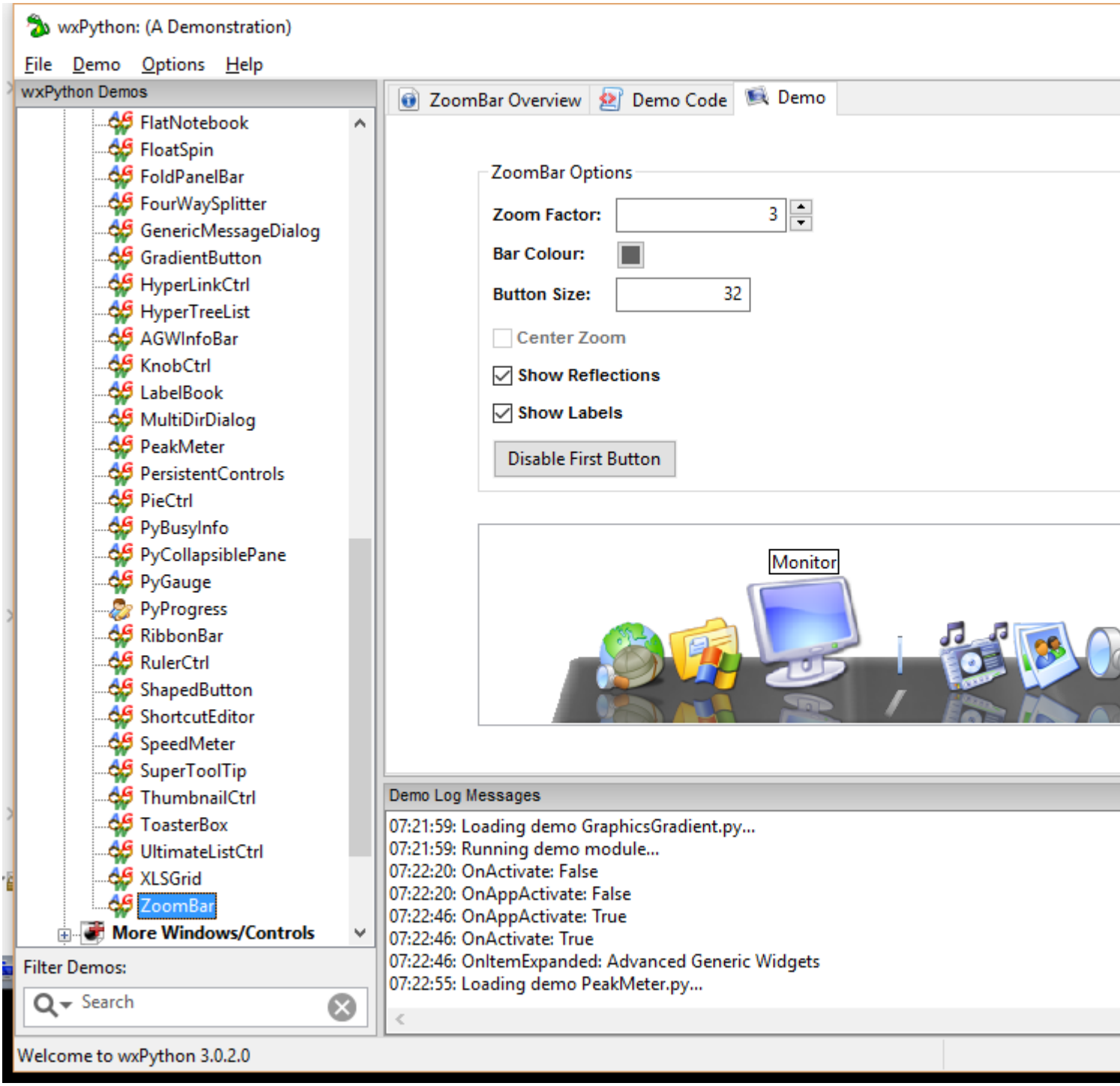
The wxPython demo with all the branches closed:



One of the recent additions:



One of the AGW, (Advanced Generic Widgets):



Examples

Installation of wxPython Phoenix

wxPython Phoenix is the latest version of wxPython, (currently *Sept 2016* without an official release). It supports both Python 2 and Python 3. You can download a snapshot build (i.e. a Python wheel) for your platform and Python version here.

wxPython Phoenix utilizes a largely automated mechanism for generating both the python bindings for the wxWidgets library and the documentation. Phoenix wxPython documentation is specifically generated for itself using Sphinx. This increases clarity as opposed to C++

documentation of the classic build, which includes many overloads that are not available in wxPython.

Python and **pip** must be installed before wxPython Phoenix can be installed.

You can use pip to install the Phoenix version of wxPython. Here is the recommended method currently:

```
python -m pip install --no-index --find-links=http://wxpython.org/Phoenix/snapshot-builds/ --trusted-host wxpython.org wxPython_Phoenix
```

When you use this command, pip will also install **wxWidgets**. This complex pip command will likely become 'pip install wxpython' when Phoenix is officially released.

Note: wxPython Phoenix is currently in beta and doesn't have all the widgets that the Classic version has.

Installation of wxPython Classic

wxPython Classic is a **Python 2** build of the wxPython library. Generation of the python bindings require a large number of manual interventions and the documentation is simply the wxWidgets documentation which contains some annotations on wxPython mechanisms as such there is normally a delay of weeks to months between a new release of wxWidgets and the matching release of wxPython.

Go to the download page on the wxPython website to see if there is already a version of wxPython that you can download for your platform.

The latest version of Classic is **3.0.2.0**

Windows

There are installers for Python 2.6 and 2.7 for 32-bit and 64-bit Windows platforms on the website. Just download one of these and run them to install it.

Note: Make sure you download a wxPython installer for the right Python you have installed. For example, if you have Python 2.7 32-bit, then you want a wxPython 32-bit installer

Mac

If you have OSX **10.5 or above**, then you will want to download and install the **Cocoa** version of wxPython. The Cocoa version also supports 64-bit Mac.

If you have a Mac with a version of OSX less than **10.5**, then you will want the **Carbon** build.

Linux

The first thing to check if your Linux platform's package manager (i.e. yum, apt-get, etc) to see if it has a version of wxPython that you can install. Unfortunately, a lot of Linux packages for wxPython are for version 2.8.12.1 instead of 3.0.2.0. If your package manager doesn't have the latest

version, you will probably have to build it yourself.

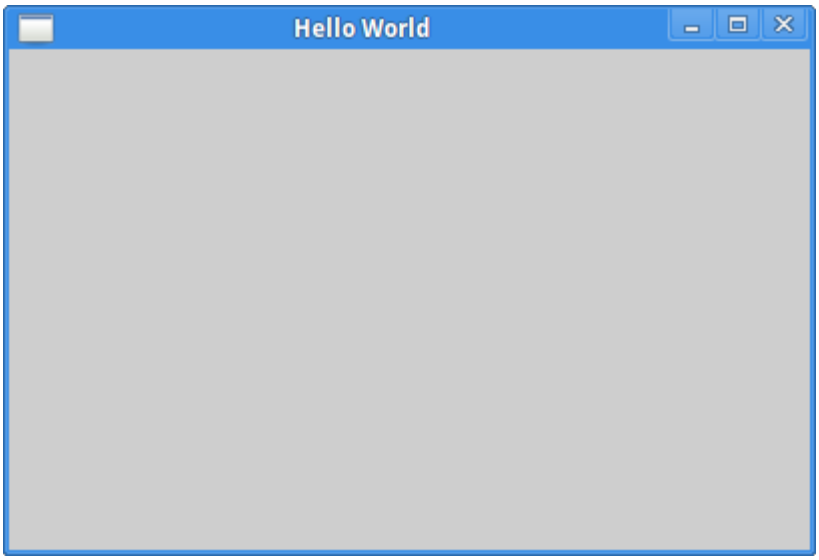
There are build instructions for 3.0.2.0-Classic here

Hello World

A simple way to create a **Hello World** program:

```
import wx
app = wx.App(redirect=False)
frame = wx.Frame(parent=None, id=wx.ID_ANY, title='Hello World')
frame.Show()
app.MainLoop()
```

Output:



A more typical example would be to subclass **wx.Frame**:

```
import wx

class MyFrame(wx.Frame):

    def __init__(self):
        wx.Frame.__init__(self, None, title='Hello World')
        self.Show()

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

This can also be rewritten to use Python's **super**:

```
import wx

class MyFrame(wx.Frame):

    def __init__(self, *args, **kwargs):
```

```
        """Constructor"""
        super(MyFrame, self).__init__(*args, **kwargs)
        self.Show()

if __name__ == '__main__':
    app = wx.App(False)
    frame = MyFrame(None, title='Hello World')
    app.MainLoop()
```

What is a wxPython Release Series?

The wxWidgets project has adopted the release model used by the Linux Kernel project where there are alternating sets of releases where one set are considered "stable" and the next set are considered "development." For wxWidgets "stable" and "development" do not refer to bugginess, but to the stability of the API and backwards compatibility.

- **Stable**: For the duration of the series existing APIs are not modified, although new non-virtual class methods and such can be added. Binary compatibility of the C++ libs is maintained by not allowing any changes that modify the in-memory size or layout of the classes and structs. This can and often does impose limitations on what kinds of enhancements or bug fixes can be performed in a stable release series, however this really only affects the C++ layer because in Python being backwards compatible has a slightly different connotations.

- **Development**: The main purpose of the development series of releases is to add new functionality or to correct problems that could not be corrected in a stable series because of binary compatibility issues, all in an effort to create the next stable series. So for the duration of the development series existing the APIs are allowed to be modified or removed as needed, although most of the time C++ source-level compatibility is maintained via deprecated overloaded functions or macros, etc. For wxPython this often means that there will be source-level incompatibilities because there is no overloading or macros, and in order to support the new version of the API sometimes the old version has to be removed.

Because of the binary compatibility issues, the latest development version of wxWidgets/wxPython can often be less buggy than the latest version of the last stable release series. However there is the trade-off that the APIs may be changing or evolving between versions in the development series.

How do the version numbers work?

For releases wxPython uses a 4 component version number. While this looks a lot like how version numbers are used in other Open Source projects, there are a few subtle differences. So for some release **A.B.C.D** you can deduce the following:

1. **Release Series**: The first two components of the version number (**A.B**) represent the release series, and if the **B** component is an even number then it is a stable series, if it is an odd number then it is an development release series. For example, 2.4, 2.6, and 2.8 are stable and the API is more or less frozen within each series, and 2.3, 2.5, and 2.7 are development and the API and functionality is allowed to change or evolve as needed.

Because of this there can be quite large changes between one stable series to the next (say 2.4 to 2.6) and this often throws people off because in other projects changes of that magnitute would have caused the first component of the version number to change. Instead you should think of the combination of **A.B** as being the major number of the version.

2. **Release Number**: The third component of the version number (C) represents one of the releases in a release series. For example, 2.5.0, 2.5.1, 2.5.2, 2.5.3... are all releases in the 2.5 release series. (And since in this case it is an development series then the API and functionality of 2.5.3 has evolved to be different in places than it was in 2.5.0.) The C++ wxWidgets releases usually stop here and only A.B.C releases are made.

3. Subrelease number, or wxPython release: The fourth component of the version number (D) is used to represent a subrelease, or incremental releases betweeen the official wxWidgets releases. These releases include fixes for wxWidgets bugs that wxPython may have exposed, or minor enhancements that are important for wxPython. This is not an arbitrary wxWidgets snapshot, but rather a tested version of the code with fixes and enhancements not yet available from wxWidgets except from the source code repository.

Source: https://wiki.wxpython.org/ReleaseSeries

Read Getting started with wxpython online: https://riptutorial.com/wxpython/topic/6690/getting-started-with-wxpython

Chapter 2: Drag and Drop

Introduction

wxPython provides several different kinds of drag and drop. You can have one of the following types: `wx.FileDropTarget`, `wx.TextDropTarget`, or `wx.PyDropTarget`.

The first two are pretty self-explanatory. The last one, wx.PyDropTarget, is just a loose wrapper around wx.DropTarget itself. It adds a couple extra convenience methods that the plain wx.DropTarget doesn't have. We'll start with a wx.FileDropTarget example.

Examples

FileDropTarget

```
import wx


class MyFileDropTarget(wx.FileDropTarget):
    """"""

    def __init__(self, window):
        """Constructor"""
        wx.FileDropTarget.__init__(self)
        self.window = window

    def OnDropFiles(self, x, y, filenames):
        """
        When files are dropped, write where they were dropped and then
        the file paths themselves
        """
        self.window.SetInsertionPointEnd()
        self.window.updateText("\n%d file(s) dropped at %d,%d:\n" %
                               (len(filenames), x, y))
        for filepath in filenames:
            self.window.updateText(filepath + '\n')

        return True


class DnDPanel(wx.Panel):
    """"""

    def __init__(self, parent):
        """Constructor"""
        wx.Panel.__init__(self, parent=parent)

        file_drop_target = MyFileDropTarget(self)
        lbl = wx.StaticText(self, label="Drag some files here:")
        self.fileTextCtrl = wx.TextCtrl(self,
                                        style=wx.TE_MULTILINE|wx.HSCROLL|wx.TE_READONLY)
        self.fileTextCtrl.SetDropTarget(file_drop_target)
```

```
        sizer = wx.BoxSizer(wx.VERTICAL)
        sizer.Add(lbl, 0, wx.ALL, 5)
        sizer.Add(self.fileTextCtrl, 1, wx.EXPAND|wx.ALL, 5)
        self.SetSizer(sizer)

    def SetInsertionPointEnd(self):
        """
        Put insertion point at end of text control to prevent overwriting
        """
        self.fileTextCtrl.SetInsertionPointEnd()

    def updateText(self, text):
        """
        Write text to the text control
        """
        self.fileTextCtrl.WriteText(text)


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

    def __init__(self):
        """Constructor"""
        wx.Frame.__init__(self, parent=None, title="DnD Tutorial")
        panel = DnDPanel(self)
        self.Show()


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

TextDropTarget

```
import wx


class MyTextDropTarget(wx.TextDropTarget):

    def __init__(self, textctrl):
        wx.TextDropTarget.__init__(self)
        self.textctrl = textctrl

    def OnDropText(self, x, y, text):
        self.textctrl.WriteText("(%d, %d)\n%s\n" % (x, y, text))
        return True

    def OnDragOver(self, x, y, d):
        return wx.DragCopy


class DnDPanel(wx.Panel):
    """"""

    def __init__(self, parent):
        """Constructor"""
        wx.Panel.__init__(self, parent=parent)
```

```
        lbl = wx.StaticText(self, label="Drag some text here:")
        self.myTextCtrl = wx.TextCtrl(
            self, style=wx.TE_MULTILINE|wx.HSCROLL|wx.TE_READONLY)
        text_dt = MyTextDropTarget(self.myTextCtrl)
        self.myTextCtrl.SetDropTarget(text_dt)

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

    def WriteText(self, text):
        self.text.WriteText(text)


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

    def __init__(self):
        """Constructor"""
        wx.Frame.__init__(
            self, parent=None, title="DnD Text Tutorial")
        panel = DnDPanel(self)
        self.Show()


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

PyDropTarget

```
import  wx


class MyURLDropTarget(wx.PyDropTarget):

    def __init__(self, window):
        wx.PyDropTarget.__init__(self)
        self.window = window

        self.data = wx.URLDataObject();
        self.SetDataObject(self.data)

    def OnDragOver(self, x, y, d):
        return wx.DragLink

    def OnData(self, x, y, d):
        if not self.GetData():
            return wx.DragNone

        url = self.data.GetURL()
        self.window.AppendText(url + "\n")

        return d


class DnDPanel(wx.Panel):
```

```
    """"""

    def __init__(self, parent):
        """Constructor"""
        wx.Panel.__init__(self, parent=parent)
        font = wx.Font(12, wx.SWISS, wx.NORMAL, wx.BOLD, False)

        # create and setup first set of widgets
        lbl = wx.StaticText(self,
                            label="Drag some URLS from your browser here:")
        lbl.SetFont(font)
        self.dropText = wx.TextCtrl(
            self, size=(200,200),
            style=wx.TE_MULTILINE|wx.HSCROLL|wx.TE_READONLY)
        dt = MyURLDropTarget(self.dropText)
        self.dropText.SetDropTarget(dt)
        firstSizer = self.addWidgetsToSizer([lbl, self.dropText])

        # create and setup second set of widgets
        lbl = wx.StaticText(self, label="Drag this URL to your browser:")
        lbl.SetFont(font)
        self.draggableURLText = wx.TextCtrl(self,
                                            value="http://www.mousevspython.com")
        self.draggableURLText.Bind(wx.EVT_MOTION, self.OnStartDrag)
        secondSizer = self.addWidgetsToSizer([lbl, self.draggableURLText])

        # Add sizers to main sizer
        mainSizer = wx.BoxSizer(wx.VERTICAL)
        mainSizer.Add(firstSizer, 0, wx.EXPAND)
        mainSizer.Add(secondSizer, 0, wx.EXPAND)
        self.SetSizer(mainSizer)

    def addWidgetsToSizer(self, widgets):
        """
        Returns a sizer full of widgets
        """
        sizer = wx.BoxSizer(wx.HORIZONTAL)
        for widget in widgets:
            if isinstance(widget, wx.TextCtrl):
                sizer.Add(widget, 1, wx.EXPAND|wx.ALL, 5)
            else:
                sizer.Add(widget, 0, wx.ALL, 5)
        return sizer

    def OnStartDrag(self, evt):
        """"""
        if evt.Dragging():
            url = self.draggableURLText.GetValue()
            data = wx.URLDataObject()
            data.SetURL(url)

            dropSource = wx.DropSource(self.draggableURLText)
            dropSource.SetData(data)
            result = dropSource.DoDragDrop()


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

    def __init__(self):
        """Constructor"""
```

```
        wx.Frame.__init__(self, parent=None,
                          title="DnD URL Tutorial", size=(800,600))
        panel = DnDPanel(self)
        self.Show()


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

Read Drag and Drop online: https://riptutorial.com/wxpython/topic/9709/drag-and-drop

Credits

S. No	Chapters	Contributors
1	Getting started with wxpython	4444, Boštjan Mejak, Community, Mike Driscoll, Steve Barnes
2	Drag and Drop	Mike Driscoll