
The wx.BoxSizer

Learning the necessary details to effectively use the wx.BoxSizer has caused excessive and needless frustration for many people, including myself. This is solely due to the lack of complete documentation. There are countless "tutorials" [sic] and demo apps, but none fully document *and* fully explain all the various argument flags. This is exacerbated by the ~~actual design~~ implementation of the sizer which requires logically **OR**ing together unrelated **flag=** constants that have very unrelated effects. I can only guess that the BoxSizer must have been "**designed by committee**" rather than in the manner that the vast majority of the huge number of functions and classes that make up the core of **wxPython**. ** I think it's best not to make critical remarks. **

Sizers are **wx.Control** auto-positioning ** geometry management is what they are. that term is common in tcl/tk ** algorithms. The BoxSizer is the ~~most~~ a basic sizer and probably the most used. It is not difficult to use once the meanings of all its flags are understood. Understanding the BoxSizer will go a long way toward knowing how to use the other more complex sizers. This sizer is deceptively intricate, whereas the seeming more complex sizers are actually more straightforward to use (although, more difficult to maintain).

** This next bit about 'stack' I don't agree with. I find pack is much better as stack makes me think things are being placed on top of each other as opposed to being side by side all visible, of course, I have some experience with tcl/tk's pack geometry manager which is very similar to the boxsizer. I think you should simply say the added widgets are 'sequenced' or 'packed' along the major axis per the flags. Lose the opinions and editorializing or you'll scare new people off. **

As ubiquitous as the BoxSizer may be, it is poorly named. A much better one would be "StackSizer". The BoxSizer does actually stack (sequence along one axis) wx controls in either the vertical or horizontal direction, from top-to-bottom or left to-right, respectively. When instantiating a BoxSizer, ~~it's~~ its "major" axis must be declared to be either the **wx.VERTICAL** or **wx.HORIZONTAL**. This is the

axis on which controls will be stacked. A seemingly magical trait of all sizers, in general, is that they auto-arrange their controls whenever the page is resized. Instantiating a BoxSizer takes the general form:

Note: Square brackets, **[ ]**, in the code boxes indicate optional arguments; curly brackets, **{ }**, indicate the associated argument values. ** needs to be before first use **

```
{ bxSzrName } = wx.BoxSizer( wx.VERTICAL orientation )        # There are
no other parameters. Especially not a parent !
```

Controls of any type can be positioned by the sizer using the **.Add()** method so that their absolute positions can be automatically controlled. The general syntax to **Add()** a control is:

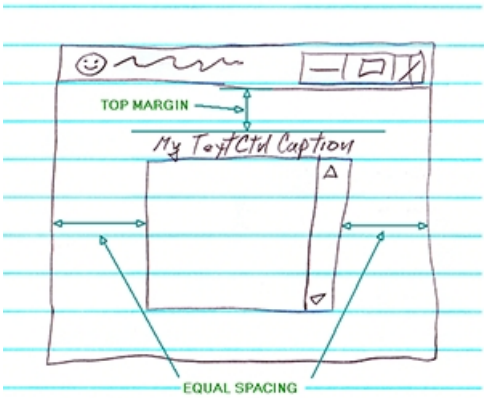
```
{bxSzrName}.Add( { a control name, a sizer spacer or any other sizer
                   a widget name, a sizer name, a wx.Size object or
(width, height) tuple },
                              [ proportion=]{ a positive integer },
                              [ flag=]{ a logical "OR" phrase of border
padding and minor axis positioning flags },
                              [ border=]{ a positive integer pixel quantity
of spacing } )

** the wx doc shows only 'item' as the first arg and all the variations
were not easily discovered. So say it for all to understand. 'control'
doesn't tell me that a StaticText can go there. 'sizer spacer', what's
that? And I think 'sizer name' is actually a sizeritem which the docs are
unclear on.**
```

The square brackets, **[ ]**, in the pseudocode snippet above indicate optional arguments. The curly brackets, **{ }**, indicate the associated argument values. Even though the last three argument parameters are optional, it's likely that an optional parameter is needed for some controls to result in a reasonably acceptable appearance.

Here's the first task - lay out a **Frame** so it looks like this:

- . .



This **wx.Frame** will consist of just a **wx.StaticText** and a **wx.TextCtrl**. The most important positioning requirements are:

- The **StaticText** is used as the caption centered immediately over top the **TextCtrl**.
- Both controls are centered horizontally within the "page" (i.e., the Frame's ***client area***).
- Both controls have equal spacing to the edges of the client area for both axes.

My term ***page*** is used here to describe any kind of **wx** container control such as a **wx.Frame**, a **wx.Window**, a **wx.Panel**, a **wx.Dialog**, a notebook tab, ~~etc.~~, etc. Any **wx** control that can hold another control is a container by definition and can have a sizer position its child controls. In general, if a sizer is used within a container control, then all the child controls should be positioned by the sizer. This is not a requirement, but if a child control is not positioned by the sizer it may end up be a "rogue" control that misbehaves when the page is resized either by the user or programmatically.

Nearly all **[Ed.: all ?]** controls are subclassed (derived from) the **wx.Window**, and so, they normally retain most of the **wx.Window** class methods and class variables. Consider that a **wx.Button** looks just like a **wx.Window** loaded with a **wx.Bitmap** which changes its bitmap when it is left-clicked. A **wx.Frame** looks just like a **wx.Window** with a toolbar at the top and with a border that is an active control (actually a composite of individual edge + corner controls) that allows the window to be resized by the user.

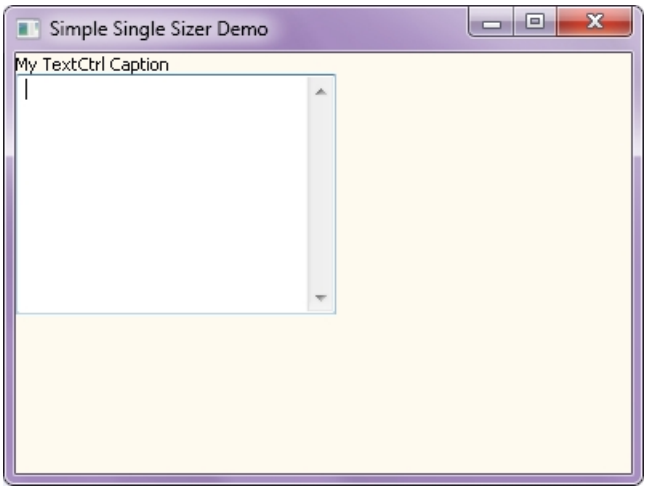
A **wx.Panel** is a **wx.Window** without any visible border and controls at all ! That's why sizers can be implemented inside any container and not just **wx.Frames** and **wx.Windows**. It might be said, "All in the world is a window". A **wx.Window** in the **wx** world, that is.

** The above 3 paragraphs are welcome but I think would be better placed at the top as 'background' or 'widget basics'. They're somewhat out of place here. If you already know this stuff they're in the way. If you don't already know it you need it earlier. **

The code to create the two controls (i.e., instantiate wxPython control objects), create a BoxSizer and then have that sizer locate their positions is as follows:

```
   1         # The caption wx.StaticText will be placed directly above a
wx.TextCtrl.
   2         # Note that no "pos=" positioning parameter is needed
because the BoxSizer
   3         # will determine its positioning.
   4         # Even if a position would be given it would be ignored and
overridden by the sizer.
   5         #
```

```
   6         caption_stTxt = wx.StaticText( self, -1, 'My TextCtrl
Caption' )
   7         self.listing_txtCtrl = wx.TextCtrl( self, -1,
style=wx.TE_MULTILINE, size=(200, 150) )
   8
   9         allCtrls_vertSizer = wx.BoxSizer( wx.VERTICAL )
  10
  11         allCtrls_vertSizer.Add( caption_stTxt,       proportion=0 )
  12         allCtrls_vertSizer.Add( self.listing_txtCtrl, proportion=0 )
  13
  14         self.SetSizer( allCtrls_vertSizer )
  15         self.Layout()                                              #
self.Fit()           # use one or the other
```

- 

The sizer has the control properly "stacked" in the **wx.VERTICAL** direction.The next most important rearrangement is to position the two controls in the horizontal center of the Frame client. To do this the **flag=** parameter will be used.

The flag= Argument

The **flag=** argument gives additional instructions for how the BoxSizer should place controls on the page both relative to the other controls in the sizer as well as in reference to the page client area's four edges. The predefined wx flag value constants affect three nearly independent groups of placement and control size properties.

Minor Axis Positioning flags

If a BoxSizer is declared **wx.VERTICAL** then one of the flags **wx.ALIGN_LEFT** or **wx.ALIGN_RIGHT** can be used to "push" the control in that direction of the BoxSizer's **minor axis**. Likewise, a **wx.HORIZONTAL** BoxSizer can be given the **wx.ALIGN_TOP** or **wx.ALIGN_BOTTOM** minor axis positioning flags. The **wx.ALIGN_CENTER** flag will simply center the control no matter what ~~what is~~ the major axis. Note that any combination of any major axis positioning flags will be

silently ignored because they have no meaning in relation to the major axis. ** This last sentence is unclear. I know what it means but it both says too much and not enough. **

wx.ALIGN_LEFT # BoxSizer major axis must be wx.VERTICAL or this will be ignored.

wx.ALIGN_RIGHT # For a wx.VERTICAL BoxSizer, only.

wx.ALIGN_TOP # For a wx.HORIZONTAL BoxSizer, only.

wx.ALIGN_BOTTOM # For a wx.HORIZONTAL BoxSizer, only.

wx.CENTER = wx.CENTRE = ALIGN_CENTER = wx.ALIGN_CENTRE # For both axis orientations.

wx.CENTER_VERTICAL ~~# No definition for this is available.~~ ** it's a tutorial, make something up! **

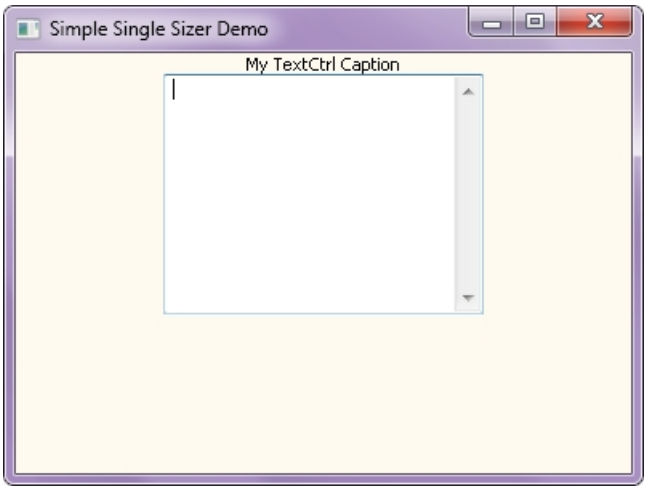
wx.CENTER_HORIZONTAL ~~# No definition for this is available.~~

The two sizer **.Add()** statements are modified to center the controls like this:

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

The complete demo program is here:

SIMPLE_SINGLE_SIZER_2_A.PY

- 

That's better.All that's left to be done is to add some space between the controls and the frame client's ~~to~~ edge. There are two ways to put spacing on an edge of a control:

1) Include a **flag=** argument such as **wx.Top** constant in the **.Add()** statement as well as the associated **border=** pixel integer value.

2) Directly use one of three kinds of BoxSizer "spacers" just before **.Add()**ing the StaticText.

Style 2) is much easier to read and use. Style 1) will be covered later in this tutorial.

a) sizerName.AddSpacer(~~{int}~~ wx.Size) This inserts a ~~square~~ space of {int} by {int} dimensions.

b) sizerName**.Add((w, h))** This inserts a rectangular space of (w, h) dimensions.

c) sizerName.**AddStretchSpacer([prop=]{int})**

This inserts a one-dimensional spacer that is extends only along the sizer's major axis. This spacer has the property of expanding to the limit of what room there is left along the sizer's major axis that is not already taken by fixed sized controls and spacers. If multiple **AddStretchSpacer()**s are present, they will divide up the available excess room left over from fixed sized controls and spacers equally in proportion to their prop= argument values. Example:

Suppose a vertical BoxSizer is created and the total vertical cliendimension (the "extent") is 100 pixels. There will be a single control with an **.AddStretchSpacer()** on either side of that control:

```
   1         self.SetClientSize( (123, 100) )
   2
   3         allCtrls_vertSizer = wx.BoxSizer( wx.VERTICAL )
   4
   5         allCtrls_vertSizer.AddStretchSpacer( prop=3 )
   6
   7         allCtrls_vertSizer.Add( caption_stTxt,
flag=wx.ALIGN_CENTER )
   8         allCtrls_vertSizer.Add( self.listing_txtCtrl,
flag=wx.ALIGN_CENTER )
   9
  10         allCtrls_vertSizer.AddStretchSpacer( prop=7 )
  11
  12         self.SetSizer( allCtrls_vertSizer )
  13         self.Layout()                          # self.Fit()  # always
use one or the other ** Why? Maybe you don't want to get into this here
so leave the comment out **
```

Suppose the TextCtrl's height is 70 pixels. That leaves 30 pixels, total, of "left over" client room on the vertical axis to be allocated to the two Stretch spacers. Since the proportion argument values are 3 and 7, the 30 pixels of left over space will be allocated to the spacers in proportions of 3 : 7, respectively.

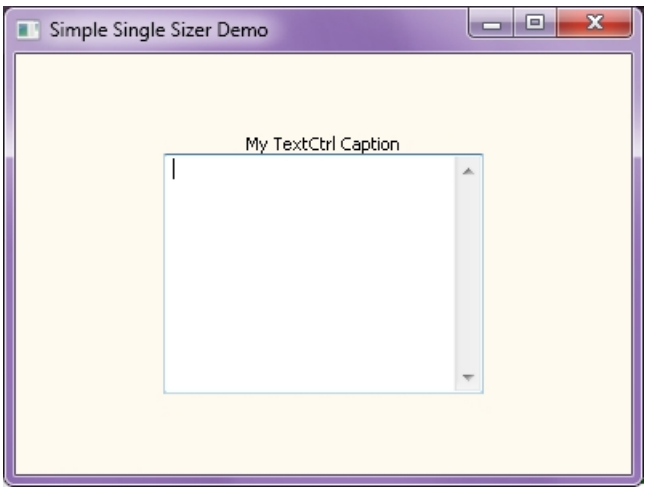
Leading Stretch spacer size = (3 / (3 + 7)) * 30 pixels = 9 pixels

Trailing Stretch spacer size = (7 / (3 + 7)) * 30 pixels = 21 pixels

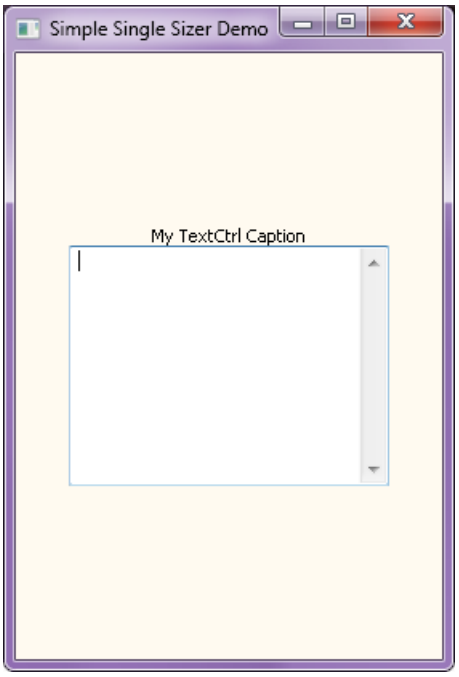
This example's proportion argument values are rather bizarre and serve only to help clarify how a BoxSizer calculates each Stretch spacer's size. The Stretch spacer is much more often used to equally pad both ends of a control or group of controls so that the spacing between the edges of the container and the ends of the control(s) are equal. This is exactly what is needed for our "simple sizer" demo program:

** No picture for this one? **

```
   1         allCtrls_vertSizer = wx.BoxSizer( wx.VERTICAL )
   2
   3         allCtrls_vertSizer.AddStretchSpacer( prop=1 )
   4
   5         allCtrls_vertSizer.Add( caption_stTxt,
flag=wx.ALIGN_CENTER )
   6         allCtrls_vertSizer.Add( self.listing_txtCtrl,
flag=wx.ALIGN_CENTER )
   7
   8         allCtrls_vertSizer.AddStretchSpacer( prop=1 )
   9
  10         self.SetSizer( allCtrls_vertSizer )
  11         self.Layout()                          # self.Fit()  # always
use one or the other
```

- 

The great feature of sizers is that they automatically adjust the placements of all their controls when the frame is resized for any reason. In the frame shown below the controls' centering along the sizer's major axis (vertical) is maintained by the two Stretch spacers. The minor axis (horizontal) centering is maintained by the **flag=wx.ALIGN_CENTER** parameters given when each of the two controls were **.Add()**ed to allCtrls_vertSizer**:**

- 

We've seen some of the most fundamental uses of the BoxSizer: Alignment and Spacers. On the next page we'll explore space padding on the controls, themselves, rather than inserting spacers between controls.

Next: BoxSizerFromTheGroundUp/ControlsPadding

BoxSizerFromTheGroundUp

BoxSizerFromTheGroundUp/ControlsPadding

BoxSizerFromTheGroundUp/ControlsResizing

BoxSizerFromTheGroundUp/NestedBoxSizers

BoxSizerFromTheGroundUp/DivideAndConquer

BoxSizerFromTheGroundUp (last edited 2010-09-29 16:49:20 by WinCrazy)