course: COMP6210 week: 02 lesson: 01 topic: gui-part2

DOWNLOAD PDF CLICK HERE

WPF Common Controls

Within WPF you can use a lot of predefined controls, but in Visual Studio under the toolbox pane there is a section with common tools.

Most of these controls relate to input, some to output and others to the layout of you page.

In WPF you can use 2 main methods when creating your application.

- Drag 'n' Drop and the property window
- Code everything in XAML

You can interchange between the 2 and the updates will be applied no matter what you do. Here are some use cases for this:

- Your main style is drag 'n' drop, but sometimes you need to quickly change something like a spelling mistake in a label or a specific number in your margins.

- Your main style is XAML, but you don't know the name of a property or need to visually see a control, so you use the toolbox.

Whatever you do it doesn't matter, since the XAML gets created when you use drag 'n' drop.

Let's have a look at some of these common tools and how they work.

For the examples the material below will reference to the controls using the XAML syntax, unless there is a specific reason to mention something else aswell.

By clicking on each item you can see the documentation on the microsoft site.

Input Controls

- TextBox
- RadioButton
- CheckBox
- Button

Arguably you could add the ListBox, DataGrid and the ComboBox to this as well, however they can be populated with data and there for be listed in the output control section as well.

Output Controls

- Label
- TextBlock
- DataGrid

- ComboBox
- ListBox
- Image

What is the difference between a label and a textblock?

Layout Controls

- Border
- Grid
- Rectangle
- StackPanel
- TabControl

Working with Layouts

One of the main difference between Windows Forms and WPF is how the application is rendered onto the screen.

WPF uses DirectX and can therefore provide access to move advanced UI properties like transparency, gradients and scale.

Since WPF, desktop applications can be made with a responsive layout in mind. This means that the control elements can grow and shrink when the size of the application windows changes.

> *Note:* In Windows 10, instead of using WPF you can use UWP and in that framework you can create adaptive layouts where the layout changes.

Using Grids

For the most part you will work with and by default WPF works with a grid layout. A grid is made up of Rows and Columns and you can place controls within the cells that are created by them. The number of Rows and Columns can be set by you for each application and you can set dynamic sizes to them, so that they scale with the application.

You can still use properties like width and height, but you will use them in combination with the layout of your grid.

Each grid is made up of 2 sections:

- `<Grid.ColumnDefinitions>` and `<Grid.RowDefinitions>` to specify the amount of columns and rows and their width and height respectively.
- The Controls that you want to appear on your application, they all have the **attached properties** of `Grid.Column` and `Grid.Row` to specify where you want to see them on the screen.

To specify the size of the columns and rows, you can do one of 3 things:

1. Using the `*` symbol:
When using the `*` symbol, the rows and columns takes up all the remainder space. `2*` scales the space needed to 2 times of the rest of the items. If you have multiple rows or columns, the space is divided accordingly.

2. Absolute size
When using number values (i.e. 100 or 50) the row or column take up that much device-independent units of the screen. This is useful when you need something to be a specific size and not change.

3. Auto size
When you need the size of a row or a column to automatically adjust to the highest or widest piece of content then use the **auto** size setting.

Some Other things

- You are able to nest Grids and place a whole new grid inside of the cell of a parent grid, so this gives you a lot of flexibility.
- You can also set controls to go across multiple columns and rows by using the `Grid.ColumnSpan` and `Grid.RowSpan` attached property on the elements you want to be bigger than a specified cell.

Using Stack Layout

Stack layout allow you to group multiple controls together and set the flow of these objects to be vertical or horizontal. Unlike a grid, a stack layout can only do this for a single row or column.

A stacklayout can be placed inside of a grid cell and can span across multiple cells if required.

This is very useful if you want to group a set of controls together, but not have to deal with a whole grid.

Rectangle

A rectangle is a solid block that holds no content, but can be used as a filler. Rectangles are often used to specify a background color to an area if there is no other content that can be used.

Take Away

By now you should have a good understanding of what kind of controls there are available. Remember these are the "common" ones, every other control can be found in the toolbox list or the documentation site.

The idea of knowing about these controls is that you can now start thinking about nice layouts.

WPF has been around since .Net 3.0, but is not really going anywhere soon as it is being ported into .Net Core 3.0 (due out for release this September).