BoxSizerFromTheGroundUp

BoxSizerFromTheGroundUp/ControlsPadding

BoxSizerFromTheGroundUp/ControlsResizing

BoxSizerFromTheGroundUp/NestedBoxSizers

BoxSizerFromTheGroundUp/DivideAndConquer

The Border Spacing Flags

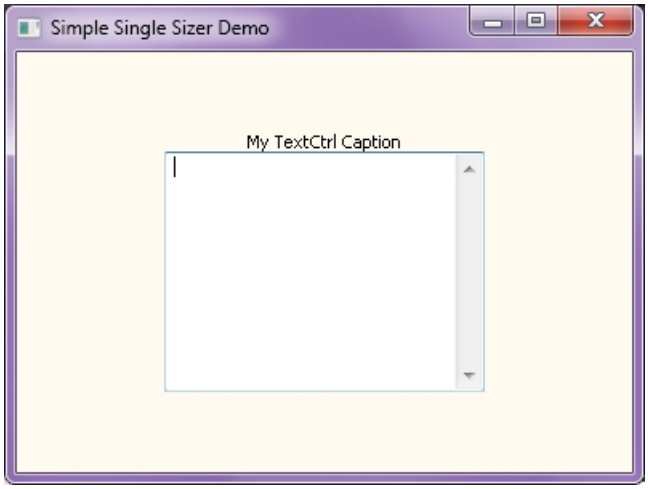
We've seen the **Alignment** flags, so now we'll look into the Border Spacing flags. These flags ~~have almost nothing to do with the Alignment flags, yet they~~ must be logically "**OR**"ed into the value of the **flag=** argument of the sizer.Add() method along with any Alignment flags that are needed. These flags are used to specify which sides of the control ~~that~~ are to be bordered (padded) with a space the size of the value of the **border=** argument. Any combination of these flags may be used in the **flag=** argument, but they will all be assigned the single **border=** value of space (in pixel units).

Border Spacing Flag Constants:

- **wx.TOP**
- **wx.BOTTOM**
- **wx.LEFT**
- **wx.RIGHT**
- **wx.ALL**

The **wx.ALL** flag is a convenience definition that specifies that all four sides of the control are to be bordered.

Here's the current state of the sample program:

- 

I want to put a small separation between the StaticText and the TextCtrl. We've seen how to put in a square space with the .AddSpacer() method and to .Add() a rectangular space. Here's yet another way using the flag= and border= parameters. We could either border the top of the TextCtrl or the bottom of the StaticText or even border both. ~~I hope you now see why just adding a spacer in between the controls is simpler.~~ For now, the top of the TextCtrl will be padded. All that's needed to be done is to modify the .Add() statement for the TextCtrl to put in, say, 7 pixels of border padding.

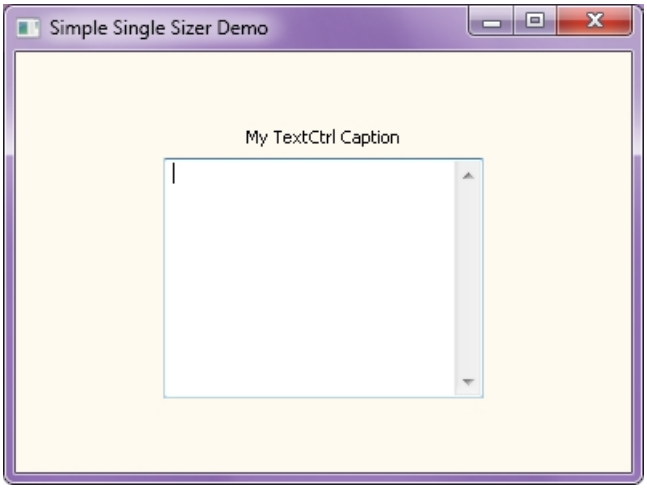
It was :

```
        allCtrls_vertSizer.Add( self.myListing_txtCtrl,
flag=wx.ALIGN_CENTER )
```

Change it to:

```
        allCtrls_vertSizer.Add( self.myListing_txtCtrl,
flag=wx.ALIGN_CENTER | wx.TOP, border=7 )
```

~~Note that the effect of **wx.ALIGN_CENTER** has absolutely nothing at all to do with the effect of **wx.TOP** other than they are both applied to the same control.~~ We now get:

- 

The appearance is now a little nicer and it was easy to do. But, it is more direct and clearer to simply insert the line:

```
allCtrls_vertSizer.Add( (0, 7) )            # A one-dimensional
vertical spacer. The best option, overall.
```

or even:

```
allCtrls_vertSizer.AddSpacer( 7 )       # A square spacer. It's Its
minor axis dimension wont't interfere in this case.
```

When it comes to spacers, "there's more than one way to skin a cat". However, another person who is tries to figure out your code using the wx.TOP argument has to make the effort to realize that the **wx.ALIGN_CENTER** parameter has nothing to do with the **wx.TOP** parameter even though they are lumped together in the same **flag=** argument.

The border padding flags can get very confusing to use and modify, especially if adjacent controls both use them. It's far simpler and easier to understand by just inserting a spacers using separate statements. Doing this completely isolates the effect of the space and applies it to exactly where it is needed without any unintentional interactions and side effects. In this regard it is better to use either of the .Add((w, 0)) or .Add((0, h)) spacer insertion statements rather than the .AddSpacer(n) statement. A square spacer's minor axis dimension cannot be reduced to 0 and could unintentionally collide with other controls in an adjacent sizer. Using either of the one-dimensional spacers guarantees this will not happen. But, the dimension value must be placed in the proper ordinate position of the given (x, y) coordinate.

** The claim that AddSpacer or Add(tuple) is the cure-all for minor adjustments only holds for adjustments in the major orientation. Please clarify this point. **

Having not to even to think about which is the major and minor axes of a BoxSizer can be totally avoided by using this function:

```
 1 def AddLinearSpacer( boxsizer, pixelSpacing ) :
 2     """ A one-dimensional spacer for use with any BoxSizer """
 3
 4     orientation = boxsizer.GetOrientation()
 5     if   (orientation == wx.HORIZONTAL) :
 6         boxsizer.Add( (pixelSpacing, 0) )
 7
 8     elif (orientation == wx.VERTICAL) :
 9         boxsizer.Add( (0, pixelSpacing) )
10     #end if
11
12 #end def
```

~~This algorithm is so simple that it's incomprehensible to me why this hasn't already been built into the wx.~~BoxSizer~~ as a method. I guess this is just another example of the ~~BoxSizer~~'s "Design by Committee". Using this function is easier and foolproof, axes-wise. E.g.:~~

```
~~ 1 AddLinearSpacer( allCtrls_vertSizer, 7 )          # Look, Ma !  It figures out the right orientation all by itself.~~
```

AddLinearSpacer.py

That's all there is to spacing controls from each other and from the edges of the page. We'll now see how the BoxSizer can automatically resize a control when the page is resized.

Next Page: BoxSizerFromTheGroundUp/ControlsResizing

BoxSizerFromTheGroundUp

BoxSizerFromTheGroundUp/ControlsPadding

BoxSizerFromTheGroundUp/ControlsResizing

BoxSizerFromTheGroundUp/NestedBoxSizers

BoxSizerFromTheGroundUp/DivideAndConquer

BoxSizerFromTheGroundUp/ControlsPadding (last edited 2010-09-29 16:38:31 by WinCrazy)