

Learning Python

Getting results for beamlines and scientific programming

Intermediate python: GUI programming with WxPython, part 3

Outline of topics to be covered

1. Quick wxPython (Sizers/Widgets/widget id) review
2. Responding to events
3. Model-View-Controller
4. Designing event handlers
5. Validators
6. Multiple windows (modal & non-modal)

Note that code used in examples in this talk are posted in a file oct19Examples.zip on the Python Interest Group (PIG) web page.





REVIEW OF WXPYTHON

Short wxPython sample application

Subclass wx.Frame

- Call wx.Frame `__init__`
- Create Panel as child of Frame (self)
- Create sizer & associate with panel
- Create widgets & add to sizer
- Size window to match widgets
- Set “show” flag on Frame

Create App

Create Frame

Start event loop

```
import wx
class mywxframe(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None)
        pnl = wx.Panel(self)
        sizer = wx.BoxSizer(wx.VERTICAL)
        pnl.SetSizer(sizer)
        w = wx.StaticText(
            pnl,wx.ID_ANY,'Widget #1')
        sizer.Add(w,0,wx.ALL,10)
        w = wx.StaticText(
            pnl,wx.ID_ANY,'... Widget #2')
        sizer.Add(w,0,wx.ALL,10)
        sizer.Fit(self)
        self.Show()

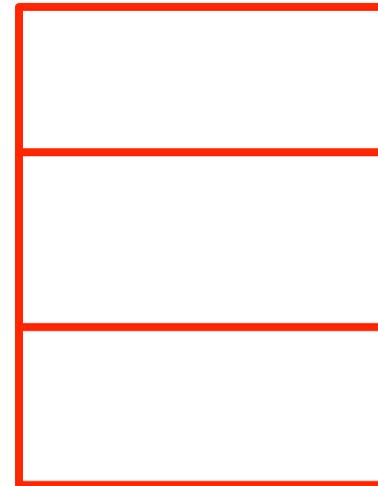
app = wx.PySimpleApp() # create the App
frm = mywxframe()
app.MainLoop()
wx4.py
```



We use Sizers to arrange widgets

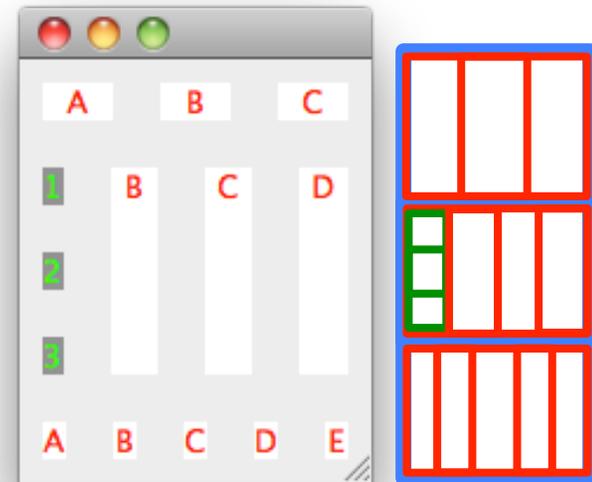


`wx.BoxSizer(wx.HORIZONTAL)`



`wx.BoxSizer(wx.VERTICAL)`

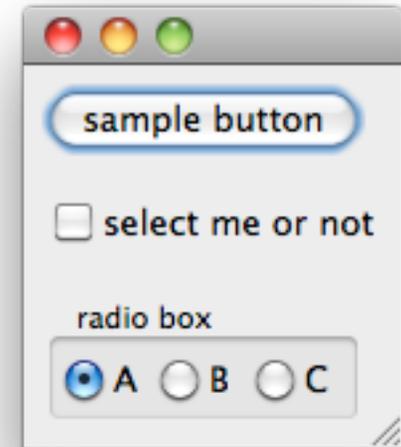
Sizers can be placed in sizers for more complex arrangements



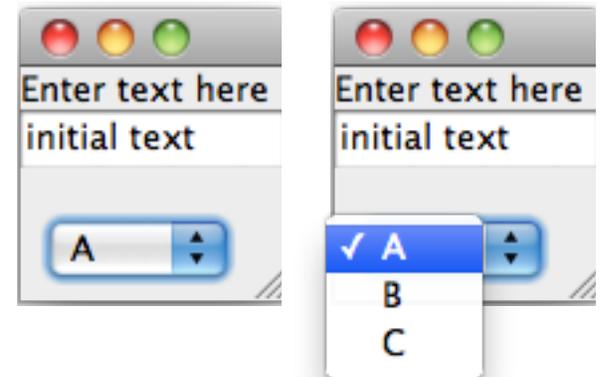
Commonly used widgets

Making widgets tends to be easy

- **wx.Button**: creates a widget intended to respond when “clicked” on by a mouse. Usually labeled with text, but a variation, the **wx.BitmapButton**, is labeled with a bitmap image.
- **wx.CheckBox**: creates a widget that can be selected or unselected. An option is a 3-state button that can be on, off or other.
- **wx.RadioButton** where only one button in a group can be selected.
- **wx.TextCtrl**: creates a widget where a user can enter text. May be single line or multiline. Probably the most widely used widget next to buttons.
- **wx.Choice**: offers a menu with a fixed list of options. (Also see **wx.ComboBox**.)



wx8.py



wx8a.py



Widget id's

Every widget in a window (frame) must have a unique id code. Usually this is done by setting the id parameter when a widget is created to be `wx.ID_ANY` (which is -1). This is often the default. If `wx.ID_ANY` is used, wxPython creates a unique id.

```
w = wx.StaticText(pnl,wx.ID_ANY,'Widget #1')
```

Most of the time one does not care about the id value. When you do need to know the Id of a widget, it can be obtained using `<widget>.GetID()`.

```
id = w.GetID()
```

One exception, (to be shown later) there are some commonly used widgets, such as Exit in menus, Yes & No buttons in dialogs,... wxPython defines standard Ids for these and it is best to use them.





RESPONDING TO EVENTS

Needed to “wire up” buttons, etc. to code



Binding an event to widgets

This is the preferred way to make a binding. It uses a method of the widget (button) to connect to the callback.

This routine referenced in the bind will be called when the event (`wx.EVT_BUTTON`) is generated when a button is pressed.

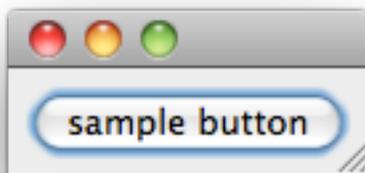
```
class mywxframe(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None)
        pnl = wx.Panel(self)
        sizer = wx.BoxSizer(wx.VERTICAL)
        pnl.SetSizer(sizer)
        btn = wx.Button(pnl, -1,
                        "sample button")
        btn.Bind(wx.EVT_BUTTON, self.OnClick)
        sizer.Add(btn, 0, wx.ALL, 10)

        sizer.Fit(self)
        self.Show()

    def OnClick(self, event):
        print 'Button pressed'
```

wx9.py

```
bht3:tutorial toby$ python wx9.py
Button pressed
Button pressed
```



Other commonly used events

<code>wx.EVT_BUTTON</code>	button press
<code>wx.EVT_CHECKBOX</code>	called when the appropriate widget is operated
<code>wx.EVT_COMBOBOX</code>	
<code>wx.EVT_SLIDER</code>	
<code>wx.EVT_RADIOBOX</code>	
<code>wx.EVT_MENU</code>	
<code>wx.EVT_CLOSE</code>	called when a frame (window) is closed/changed
<code>wx.EVT_SIZE</code>	
<code>wx.EVT_TEXT_ENTER</code>	typing in a TextCtrl
<code>wx.EVT_KILL_FOCUS</code>	generated when a widget is entered or left
<code>wx.EVT_SET_FOCUS</code>	
<code>wx.EVT_KEY_UP</code>	when a keyboard key is released
<code>wx.EVT_MOUSEWHEEL</code>	when a mouse button is pressed or the mouse is moved
<code>wx.EVT_LEFT_DOWN</code>	
<code>wx.EVT_MOTION</code>	



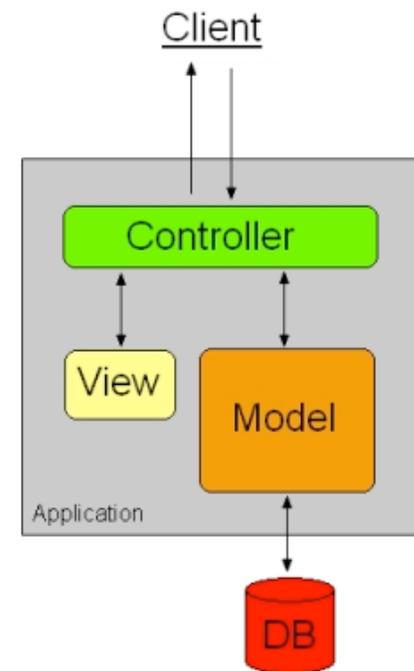
Aside: The Model-View-Controller Design Paradigm

It is almost always a good idea to write code in three separate sections:

The model: which contains the program's data structures and "domain logic"

The view: which displays the information in the Model to the user

The controller: this allows the user to interact and perhaps modify the model



Model-View-Controller in Scientific Coding

- **The Model** usually corresponds to the experimental data, fitting parameters, and all the computational code used with that data.
- **The View** will be the tables and graphics that are generated to allow the user to visualize the contents of the Model
- **The Controller** will be the GUI

These should all be in separate classes and, preferably, separate modules.

Why?

1. You may want to have an exclusively computational program that operates for long periods or runs on a cluster. This will need to use the Model without any code from the View or Controller
2. You may want to replace or augment the visualization, and do not want to have to replace all code
3. You may decide to create different types of GUIs and again want to reuse as much code as possible.

Note that this is only a paradigm. For scientific software, sometimes it makes sense to combine the view and controller, or more rarely, the model and view, but always keep the GUI separate from the computation/data storage module(s).



Getting information into an event handler

- One of the problems in event-driven programming is that events live in the Controller layer, but affect parameters in the Model and/or operation of the View. Even though when will have access to the Model and View objects when the code to build the GUI and setup the bind is executing, there is no way to pass this into the event handler.
- One choice is to place the event handler in the Model or View objects, but this has problems since this is often GUI-specific code that really should live in the Controller.



Example of the information routing problem

This code has two classes:

- mycalc (for data/computation) &
- myframe (GUI implementation)

we really want them separate

My hokey GUI example: pressing button reads TextCtrl and then disables it so it can't be changed.

The GUI part of this is implemented here, but I need to get the value I read into the mycalc object.

- How can I do this?

```
import wx
class mycalc:
    def __init__(self):
        self.myvar = None
    def SetVar(self, val):
        self.myvar = val

class myframe(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None)
        pnl = wx.Panel(self)
        sizer = wx.BoxSizer(wx.VERTICAL)
        pnl.SetSizer(sizer)
        btn = wx.Button(pnl, -1, "read")
        btn.Bind(wx.EVT_BUTTON, self.OnClick)
        sizer.Add(btn, 0, wx.ALL, 10)
        self.tc = wx.TextCtrl(pnl, -1,
                              'xxx', size=(100, -1))
        sizer.Add(self.tc, 0, wx.ALL, 10)
        sizer.Fit(self)
        self.Show()

    def OnClick(self, event):
        print 'Value = ', self.tc.GetValue()
        # but how do I get value into CalcObj?
        self.tc.Disable() # disable the TextCtrl

app = wx.PySimpleApp() # create the App
CalcObj = mycalc()
frm = myframe()
app.MainLoop()
```



wx10.py

Solution: save pointers to required objects

Note that the only way to get information into the event handler is by defining variables in the class for later use. This is usually done in the `__init__` function for the class.

When I create frm (class myframe), I pass it a reference to the data object (CalcObj, class mycalc).

This gets saved in the myframe class

The OnClick method can then access the data object

```
...
class myframe(wx.Frame):
    def __init__(self, DataObj):
        wx.Frame.__init__(self, None)
        self.DataObj = DataObj
        pnl = wx.Panel(self)
        sizer = wx.BoxSizer(wx.VERTICAL)
        pnl.SetSizer(sizer)
        btn = wx.Button(pnl, -1, "read")
        btn.Bind(wx.EVT_BUTTON, self.OnClick)
        sizer.Add(btn, 0, wx.ALL, 10)
        self.tc = wx.TextCtrl(pnl, -1,
                               'xxx', size=(100, -1))
        sizer.Add(self.tc, 0, wx.ALL, 10)
        sizer.Fit(self)
        self.Show()

    def OnClick(self, event):
        # here I put the value into CalcObj:
        self.DataObj.SetVar(self.tc.GetValue())
        self.tc.Disable() # disable the TextCtrl

app = wx.PySimpleApp() # create the App
CalcObj = mycalc()
frm = myframe(CalcObj)
app.MainLoop()
print 'saved val = ', CalcObj.myvar
```

wx10a.py

Implementing a generic event handler

- Save info in dict(s) indexed by widget Id or by widget object that will be used to determine what the event handler will do
- If the event handler will need to do different things based on what type of event occurred, also look up the “event type Id”

(for example `eventId = wx.EVT_BUTTON.typeId`).

Use a nested dict?

In the event handler use event parameter to get information on the widget causing the event and the event type:

`event.GetId()`: provides the Id of the widget that generated the event

`event.GetEventObject()`: provides the object that generated the event

`event.GetEventType()`: provides the event type Id

- To see what else is available, use `print (dir (event))`. Note that different types of events have different methods and values defined,



Generic Event Handler Example

```
class mywxframe(wx.Frame):
    def __init__(self,DataObj):
        wx.Frame.__init__(self, None)
        self.DataObj = DataObj
        self.DisableList = {}
        self.SetList = {}
        pnl = wx.Panel(self)
        sizer = wx.BoxSizer(wx.VERTICAL)
        pnl.SetSizer(sizer)
        btn = wx.Button(pnl, -1,"read")
        btn.Bind(wx.EVT_BUTTON, self.OnClick)
        sizer.Add(btn,0,wx.ALL,10)
        self.tc = wx.TextCtrl(pnl, -1,
                               'xxx',size=(100,-1))

        id = btn.GetId()
        print 'Button id =',id
        self.DisableList[id] = self.tc
        self.SetList[id] = self.DataObj.SetVar

        sizer.Add(self.tc,0,wx.ALL,10)
        sizer.Fit(self)
        self.Show()
```

wx10b.py

```
def OnClick(self, event):
    id = event.GetId()
    print 'Button pressed is =',id
    sl = self.SetList.get(id)
    if sl is not None:
        sl(self.tc.GetValue())
    disabl = self.DisableList.get(id)
    if disabl is not None:
        disabl.Disable() # disable
```

```
bht3:tutorial toby$ python wx10b.py
Button id = -203
Button pressed is = -203
saved val = yyy
```



Another variant, use object as a key rather than id

```
class mywxframe(wx.Frame):
    def __init__(self,DataObj):
        wx.Frame.__init__(self, None)
        self.DataObj = DataObj
        self.DisableList = {}
        self.SetList = {}
        pnl = wx.Panel(self)
        sizer = wx.BoxSizer(wx.VERTICAL)
        pnl.SetSizer(sizer)
        btn = wx.Button(pnl, -1,"read")
        btn.Bind(wx.EVT_BUTTON, self.OnClick)
        sizer.Add(btn,0,wx.ALL,10)
        self.tc = wx.TextCtrl(pnl, -1,
                               'xxx',size=(100,-1))
        print 'Button =',btn
        self.DisableList[btn] = self.tc
        self.SetList[btn] = self.DataObj.SetVar

        sizer.Add(self.tc,0,wx.ALL,10)
        sizer.Fit(self)
        self.Show()
```

wx10c.py

```
def OnClick(self, event):
    btn = event.GetEventObject()
    print 'EventObject =',btn
    sl = self.SetList.get(btn)
    if sl is not None:
        sl(self.tc.GetValue())
    dis = self.DisableList.get(btn)
    if dis is not None:
        dis.Disable() # disable
```



Validators

Validators are classes that can be associated with TextCtrl widgets. The validator can check each character as it is typed into the box (and possibly ignore invalid characters) or it can check if the contents is valid.

I have a pair of sample validators on the PIG web page (file validators.py:

<https://confluence.aps.anl.gov/download/attachments/2523138/validators.py>)

- One enforces that the values entered match that of a unsigned or optionally signed integer. Any other character is ignored.
 - When the box loses focus, the entire contents is checked to make sure it is a valid integer. If not, the widget is highlighted
 - Optionally a function can be supplied to execute on a valid integer
 - Minimum and maximum values can be specified
- The other is for floats, optionally with exponential notation (e or E is allowed)
 - When the box loses focus, the entire contents is checked to make sure it is a valid float. If not, the widget is highlighted
 - Optionally a function can be supplied to execute on a valid float

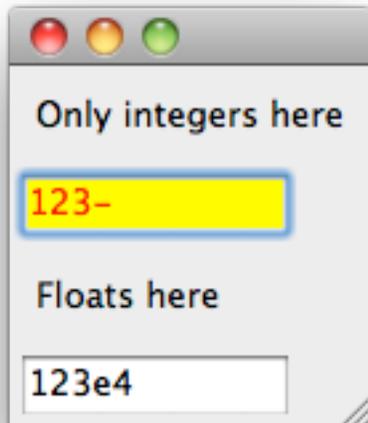
The validator is used very simply, by adding a parameter to the TextCtrl



Validator example

```
import wx
import validators
class mycalc:
    def __init__(self):
        self.myvar = None
    def SetVar(self, val, obj):
        self.myvar = val
```

```
app = wx.PySimpleApp() # create App
CalcObj = mycalc()
frm = mywxframe(CalcObj)
app.MainLoop()
print 'saved val = ', CalcObj.myvar
```



```
class mywxframe(wx.Frame):
    def __init__(self, DataObj):
        wx.Frame.__init__(self, None)
        self.DataObj = DataObj
        pnl = wx.Panel(self)
        sizer = wx.BoxSizer(wx.VERTICAL)
        pnl.SetSizer(sizer)

        w = wx.StaticText(pnl, -1,
                           "Only integers here")
        sizer.Add(w, 0, wx.ALL, 10)
        tc = wx.TextCtrl(
            pnl, -1, size=(100, -1),
            validator=validators.IntValidator(
                positiveonly=False))
        sizer.Add(tc, 0, wx.ALL, 5)

        w = wx.StaticText(pnl, -1, "Floats here")
        sizer.Add(w, 0, wx.ALL, 10)
        tc = wx.TextCtrl(
            pnl, -1, size=(100, -1),
            validator=validators.FloatValidator(
                DataObj.SetVar))
        sizer.Add(tc, 0, wx.ALL, 5)

        sizer.Fit(self)
        self.Show()
                                                                 wx11.py
```

Modal dialogs

Modal dialogs are used to display information to the user (such as an error message) or get input from the user via a new window; the program does not allow anything else to occur until the user has closed the new window.

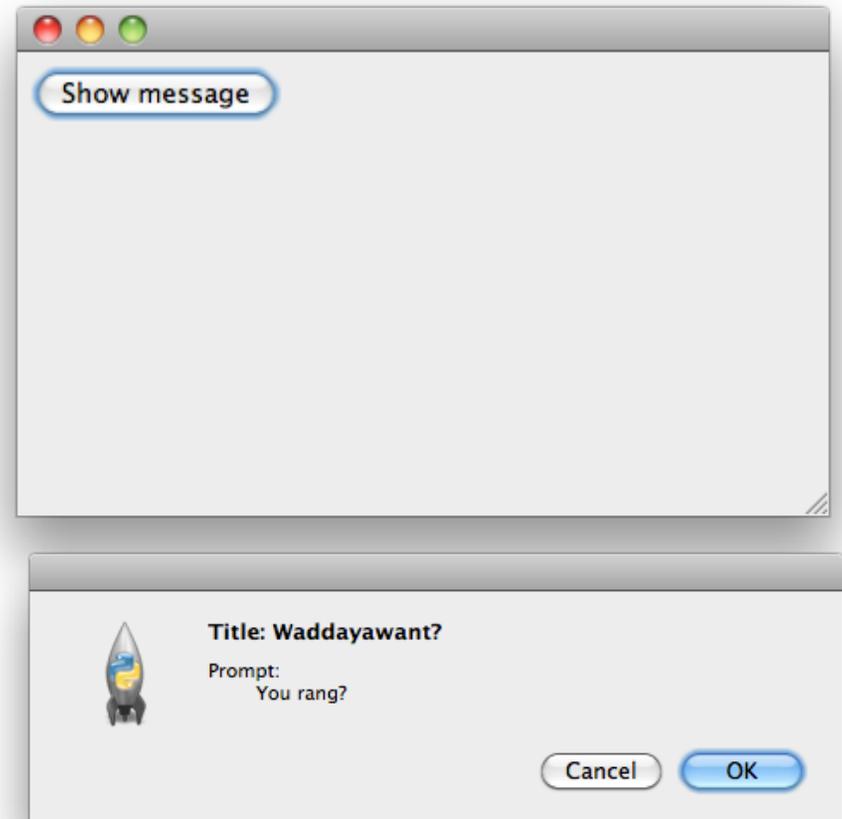
- The easiest way to implement a modal dialog is to use one of the many “stock” widgets in wxPython, such as `DirDialog`, `FileDialog`, `MessageDialog`, `TextEntryDialog` (~15 choices in total)



MessageDialog Example

```
class mywxframe(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None)
        pnl = wx.Panel(self)
        sizer = wx.BoxSizer(wx.VERTICAL)
        pnl.SetSizer(sizer)
        btn = wx.Button(pnl, -1,
                        "Show message")
        btn.Bind(wx.EVT_BUTTON,
                self.OnClick)
        sizer.Add(btn, 0, wx.ALL, 10)
        self.Show()

    def OnClick(self, event):
        dlg = wx.MessageDialog(self,
                               "Prompt:\n\tYou rang?",
                               'Title: Waddayawant?',
                               wx.OK | wx.CANCEL |
                               wx.ICON_QUESTION)
        if dlg.ShowModal() == wx.ID_OK:
            print "OK!"
        dlg.Destroy()
wx12.py
```



The dialog is created in response to the button. The main window is inactive when the dialog is open.

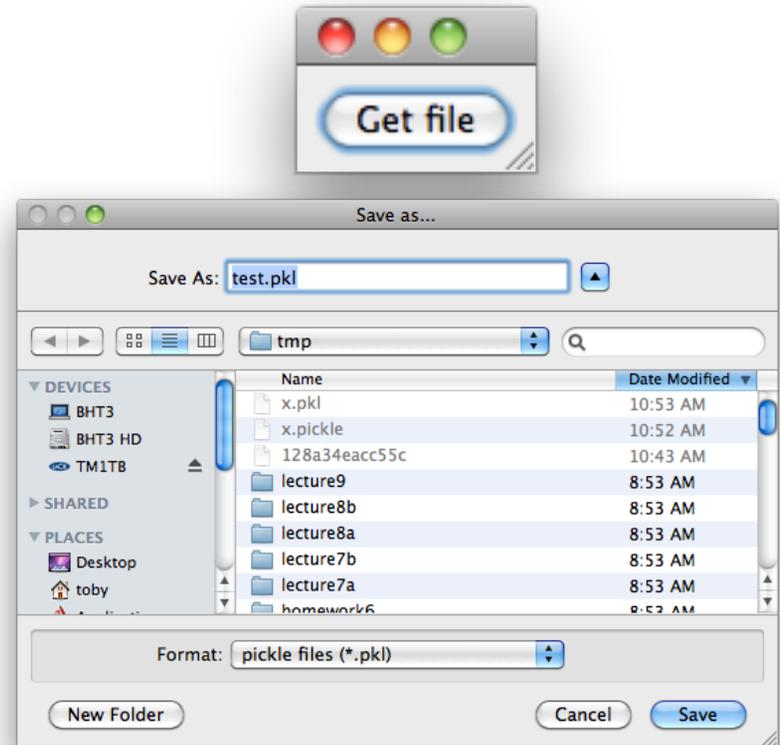


FileDialog Example

```
class mywxframe(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None)
        pnl = wx.Panel(self)
        sizer = wx.BoxSizer(wx.VERTICAL)
        pnl.SetSizer(sizer)
        btn = wx.Button(pnl, -1,
                        "Get file")
        btn.Bind(wx.EVT_BUTTON, self.OnClick)
        sizer.Add(btn,0,wx.ALL,10)
        sizer.Fit(self)
        self.Show()

    def OnClick(self, event):
        wildcard = "pickle files (*.pkl)|*.pkl"
        dlg = wx.FileDialog(
            self, message="Save as...",
            defaultDir="/tmp",
            defaultFile="test.pkl",
            wildcard=wildcard,
            #style=wx.FD_OPEN
            style=wx.FD_SAVE)
        if dlg.ShowModal() == wx.ID_OK:
            print 'path = ',dlg.GetPath()
        else:
            print 'Cancelled!'
```

wx12a.py



The dialog is created in response to the button. The main window is inactive when the dialog is open. The dialog uses the native file browser for the OS/WM of computer

For complex modal dialogs, subclass wx.Dialog

```
class TestDialog(wx.Dialog):
    def __init__(self, parent, ID):
        # Instead of calling wx.Dialog.__init__ we use this:
        pre = wx.PreDialog()
        pre.Create(parent, ID, "Override the defaults", wx.DefaultPosition,
                  wx.DefaultSize, wx.DEFAULT_DIALOG_STYLE)
        self.PostCreate(pre)
        sizer = wx.BoxSizer(wx.VERTICAL)
        label = wx.StaticText(self, -1, "Variables used in data collection")
        sizer.Add(label, 0, wx.ALIGN_CENTRE|wx.ALL, 5)
        line = wx.StaticLine(self, -1, size=(20,-1), style=wx.LI_HORIZONTAL)
        sizer.Add(line, 0, wx.GROW|wx.ALIGN_CENTER_VERTICAL|wx.RIGHT|wx.TOP, 5)
        label = wx.StaticText(self, -1, "(useful stuff goes here)")
        sizer.Add(label, 0, wx.ALIGN_CENTRE|wx.ALL, 5)
        btnsizer = wx.StdDialogButtonSizer()
        btn = wx.Button(self, wx.ID_OK)
        btn.SetDefault()
        btnsizer.AddButton(btn)
        btn = wx.Button(self, wx.ID_CANCEL)
        btnsizer.AddButton(btn)
        btnsizer.Realize()
        sizer.Add(btnsizer, 0, wx.ALIGN_CENTER_VERTICAL|wx.ALL, 5)
        self.SetSizer(sizer)
        sizer.Fit(self)
        pre.CenterOnParent()
```

wx14.py

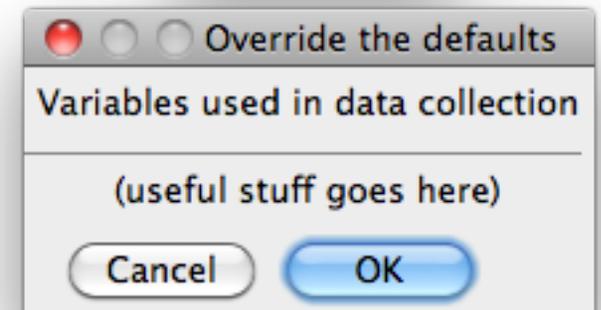
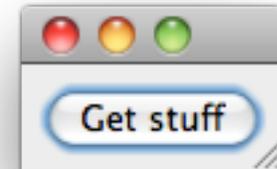


Using the subclassed wx.Dialog (pretty much like any other Dialog)

```
import wx
class mywxframe(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None)
        pnl = wx.Panel(self)
        sizer = wx.BoxSizer(wx.VERTICAL)
        pnl.SetSizer(sizer)
        btn = wx.Button(pnl, -1,
                        "Get stuff")
        btn.Bind(wx.EVT_BUTTON, self.OnClick)
        sizer.Add(btn,0,wx.ALL,10)
        sizer.Fit(self)
        self.Show()

    def OnClick(self, event):
        dlg = TestDialog(self, -1)
        # returns after window is closed
        val = dlg.ShowModal()
        if val != wx.ID_OK:
            dlg.Destroy() # cancel was pressed
            return
        # do something with dlg contents here
        dlg.Destroy() # close it
        return
```

wx14.py



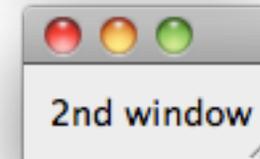
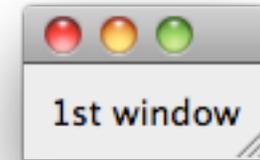
Non-modal windows

```
import wx
class mywxframe(wx.Frame):
    def __init__(self, parent, lbl):
        wx.Frame.__init__(self, parent)
        pnl = wx.Panel(self)
        sizer = wx.BoxSizer(wx.VERTICAL)
        pnl.SetSizer(sizer)
        w = wx.StaticText(pnl, -1, lbl)
        sizer.Add(w, 0, wx.ALL, 10)
        sizer.Fit(self)
        self.Show()

app = wx.PySimpleApp() # create the App
frm1 = mywxframe(None, '1st window')
frm2 = mywxframe(None, '2nd window')
frm3 = mywxframe(None, '3rd window')
app.MainLoop()
print 'event loop done'                                     wx13.py
```

Frames can be “top level” – with no parents. Each window has no parent and thus is independent.

- The Event loop runs until all are closed



Child windows

```
import wx
class mywxframe(wx.Frame):
    def __init__(self, parent, lbl):
        wx.Frame.__init__(self, parent)
        pnl = wx.Panel(self)
        sizer = wx.BoxSizer(wx.VERTICAL)
        pnl.SetSizer(sizer)
        w = wx.StaticText(pnl, -1, lbl)
        sizer.Add(w, 0, wx.ALL, 10)
        sizer.Fit(self)
        self.Show()

app = wx.PySimpleApp() # create the App
frm1 = mywxframe(None, '1st window')
frm2 = mywxframe(frm1, '2nd window\nchild of 1st')
frm3 = mywxframe(frm2, '3rd window\nchild of 2nd')
app.MainLoop()
print 'event loop done'
```

wx13a.py

Here each new frame is a child of the former. Only the first is “top level” (no parent).

- Closing the 2nd window kills the 3rd.
- Closing the 1st window kills all
- The Event loop runs until all are closed which is now tied to the 1st window.



Homework

- Create a small wxPython program to do something useful. Let me know what you did or where you get stuck.

