

Wxsmith tutorial: Working with items

From Code::Blocks

Contents

- 1 Working with Menus and Components
 - 1.1 Working with Menus
 - 1.2 Windows and Components
 - 1.2.1 Excercise with Components
 - 1.3 Sizers
 - 1.3.1 wxBoxSizer
 - 1.3.2 wxStaticBoxSizer
 - 1.3.3 wxGridSizer
 - 1.3.4 wxFlexGridSizer
 - 1.3.5 wxStdDialogButtonSizer
 - 1.4 Especially Useful Standard Components
 - 1.4.1 wxButton
 - 1.4.2 wxStaticText
 - 1.4.3 wxTextCtrl
 - 1.4.4 wxPanel
 - 1.4.5 wxChoice
 - 1.4.6 wxComboBox
 - 1.4.7 wxListBox
 - 1.4.8 wxNotebook
 - 1.4.9 wxCheckBox
 - 1.4.10 wxRadioBox
 - 1.4.11 wxGauge

Working with Menus and Components

In the previous tutorial we learned how to create a small application using wxWidgets. We created a window with a few components inside. In this tutorial, we will first see how to use the menu bar at the top of the main window and then focus on sizers and wxWidgets components which can be put on a form or window.

A Note on Terminology

BYO, the creator of wxSmith and original author of these tutorials, wrote "Each resource in wxSmith is made of items. All your buttons, text boxes, lists etc are items. Also some more abstract things like sizers, spacers (used to add empty space), menu entries or even timers are also called items in wxSmith. So basically anything that's inside the resource may be called an item. I'll also use the term component since it reflects the valid meaning too."

In fact, the word "item" is not found in the index of the Julian Smart and Kevin Hock book *Cross-Platform GUI Programming with wxWidgets*. The word is used in the book specifically for the elements of the drop-down lists of choice boxes, combo boxes, and other members of the class `wxControlWithItems`. In the revision of these tutorials, the word "item" is therefore generally being reserved for items in these lists and for items in a menu. Otherwise, the word "item" is generally being replaced by "component", "widget", or "control" as seems appropriate, though of course, it is also occasionally used with no special technical meaning. Also, BYO often uses the word "resource" to refer to a form or window. Smart and Hock use word the "resource" sparingly, mainly in the context of "resource files". The word seems overworked and needlessly abstract, so when what is intended is a form or window, the revisions will use one of those words, with no sharp distinction between them.

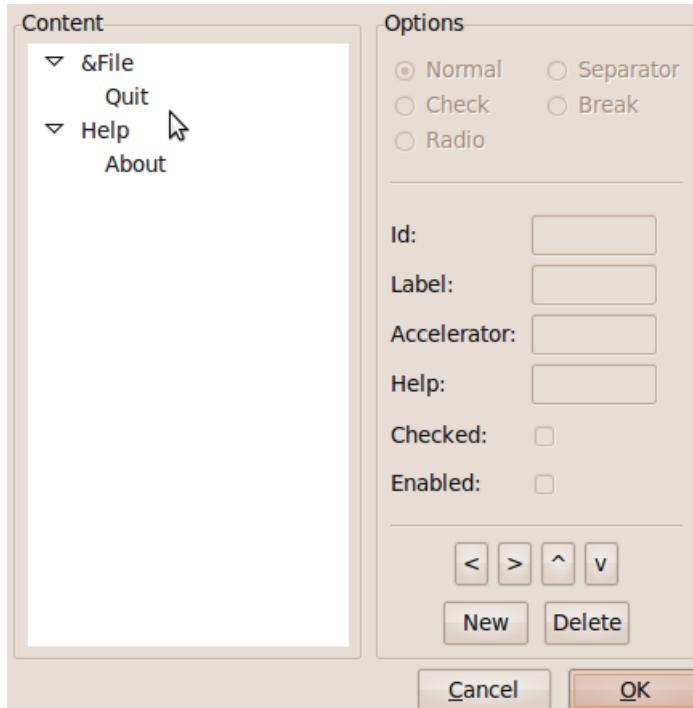
Note that all screenshots presented here were made on Linux but they should not be very different from the windows ones.

Working with Menus

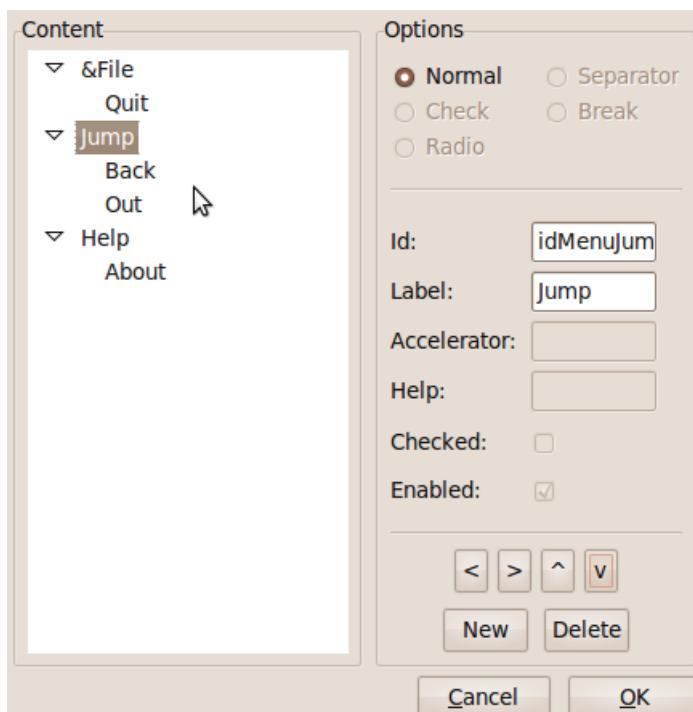
Following the directions given in Tutorial 1, start a new Code::Blocks project; call it Tutorial_2. Remember to add "event.Skip(TRUE);" at the appropriate place in the code, as explained in Tutorial 1.

Build and run the program. You will see that it has two items on the main menu. We will add one more, Jump, at the top level and under it two more, Back and Out. Back will do nothing, but we will make Out close the program.

Click on the "Tutorial_2frame.wxs" tab of the Editor window. Just below the words "Tutorial_2framw.wxs" is a panel with two square icons. Double left click on the leftmost one, which shows the word *File* suggestive of a menu. Up comes the menu inspector window shown below.

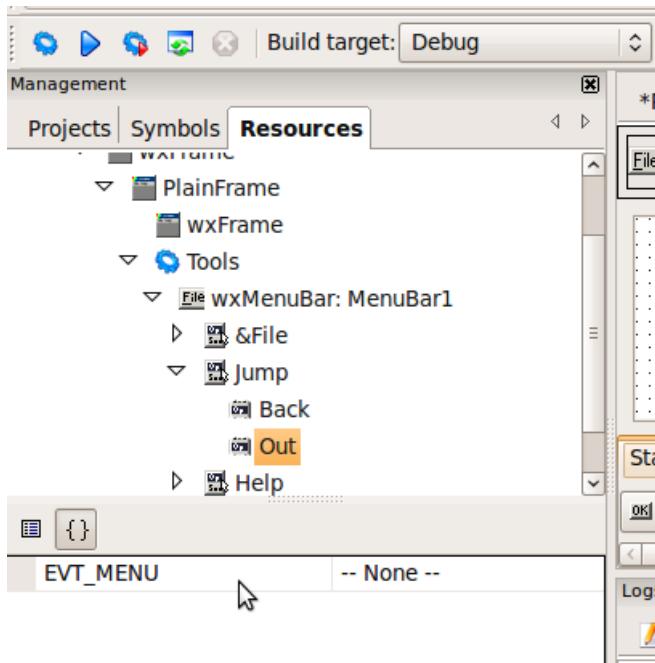


Click on the various menu items shown and note how the content of the fields shown on the right side of the window are filled in. Then click on the New button (in the lower right part of the window) to add the three additional menu items mentioned above, namely Jump at the top level and under it Back and Out. It does not matter where the items first appear; you use the arrow keys just above the "New" and "Delete" buttons to move around the item you have selected. For the "Id" field of the "Out" item, put idMenuOut. You should make the menu inspector window look like this:

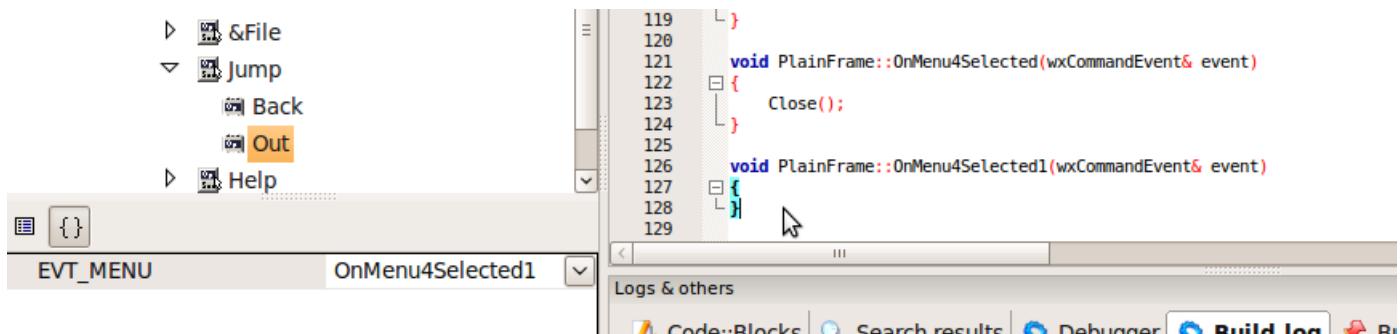


When you have the menu items properly arranged, click OK.

When you now build and run, you will see the new menu. But clicking Jump | Out does not close the program as desired because we have not programmed the response to that event. To do so, we want to make wxSmith set up the frame for the routine which will be called when the menu item "Out" is clicked. Go to the Management pane, Resources tab and persistently click on the little right-pointing triangles until the "Out" item can be seen, as in the screen shot below.



Click on the word "Out" and then on the {} icon in the panel below the Management pane. That will cause the pane below to show the words "EVT_MENU --None--". Click on the word "None" or the down arrow at the right end of the field; a menu drops down. Pick the item "-- Add New Handler --", accept the suggested name (OnMenu3Selected), and over in the editor, the frame of our handler has been written, as shown below. We have only to write the code, which will be simply "Close();".

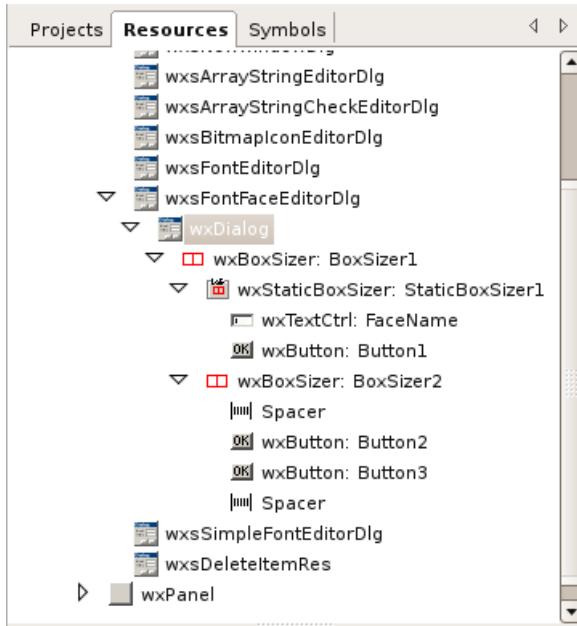


Now we can "Build and run", drop down our menu, click on "Out", and the program closes, as desired.

Windows and Components

Each window in wxSmith is made of components. All your buttons, text boxes, lists, and so on are components. Also some more abstract things like sizers, spacers (used to add empty space), menu entries or even timers are also components.

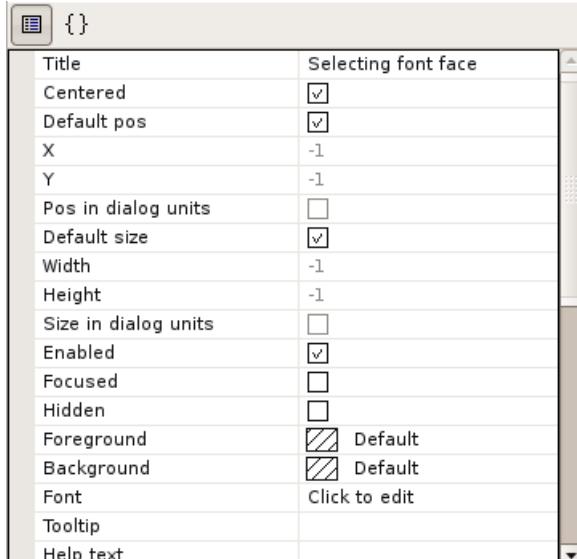
Every window has a root component. For a dialog window, it would be a wxDialog component; for a frame window, it would be wxFrame component; for a panel it would be a wxPanel component. In the previous tutorial you probably noticed that some components may have "children". For example, sizers had components inside their boxes. In this tutorial, you have seen sub-menu items inside main-menu items. The term "children" suggests the idea of a family tree, and indeed precisely such a tree structure is directly represented in the resource browser, as shown in the example below.



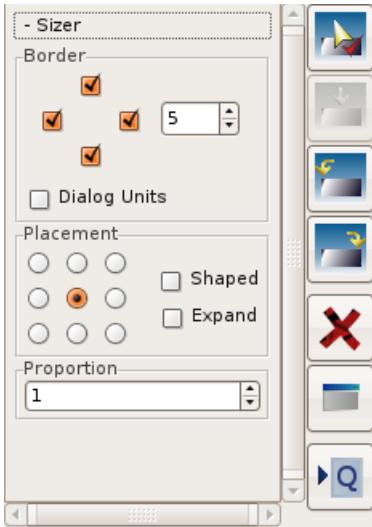
In the picture above, you will see an example of such a tree. `wxFontFaceEditorDlg` is the currently opened form with a root component of type `wxDialog`. We can see that this resource consists of a few sizers and buttons, one text area and some spacers. It's definitely a fairly simple form. For more complicated forms, there could be more than 100 components. The tree (or Resource) browser can be very helpful for locating the code connected with a component; clicking on the component in the browser brings up its code in the editor.

Each component has its properties - these are usually some values which describe the component. Some of them may affect the way component looks, some may change the component's behavior, and some may be dedicated to programming facilities. You will also find that there are some properties -- such as position and size -- common to many components.

`wxSmith` allows you to edit components inside the properties browser (the window which is usually under the resource browser). If you select a component either by clicking on it in the resource browser or by clicking on it inside the editor, the properties editor switches to edit its properties. Here's an example of the property browser:



You can also use the Quick-Properties panel available after pressing the "Q" button on the right side of the editor:

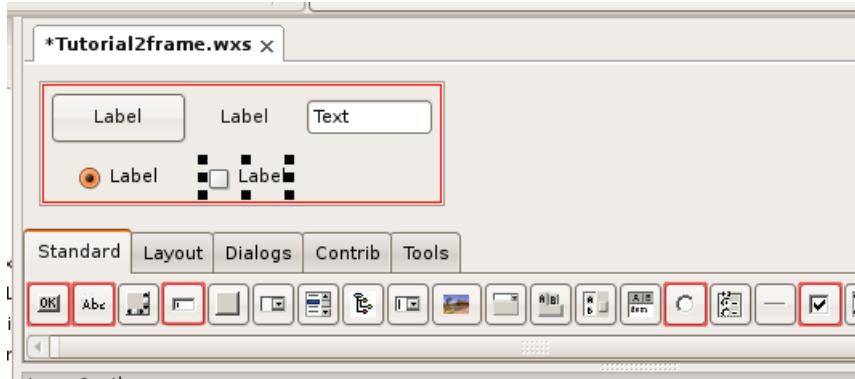


The purpose of this panel is to provide most common properties in user-friendly way. It hasn't proven as useful as I had hoped, and you will probably find it helpful only while adjusting sizer-related settings.

The last and the easiest way to work on a component is directly in the editor. You can grab drag boxes and change the component's size. You can also change its position by simply dragging it to another place.

Excercise with Components

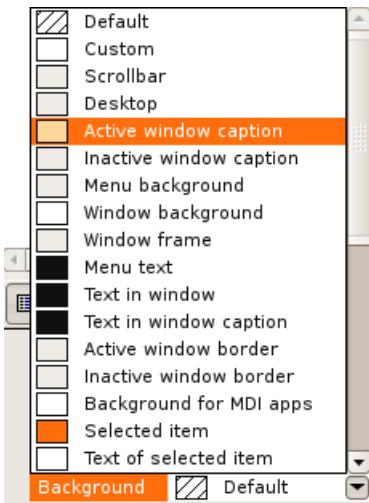
Enough generalities; let's play with components. We can continue with the project started at the beginning of this tutorial. First let's add `wxFlexGridSizer` into our window. Like all sizers, it is located in Layout tab. (You'll remember how to add components from the previous tutorial). Now let's add the following components into the sizer in the order given: `wxButton`, `wxStaticText`, `wxTextCtrl`, `wxRadioButton` and `wxCheckBox`. They can all be found on the Standard tab. The result should look like this:



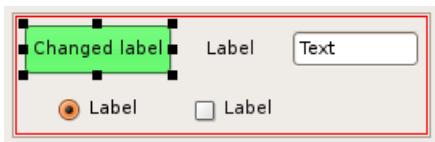
In this picture, components I've selected have a light red border.

Now that we have our playground, let's change some components. First, let's change the label and color of the button. To do so, click on the button and move to the properties browser. "Label" is the first property, and it's quite easy to change. Just click on the word "Label" on the right and type new text.

Color is little bit harder to find. Usually components have two colors - Background and Foreground color. Background, as the name implies, is used to fill the component's background. Foreground, on the other hand, is used to paint some content onto the background. Note that there's no strict interpretation of these colors so you should experiment to find out their meaning. In `wxButton`, Background is what we want. So let's find the Background property. Its value can be changed by using combobox:



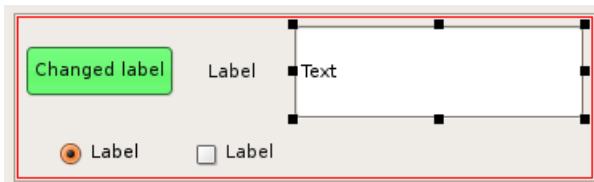
On the list you may find a set of predefined system colors. When you choose one of them, they will be read from the current system settings making your application compatible with the system theme. There are also two special entries - Default (which means that we do not want to set our color) and Custom. When you select Custom, wxSmith will open a color dialog where you can choose any color you like. I have chosen light green, and here's the result:



There are also other properties we could talk about, but the best way to learn them is to play with them, so I leave those discoveries to you.

Now let's use the mouse.

The selected component will have eight black boxes -- drag handles -- around it. These boxes can be used to change the size of the component. You will easily notice when the mouse is over a handle because the cursor changes. Let's resize the wxTextCtrl (edit box):

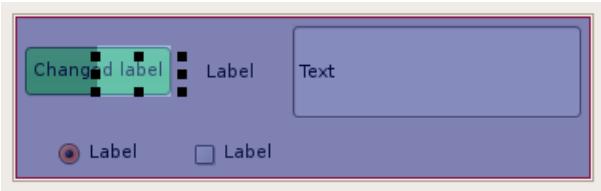


You may see that resizing the component will also change properties in the browser - **Default Size** is unchecked now and **Width** and **Height** are set to required values. There's another property which seems to be untouched - **Size in Dialog Units**, which is also related to size. I'll explain it a bit.

Usually when you must specify a size (or other size-related things) you provide a value in pixels. Alternatively you can specify values in a unit called a **Dialog Unit**. A **Dialog Unit** is usually a little bit bigger than one pixel and its size depends on the size of the current font (actually it's the size of the font of some component's parent which has a font). This is useful when you want some values to be proportional to the size of text presented in the window.

If you check '*Size in Dialog Units*', wxSmith will take care of it and sizes will be calculated in Dialog Units instead of pixels (although, naturally, you will still be able to resize components).

Now let's change the position of the radio box. "But, wait," you may say, "I thought you said that the sizer is responsible for the positioning." That's right. We have limited possibilities to change position. Inside one sizer we can only reorder components; for more complex forms we can also move a component between two sizers or to areas where sizers do not apply. So, back to our example, let's move the radio box to position it right after the button. To do so, click on the radio button and drag it onto the button. When you start dragging, you will notice that the area of the sizer changes color to blue. This color change is an aid to show you what will be the new position of the dragged component. When you move the cursor onto the button, half of the button will become light-blue. When the left half is highlighted, the dragged component will be added before it. When the right half is highlighted, dragged component will be added after it. And when there's no highlighted component, the new component will be added as the last child of the parent:



When you drop the component, it will change its position. Note that all the components after the new position also shift. That happens because the default setting of the `wxFlexGridSizer` allows only 3 components in one row (this can be changed in property browser) so the components just adapted to the new order.



Sizers

Now let's learn a little bit about components available in `wxSmith`. I'll describe all supported sizers and a few important components from the Standard palette. Let's start with sizers.

We already know that sizers are responsible for automatically setting the position and size of some components. But how do they do that? Well, it depends on which sizer has been used. So what are the rules? The generic rule is that the sizer tries to use all its available space and place the components it manages in such way that they do not overlap. If the area available to the sizer is too small, it requests a bigger area, so you can be sure that no component will fall out of the window. (Unfortunately, you cannot be sure that after such adjustments the window will be small enough to fit on the screen, so be careful about that.) Another rule which should be mentioned here is that sizers automatically set-up the minimum size required by the window. Basically when you edit a window using sizers, you should be aware that you edit the smallest layout and without some tricks, you won't be able to resize the window to something smaller.

Here are descriptions of all sizers available in `wxSmith`:

wxBxBoxSizer

`wxBxBoxSizer` is the most basic sizer available in `wxWidgets`. Its purpose is to align components in one line - one next to another - either horizontally or vertically. This sizer also tries to keep some proportions between components - for a horizontal sizer you can set factors which will keep proportionality of widths and the height of the sizer area will be equal to the height of the tallest component. Similarly, vertical sizers keep heights proportional and the width will be taken from the widest component. You specify whether the sizer is to be horizontal or vertical in the sizer's property browser.

Here are examples of layout using box sizers (the first one is the horizontal one, the second is vertical one):



In that example you may see that all components have the same width and height.

But let's assume that we want `Button2` to be two times wider than `Button1` and `Button3` to be three times wider than `Button1`. This can be easily done by using the `Proportion` property of managed components. By default, the value of all components is set to 1 so they have equal size. Changing the `Proportion` of second button to 2 would mean that it should be 2 times wider than the first or third one.

Generally you can set any value you want here. One common technique is to set percentage values - for example setting proportions to values 20, 30 and 50 would mean that the first button should occupy 20% of the space, the second one 30% and the third one 50%. The special value 0 means that this component should be skipped in proportion calculations and it won't automatically adjust its size when the main window is resized.

So if we want to get 1:2:3 proportions, we set 1 for Button1, 2 for Button2 and 3 for Button3. And here is the result:



Also note that if you create a resizable window and resize it, those 3 buttons will automatically grow and keep proportions. If you set the Proportion property for some component to 0, it won't change size as the user changes the size of the window.

The other dimension which is not controlled by the "Proportion" property may be controlled using the "Expand" and "Shaped" properties. They are simple checkboxes. If you check the first one, the component will expand its width/height and will get the size of the biggest component managed by the same sizer. Selecting "Shaped" will also make the component resize but such components will try to keep the initial proportion between width and height so it doesn't have to use all available space. The effect of the "Expand" and "Shaped" properties can be seen when components differ in width or height; here's an example:



In this example, the first button is the biggest of the three. Its size will be used to adjust the sizes of the other buttons. The second one has its "Expand" property checked, and the third one does not. What can be seen here is that the third button expanded in only one dimension (the one managed by "Proportion" property), the second dimension was left untouched.

You may also note that when a component does not fill its available area, it is centered. That's default behavior which can be changed. In the properties of components managed by sizer you will find properties called **Horizontal align** and **Vertical align**. Using these properties you can select where the component should be located when it's smaller than its available area.

wxStaticBoxSizer

This sizer is similar to wxBoxSizer with one exception: it adds an extra static box around managed components:



wxGridSizer

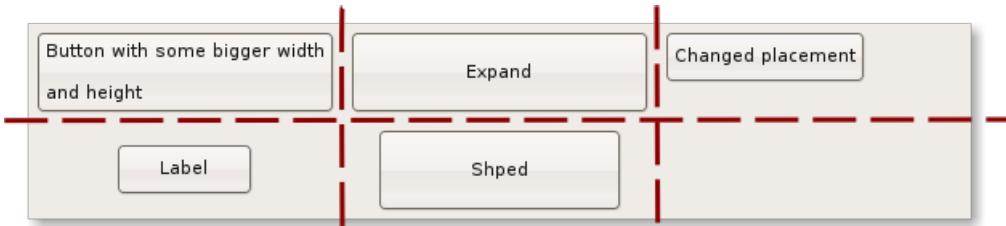
This sizer is a little bit more advanced than wxBoxSizer because it aligns components on a grid, not on one line. You can specify the number of columns or number of rows in properties "Cols" and "Rows". The default number of columns is set to 3 and number of rows is set to 0. When 0 is used, it means that sizer should automatically calculate this value to keep all managed components inside.

This sizer makes an important assumption: all cells must have the same width and all must have the same height. This means that the tallest managed component will set the height of all rows and the widest one will set the widths of all columns. If a component won't use the whole cell, it will be surrounded by empty space. (If you don't like this assumption, look at the wxFlexGridSizer.)

Among the properties of a wxGridSizer you can also find horizontal and vertical spacings. These values set the spacing added between components. The result of using them is similar to using borders around managed components but with borders you would have to set borders for all components separately.

In the case of wxGridSizer, the "Proportion" value is not used. But you may use "Expand", "Shaped" or placement properties to adjust the result.

Here's an example of a grid sizer:



wxFlexGridSizer

This is a very useful one. It works similarly to wxGridSizer; but it does not force all columns to have the same width nor all rows to have the same height. Also not all columns resize when there's more space available for the sizer. Comparing to wxGridSizer you will find two extra properties here: "Growable cols" and "Growable rows". In these properties you may provide a list of columns / rows which should automatically resize by providing a list of numbers separated using comma (,). The numbers should start from 0, so if you would like the second and third column to resize, put "1,2" into "Growable cols". This property is very useful to handle resizing of the window.

Unfortunately you won't be able to set proportions other than 1:1. The "Proportion" property is also not used in this sizer.

wxStdDialogButtonSizer

This is a specialized sizer with one purpose - to provide a set of standard buttons with respect to the platform's look and feel. This sizer has a predefined list of components which it can handle - all are buttons and you can manage them using the sizer's properties. You cannot add your custom components into this sizer.

Each button on this sizer has two properties in the sizer - whether the button should be enabled (for example wxID_OK for OK button) and its label. Note that labels for most buttons will be generated automatically and wxWidgets will skip them when it can read them from the system settings. For example, on Linux only the Context Help button does not have a system-supplied label. It is also very probable that labels will be provided with language-specific strings (depending on the current system language). Not all configurations of buttons are valid - they won't break the application but buttons will overlap (for example Yes and OK).

Here is an example of this sizer. My system is set up in Polish, so the labels are automatically in Polish. They mean: Help, Cancel, Apply and OK.



This screenshot was taken on a Linux box. You may see that the buttons have an extra image which is not available for standard buttons. With such a layout, users of the application will feel more comfortable, and such 'small' things make users say that the GUI was designed very well. On MS Windows, the sizer would have the same buttons but without images and they would be in a different order - the one that is used on windows by default.

All of the above sizers can be found on the "Layout" tab. You can find two other components there which are not sizers but can be used for layout purposes.

The first one is Spacer. It can be added to other sizers when you simply need some empty space instead of any particular component. Spacers have only width and height.

The second one is wxSplitterWindow. When you use this component, you can add one or two child components into it and they will be separated with a dividing line (it's called a sash in wxWidgets). On the application you can drag it to dynamically adjust the size of managed components.

Now that we know something about sizers let's turn to other components.

Especially Useful Standard Components

wxWidgets provides all standard graphical components like static text, buttons, edit boxes, combo boxes, and so on. All are supported in wxSmith, so you can build a nice working environment. I'll briefly describe the components that are really useful in standard applications.

wxButton



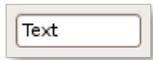
This is standard button, a very common component. Its purpose is to run some piece of code when the user clicks on it. That's it.

wxStaticText



This component only shows user-defined text on the window. It doesn't generate any events.

wxTextCtrl



This component lets the user enter some text. Its basic functionality allows entering one-line text. After changing its properties, it may become a more advanced multiline editor with support for different fonts, colors and text styles.

wxPanel



Although this component may be a separate window, it is most commonly added as a component in a parent window. It is particularly useful as a background for other components. It has a few functionalities - it can be used as background in notebooks and it may be a place where you put components without sizers. But it also has a few drawbacks - wxWidgets tends to do some tricks with wxPanel's size so it's not always possible to set the size we really want.

wxChoice



This component provides a drop-down list with available options. It works similarly to wxComboBox but the user cannot enter values. It can be used in situations when the user should use one of several available options. An event is generated when the user changes the selected component.

wxComboBox



This component provides a drop-down list similar to that of wxChoice. In the case of this component, the user can select one of the available values or enter his own if the component is not in a Read-Only state. A very good example of its usage is to provide some text entry with a stored list of previously entered values (like in search boxes). Like wxChoice, changing the selection generates an event. Additionally, changing the text and pressing enter also generates events. Tutorial 9 will illustrate its use.

wxListBox



This component provides a list where you can select one or more components. Changing the component or double-clicking on it generates an event.

wxNotebook



Notebook is very useful for complex resources since it groups content under tabs. Additionally, wxChoicebook and wxCombobook are also available but are not described here. Changing the selected notebook generates two events: one when the page starts to change (before change) and one when the page is finished changing (after change).

wxCheckBox



Check boxes may be used to map boolean values. Additionally, 2-state boxes can also be generated (with values: yes/no/unspecified). Changing the value emits an event.

wxRadioBox



Radio box works similarly to check boxes. The difference is that only one radiobox in a group can be selected. Groups may be defined by using wxRB_GROUP style. A radiobox which has this style set starts a new group. Changing the selected radiobox emits an event.

wxGauge



This component, also called a Progress Bar, can be used to show the progress of some operation. It's very useful when doing some calculation intensive operations. It can prevent users from getting frustrated when they do not see anything happening on the screen but only a window which seems hung.

That's the end of this tutorial. We will get more experience with these subjects in the following tutorials. But you can also learn a lot by experimenting: add components, change properties, resize, move. You may try adding sizers into other sizers, a very useful technique absolutely required for more complex forms. Join me for the next tutorial!

Retrieved from "https://wiki.codeblocks.org/index.php?title=Wxsmith_tutorial:_Working_with_items&oldid=7573"