humblewx Documentation

Release 0.0.0

Rickard Lindberg

July 02, 2016

Contents

Contents:

CHAPTER 1

Introduction

`humblewx` is a Python library that makes it more pleasant to create user interfaces with wxPython. It has two goals:

- Simplify writing code to construct and layout wxPython GUI components
- Enforce the Humbe Dialog Box pattern

`humblewx` started life as a module in The Timeline Project. It was created because we wanted a more pleasant way to work with all wxPython dialogs. It was later extracted to its own library.

CHAPTER 2

Downloading & Installing

`humblewx` is a regular Python package that can be downloaded and installed with `pip`.

From PyPi:

```
pip install humblewx
```

From source (recommended because PyPi is not always up to date):

```
pip install git+https://github.com/thetimelineproj/humblewx
```

After the install, you should be able to import `humblewx` like this:

```
import humblewx
```

CHAPTER 3

Tutorial

This tutorial demonstrates basic usage of `humblewx` by showing and explaining a few examples.

3.1 Hello World

Our first example is a dialog that displays the text hello world. It looks something like this:

Here is the complete code:

```
import humblewx
import wx


class HelloWorldDialog(humblewx.Dialog):

    """
    <StaticText label="Hello World" />
    """

    def __init__(self, parent):
        humblewx.Dialog.__init__(self, HelloWorldDialogController, parent)
```

```
class HelloWorldDialogController(humblewx.Controller):
    pass


if __name__ == "__main__":
    app = wx.App()
    dialog = HelloWorldDialog(None)
    dialog.ShowModal()
    dialog.Destroy()
```

Let's walk through the code piece by piece:

```
import humblewx
import wx
```

Here we import the `hubmlewx` module that is used to access its functionality. The `wx` import is only needed in the example program to create the `App` object so that we can display our dialog.

```
class HelloWorldDialog(humblewx.Dialog):
```

Here we say that we want to create a dialog. The `humblewx.Dialog` is actually a subclass of `wx.Dialog`. This subclass adds the functionality to create the GUI from a description in XML.

```
    """
    <StaticText label="Hello World" />
    """
```

This is the description of the GUI. It is written in the dosctring of the class. It is written in XML and describes what components should be in our dialog. In this case we have only one component.

```
    def __init__(self, parent):
        humblewx.Dialog.__init__(self, HelloWorldDialogController, parent)
```

Here we create the dialog. The *`__init__()`* method will read the GUI description and construct the components. The first argument, `HelloWorldDialogController` is a class that will be instantiated and used as a controller for this dialog. We'll come back to what the controller does. For now we just need to know that it must be a subclass of *`humblewx.Controller()`*.

```
class HelloWorldDialogController(humblewx.Controller):
    pass
```

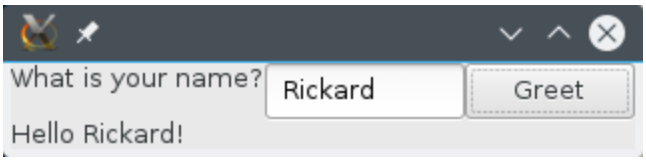
Here we define our controller. At the moment it doesn't do anything.

```
if __name__ == "__main__":
    app = wx.App()
    dialog = HelloWorldDialog(None)
    dialog.ShowModal()
    dialog.Destroy()
```

This code displays our dialog when we run the Python file.

3.2 Greeting

Our second example is a greeting dialog that allows us to enter our name, and when we press a button a greeting will be shown. It looks something like this:



Here is the dialog class:

```
class GreetingDialog(humblewx.Dialog):

    """
    <BoxSizerVertical>
        <BoxSizerHorizontal>
            <StaticText label="What is your name?" />
            <TextCtrl name="name_text_ctrl" />
            <Button label="Greet" event_EVT_BUTTON="on_greet_clicked" />
        </BoxSizerHorizontal>
        <StaticText name="greeting" label="" />
    </BoxSizerVertical>
    """

    def __init__(self, parent):
        humblewx.Dialog.__init__(self, GreetingDialogController, parent)

    def GetName(self):
        return self.name_text_ctrl.GetValue()

    def SetGreeting(self, text):
        self.greeting.SetLabel(text)
```

Here is the controller class:

```
class GreetingDialogController(humblewx.Controller):

    def on_greet_clicked(self, event):
        self.view.SetGreeting("Hello %s!" % self.view.GetName())
```

We can see that the GUI description has been extended from the previous example. We have more components and we use sizers to control how they are laid out.

The second interesting addition in this example is that we have communication between the dialog and the controller. They collaborate in a pattern inspired by the Humbe Dialog Box. The dialog corresponds to the view and the controller corresponds to the smart object.

The controller receives events from the view (such as a button click) and responds to them by calling methods on the view.

The way to connect events to the controller is via `event_` attributes in the XML. When `humblewx` sees `event_EVT_BUTTON="on_greet_clicked"`, it will automatically bind the `EVT_BUTTON` event to the `on_greet_clicked` method on the controller.

What happens when we click the greet button?

- `on_greet_clicked` is called.
- It calls `GetName` on the view.
- `GetName` in turn gets the name from the text control. The view can access the text control by the name `name_text_ctrl` because we specified the name attribute in the XML.
- `on_greet_clicked` then calls `SetGreeting` on the view with the greeting string constructed from the name.

- `SetGreeting` sets the label on the static text similarly to how `GetName` got the text from the text control.

3.3 Summary

In our experience it's very pleasant to be able to describe how the GUI should look like in XML instead of manually calling `wx` APIs. We feel that we can more rapidly create new dialogs that also look better. Changing existing ones is also more pleasant.

The separation between the view and the control makes the code even cleaner and the controller can be tested in isolation without ever invoking a GUI.

We encourage you to try this approach to creating user interfaces in wxPython. Let us know what you think.

CHAPTER 4

Topics

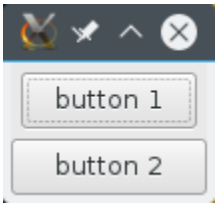
The following topics explain how to use a certain aspect of the library.

4.1 Using Sizers

Sizers is the technique used in wxPython to control the layout of components. However, using sizers directly requires writing code that is difficult to understand. Here is a simple example:

```
class SizersWxExampleDialog(wx.Dialog):

    def __init__(self, parent):
        wx.Dialog.__init__(self, parent)
        button1 = wx.Button(self, label="button 1")
        button2 = wx.Button(self, label="button 2")
        sizer = wx.BoxSizer(wx.VERTICAL)
        sizer.Add(button1, flag=wx.EXPAND|wx.ALL, border=5)
        sizer.Add(button2, flag=wx.EXPAND)
        self.SetSizerAndFit(sizer)
```

We have two buttons that are laid out vertically. It looks like this:



The problem is that this is not obvious to figure out by taking a quick look at the code. That is because the structure of the components is not reflected in the structure of the code. This problem grows larger the more components we have in our dialogs.

`humblewx` allow us to define everything about a component in one place. The hierarchical structure of XML also makes it easier to see how the components are laid out. Let's see what the above example looks like rewritten using `humblewx`:

```
class SizersHumbleWxExampleDialog(humblewx.Dialog):

    """
    <BoxSizerVertical>
        <Button label="button 1" border="ALL" />
        <Button label="button 2" />
```

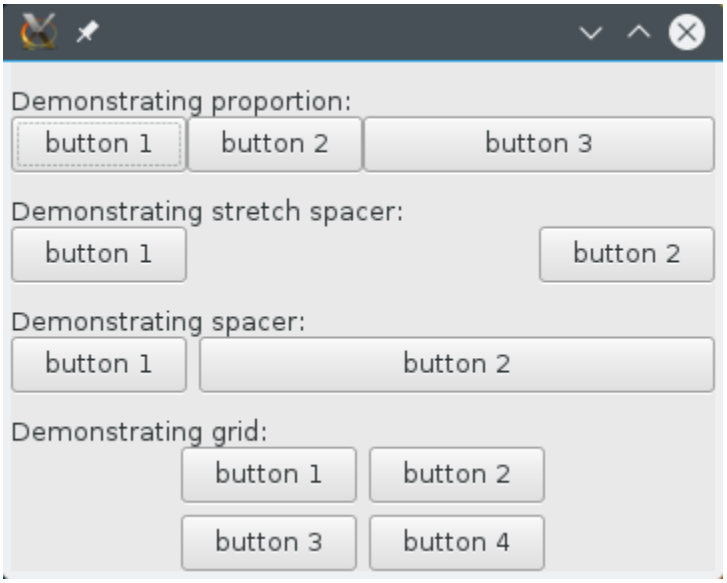
```
    </BoxSizerVertical>
    """

    def __init__(self, parent):
        humblewx.Dialog.__init__(self, humblewx.Controller, parent)
```

Quickly we can see that this dialog has two buttons and that they are laid out vertically.

Here is a larger example demonstrating what we can do with sizers.

```
class SizersFullExampleDialog(humblewx.Dialog):

    """
    <BoxSizerVertical>

        <StaticText border="TOP" label="Demonstrating proportion:" />
        <BoxSizerHorizontal>
            <Button label="button 1" proportion="1" />
            <Button label="button 2" proportion="1" />
            <Button label="button 3" proportion="2" />
        </BoxSizerHorizontal>

        <StaticText border="TOP" label="Demonstrating stretch spacer:" />
        <BoxSizerHorizontal>
            <Button label="button 1" />
            <StretchSpacer />
            <Button label="button 2" />
        </BoxSizerHorizontal>

        <StaticText border="TOP" label="Demonstrating spacer:" />
        <BoxSizerHorizontal>
            <Button label="button 1" />
            <Spacer />
            <Button label="button 2" proportion="1" />
        </BoxSizerHorizontal>

        <StaticText border="TOP" label="Demonstrating grid:" />
        <FlexGridSizer columns="2" align="ALIGN_CENTER">
            <Button label="button 1" />
            <Button label="button 2" />
            <Button label="button 3" />
            <Button label="button 4" />
        </FlexGridSizer>

    </BoxSizerVertical>
    """

    def __init__(self, parent):
        humblewx.Dialog.__init__(self, humblewx.Controller, parent)
```

The dialog looks like this:



4.2 Using Variables

```
class VariablesExampleDialog(humblewx.Dialog):

    """
    <BoxSizerVertical>
        <StaticText
            label="$(translated_label)"
        />
    </BoxSizerVertical>
    """

    def __init__(self, parent):
        humblewx.Dialog.__init__(self, humblewx.Controller, parent, {
            "translated_label": "Gutent tag",
        })
```

4.3 Using Custom Components

We have seen that we can refer to standard `wx` components such as buttons and static text fields from the XML. What about custom components that are not inside the `wx` namespace?

`humblewx` can be configured to look for components anywhere. We just need to modify the *`humblewx.COMPONENT_MODULES`* configuration variable.

Say we have this custom component that we want to use in a dialog:

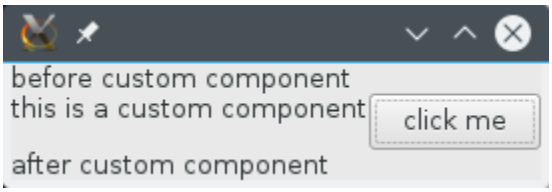
```
class CustomComponent(wx.Panel):

    def __init__(self, parent):
        wx.Panel.__init__(self, parent)
        label = wx.StaticText(self, label="this is a custom component")
        button = wx.Button(self, label="click me")
        sizer = wx.BoxSizer(wx.HORIZONTAL)
```

```
        sizer.Add(label)
        sizer.Add(button)
        self.SetSizer(sizer)
```

In the XML we can just refer to this component by name as we do with any other `wx` component:

```
class CustomComponentExampleDialog(humblewx.Dialog):

    """
    <BoxSizerVertical>
        <StaticText label="before custom component" />
        <CustomComponent />
        <StaticText label="after custom component" />
    </BoxSizerVertical>
    """

    def __init__(self, parent):
        humblewx.Dialog.__init__(self, humblewx.Controller, parent)
```

It looks like this:



In order for this to work, we have to tell `humblewx` that it should also look for components in another module. In this example, we only have one module where both the custom component and the dialog are defined. We can get a reference to that module with the following code:

```
sys.modules[__name__]
```

Next we need to modify *humblewx.COMPONENT_MODULES* to include this module:

```
humblewx.COMPONENT_MODULES.append(sys.modules[__name__])
```

We need to run this code before we create our dialog.

We can add any module to this list:

```
import foo.bar
humblewx.COMPONENT_MODULES.append(foo.bar)
```

CHAPTER 5

API

5.1 Classes

class humblewx.**Dialog**

__init__(*controller_class*, *parent*, *variables={}*, ***kwargs*)

This constructs a wx.Dialog and fills it with content according to the *GUI description* found in this class' docstring.

Parameters

- **controller_class** – The class that should be used as a controller in this dialog. It will be initialized automatically.
- **parent** – The parent window to this dialog. Passed as first argument to the __init__ method of the wx.Dialog.
- **variables** – Variables that can be accessed by name from the XML definition. Should be a mapping from strings to Python values.
- ****kwargs** – Additional parameters that are passed to the __init__ method of the wx.Dialog.

class humblewx.**Controller**

__init__(*view*)

5.2 Configuration

humblewx.**COMPONENT_MODULES**

Default [wx]

This is a list of modules where humblewx will search for components.

By default, only wx components can be found. Extend or change this list to allow humblewx to find components defined in other modules.

5.3 GUI Description Language

GUI descriptions are defined in XML.

Nodes in the XML correspond to either components or sizers. Attributes correspond to arguments passed to the constructors. For example:

```
<Button label="Hello World" />
```

Will result in the following Python code:

```
wx.Button(..., label="Hello World")
```

5.3.1 Attribute values

Often components need arguments that are not strings. Attribute values in the XML are interpreted in the following order:

Variable
Example:

```
<Button label="$(name)" />
```

If the attribute value matches the variable pattern `$(..)`, the Python value will be fetched from the variables dictionary passed to *Dialog*.

Boolean
Example:

```
<Button label="True" />
<Button label="False" />
```

If the attribute value matches eihter `True` or `False`, the Python value will be the corresponding boolean.

String
Example:

```
<Button label="Hello World" />
```

All other attribute values will be returned as Python strings.

5.3.2 Special nodes

BoxSizerVertical
This is the quivalent of the following Python code:

```
wx.BoxSizer(wx.VERTICAL)
```

BoxSizerHorizontal
This is the quivalent of the following Python code:

```
wx.BoxSizer(wx.HORIZONTAL)
```

FlexGridSizer
This creates a `wx.FlexGridSizer`. It supports the following attributes:

rows

Default `0`

The number of rows this sizer should have.

columns

Default `0`

The number of columns this sizer should have.

growableColumns
A comma separated list of integers saying which columns should be growable. (Argument to `AddGrowableCol`.)

growableRows
A comma separated list of integers saying which row should be growable. (Argument to `AddGrowableRow`.)

StaticBoxSizerVertical
This creates a static box and a corresponding sizer used to lay out child components. All attributes are passed to the `wx.StaticBox`. The sizer is created like this:

```
wx.StaticBoxSizer(..., wx.VERTICAL)
```

Spacer
This can only be used within a sizer.

StretchSpacer
This can only be used within a sizer.

5.3.3 Special attributes

name

event_*

border

borderType

proportion

align

CHAPTER 6

Release

Steps to make a release:

1. Update version number in setup.py
2. Build with `python setup.py sdist`
3. Upload to PyPi with `twine upload dist/humblewx-x.y.z.tar.gz`
4. Tag `git tag x.y.z`

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

h

Symbols

C

D

H