Blend for Visual Studio 2012 by Example Beginner's Guide

Creating applications with compelling graphics has been one of the main goals of client applications, and with the arrival of WPF, Silverlight, and HTML5 it is much easier than ever before to create interactive and rich user interfaces.

Blend for Visual Studio 2012 by Example Beginner's Guide will give you a good grounding in creating Windows, Web, and Windows Phone applications. You will also look at the various layouts and controls available in Blend and how we can create animations and behaviors in Blend.

Towards the end, you will be aware of the various capabilities that are available in Blend out of the box.

What this book will do for you...

- Learn about layouts and controls offered by Blend and how to use them
- Understand the need for and purpose of XAML and how to use it in Blend applications
- Create applications for Windows 8
- Add behaviors and states to our applications
- Explore vector graphics
- Create user and custom controls and reuse them across applications
- Understand data binding and use it in our applications

The Beginner's Guide approach...

- Clear step-by-step instructions for the most useful tasks
- Learn by doing, start working right away
- Leave out the boring bits
- Inspiring, realistic examples give you ideas for your own work
- Tasks and challenges encourage experimentation

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 2 'Layout Panels'
- A synopsis of the book’s content
- More information on **Blend for Visual Studio 2012 by Example Beginner's Guide**
Abhishek Shukla is a tech lead at Cognizant, Milwaukee, US, and completed his MS in software engineering. Over the years, he has worked with multiple technologies, mostly on the Microsoft platform, and has designed an application for Windows, web, and mobile devices. The biggest project of his career until now has been a banking product named Finacle Advizor (http://www.infosys.com/finacle/solutions/Pages/Advizor.aspx), and he wrote the first lines of code for the product. Thereafter, he has been part of multiple projects based on WPF, Silverlight, ASP.NET, HTML5, and JavaScript. Abhishek enjoys designing and developing applications with cutting-edge technologies and delivering products and applications that have seamless integration with people and processes for optimal results.


The organizations he’s worked for include Infosys, Bengaluru, India; Sapient, Noida, India; and Cognizant, Milwaukee, US.
Creating applications with compelling graphics has been one of the main goals of client applications, and with the arrival of WPF, Silverlight, and HTML5, it is much easier than ever before to create interactive and rich user interfaces. These technologies make use of the computational and graphical power of computers.

This book is a hands-on guide that provides you with a number of clear, step-by-step exercises that will help you take advantage of the real power of Blend in creating WPF, Silverlight, and HTML5 applications. It will give you a good grounding in creating Windows, web, and Windows Phone applications. You will learn about the various tools and techniques that are available in Blend and the different types of applications that we can create using Blend.

By the end of the book, you will be well aware of all the major concepts in Blend and will also be able to develop Windows, web, and Windows Phone applications. You will also be aware of the various capabilities that are available in Blend out of the box.

**What this book covers**

*Chapter 1, Getting Started with Blend*, familiarizes you with the Blend integrated development environment. You will see the various tools provided by Blend and also have a look at how the various panels in Blend are structured.

*Chapter 2, Layout Panels*, helps you understand the various layout panels provided in the WPF and Silverlight frameworks and how the content is managed in these layouts.

*Chapter 3, Working with XAML*, shows you what XAML is and how you can make use of it in your applications. You will see how XAML helps you work with Blend faster and more efficiently.
Preface

Chapter 4, Styles and Templates, teaches you what styles and templates are and how you can create, modify, and reuse them in Blend.

Chapter 5, Behaviors and States in Blend, shows you how you can attach behaviors and actions to elements and how you can use visual states in your applications.

Chapter 6, Understanding Animation and Storyboards, shows you how to create animations in Blend, modify them, and use them in your applications. This chapter also covers how you can create and design storyboards in Blend.

Chapter 7, Understanding DataBinding, covers DataBinding and how it works.

Chapter 8, Vector Graphics, provides you with the understanding of vector graphics and has a look at how it’s different from normal graphics. You will see how you can create vector graphics and advantages of using it.

Chapter 9, User Controls and Custom Controls, teaches you what user controls and custom controls are and how you can create and reuse them.

Chapter 10, Creating Windows Phone Apps, shows you how you can design and develop Windows Phone applications from Blend itself.

Chapter 11, Creating Windows 8 Store Apps, shows you how you can design and develop Windows 8 Store applications from Blend itself.

Chapter 12, Prototyping Using SketchFlow, a bonus chapter, looks at what SketchFlow is about and how it helps you in prototyping designs and getting quick and usable feedback. You can download it from https://www.packtpub.com/sites/default/files/downloads/3882OT_Chapter12.pdf.
A good tool is one that offers a combination of great controls and makes it easy to bring them together to build an amazing user interface.

In Chapter 1, Getting Started with Blend, we installed Blend and familiarized ourselves with the Blend IDE. In this chapter, we will take a look at the various layout panels.

Layout panels, including the one shown in the following screenshot, are components that control the rendering of their children, including the size, dimensions, position, and arrangement of their child content. All panels support the sizing and positioning properties of FrameworkElement. The FrameworkElement class provides the set of properties, events, and methods for WPF elements, and all the panels derive from FrameworkElement.

There are primarily five panels available in WPF:

- **Grid**: This is represented by in the tools panel. It arranges its child controls in a flexible layout of rows and columns forming a grid.
- **Canvas**: This is represented by in the tools panel. It arranges its child controls according to absolute x and y coordinates from the top-left corner of the canvas.
- **StackPanel**: This is represented by in the tools panel. It arranges its child controls in a single line, which is oriented (or stacked) horizontally or vertically.
- **WrapPanel**: This is represented by in the tools panel. It arranges its child controls in a sequence from left to right and from top to bottom. When it runs out of room at the far end of the panel, it wraps the content to the next line.
- **DockPanel**: This is represented by in the tools panel. It arranges its child controls so that they dock one edge of the panel.
Only the Grid, Canvas, StackPanel, and ScrollViewer panels are available in a Silverlight project.

Let's have a look at each of them in detail.

**Grid**

The grid layout panel arranges its child controls in a tabular structure of rows and columns. The grid layout panel allows us to position and style elements easily. This layout panel helps us in structuring our application in the form of a row-and-column layout format.

We have added a few controls in the grid layout panel to make it look like the run command available in Windows. Here's how it looks:

![Grid Layout Panel Example](image)

Grid has three major properties: `RowDefinitions`, `ColumnDefinitions`, and `ShowGridLines`. The `RowDefinitions` property is a collection of `RowDefinition`. Each `RowDefinition` becomes a row in the grid layout. The `ColumnDefinitions` property represents a collection of `ColumnDefinition`. Each `ColumnDefinition` becomes a column in the grid layout. The `ShowGridLines` property represents whether grid lines of a `Grid` panel are visible or not. In the preceding image, we have five `ColumnDefinitions` and four `RowDefinitions`, and `ShowGridLines` is true.
When we don't explicitly define any rows or columns, even then the grid layout has `RowDefinition` and `ColumnDefinition`. This takes up the entire space inside the grid layout in one cell.

When we place more than one element in the same cell, they might end up overlapping as the grid inherently does not have any mechanism to stack or queue items.

**Time for action – creating a Run window using grid**

Let's now use the **Grid** panel to create a **Run** window similar to the one present in Windows:

1. Create a new WPF project and name it `Chapter02`.
2. Right-click on the project name and select **Add New Item...**. Add a new window to the project and name it `GridLayout.xaml`. This is shown in the following screenshot:

![Add New Item](image)

3. Hover the mouse just above the grid area, and you will notice the vertical yellow line that appears with a header. This represents the column that will be added to the grid layout if we click on the grid layout. We will see a similar yellow line, but horizontal, when we move the mouse just left of the grid area. This represents the row that will be added to the grid layout if we click on the grid layout.
4. Add four columns and three rows to the grid layout, as shown in the following image:

While adding columns, if the divider does not land up where we want it, then we can hover the mouse above the column divider, and the cursor will change to show the column divider to be moved. Also, when we hover the mouse just above the grid area within a column, we will have multiple options to change the width of the column as a ratio using *, define a fixed width using pixels, or set the width to Auto. This will change the width according to the content of that column. This is discussed further in the following list:

- **Star**: This specifies the width of the column relative to the other columns. For example, if we specify the width of column 1 as $1^*$ and then the width of column 2 as $2^*$, then column 2 will have twice the width compared to column 1. And, when we specify the width of a column as $^*$, then it occupies the remaining horizontal space inside the grid layout.

- **Pixel**: This fixes the width of the column to be the same as the specified pixels.
- **Auto**: This adjusts the width of the column according to the content of the column. The following set of screenshots shows this:

5. Select **LayoutRoot** from the **Objects and Timeline** panel, and, in the **Properties** panel, set **Height** and **Width** of the grid layout as **450** and **175** respectively. This will make the grid layout match the size of the run command.

6. Also, set **HorizontalAlignment** and **VerticalAlignment** to left and top respectively, and this will place the grid layout in the top-left corner of the window. The following screenshot shows this:

7. After defining the grid layout's row and columns, we will add some content in the grid layout.
8. Go to the text icon in the tools panel and press the left button of the mouse for a few milliseconds and we will see the flyout menu to select the text tool.

9. Select the TextBlock option and draw TextBlock in the grid layout by moving the mouse to the first column and then pressing the left button of the mouse and dragging the mouse without releasing the mouse button. Then, release the mouse once TextBlock of the desired size is created. The following screenshot shows this:

10. Select the TextBlock option, move to the Properties panel, and make sure that we have Row and Column set to 0 and 1 respectively.

11. Also, we have RowSpan set to 1 and ColumnSpan set to 4. RowSpan specifies the number of rows the element will take, and ColumnSpan specifies the number of columns the element will take.

12. Also, set HorizontalAlignment and VerticalAlignment to stretch so that it takes up the space completely. This is depicted in the following screenshot:
13. With TextBlock selected in the Properties panel, set the text of TextBlock to Type the name of a program, folder, document, or Internet resource, and Windows will open it for you. This is shown in the following screenshot:

14. Now, let’s create TextBlock in the first row and zeroth column. Also, we will change the text of TextBlock to Open:

15. Now, we will go back to the tools panel and select TextBox, this time from the text controls, and add it to the first column and first row of the grid layout. Then, we will change the ColumnSpan property of TextBox to span through four columns. We will also change the Background brush of TextBox to a shade of gray. This is incorporated in the following screenshot:
16. From the tools panel, we will now select buttons and add a button in each of the second, third, and fourth columns of the third row:

![Image of buttons added to grid](image1.png)

17. Select each of the buttons one by one, and change the text of the buttons to Ok, Cancel, and Browse…, and the GridLayout window will look similar to the run command of the window:

![GridLayout window](image2.png)

**What just happened?**

We just created a window with grid as `LayoutRoot`. We arranged the elements in the flexible tabular layout available from the grid.

When a grid layout resizes, the controls placed in the cells formed by the rows and columns of the grid layout resize along with the grid layout if the height and width of the element is set to auto. So, if you are looking for a behavior in which you want your child controls to resize according to the space available in the cell in which they are placed, then the grid layout is certainly one of your options. The controls in the cell would resize, but the values to which the margin property is set are not changed. A control can be placed in a specific cell of the grid layout by setting the `Grid.Column` and `Grid.Row` properties available to an element when it is placed inside the grid layout.
We see that the height and width of the line are specified by a number followed by *. This means we are not specifying a fixed width or height as we always do; we are specifying the height and width as a ratio. So, the height of the four rows will be in the ratio 1:1:2:1.5. We can also specify a fixed height, and, if we don't specify any height, then the height of the five columns will be in the ratio 1:2:2:2:2. This discussion is encapsulated in the following code:

```xml
<Grid x:Name="LayoutRoot" Width="450" Height="175"
     HorizontalAlignment="Left" VerticalAlignment="Top"
     Background="White">
     <Grid.RowDefinitions>
         <RowDefinition/>
         <RowDefinition/>
         <RowDefinition Height="2*"/>
         <RowDefinition Height="1.5*"/>
     </Grid.RowDefinitions>
     <Grid.ColumnDefinitions>
         <ColumnDefinition/>
         <ColumnDefinition Width="2*"/>
         <ColumnDefinition Width="2*"/>
         <ColumnDefinition Width="2*"/>
         <ColumnDefinition Width="2*"/>
     </Grid.ColumnDefinitions>
 </Grid>
```

**Have a go hero**

Whenever you add rows or columns to your grid layout, make sure that the ShowGridLines property of the grid layout is either not set or set to false. This will ensure that the border between the rows and columns will be invisible at runtime. If you want these to be visible at runtime, go ahead and set ShowGridLines = true.
Layout Panels

Canvas

The canvas layout panel is another commonly used layout control. As opposed to a grid layout, a canvas layout has no columns or rows, and all controls must be absolutely positioned—provided as offsets from the edges of the canvas layout. But, the elements can still overlap inside the canvas layout as in the case of the grid layout. This means that we can specify the position of an element on the canvas by specifying the top, left, right, and bottom properties of the element. These properties specify the position of the element from the top, left, right, and bottom walls of the canvas layout respectively. The canvas layout is the only layout that allows you to have an absolute coordinate for each object.

Time for action – using canvas

Let’s use the same example that we used in the last section:

1. Right-click on LayoutRoot (grid) and select Change Layout Type | Canvas. This feature is provided by Blend to easily switch between the various layout types. The following screenshot shows this:

We will notice a few things about the layout:

- The gridlines disappear from the layout
- When we select any of the controls, we would notice the left and top properties (as opposed to the Grid.Row and Grid.Column properties in the grid layout). The following screenshot shows this:
2. We can also achieve the same positioning that we have using margins instead of using Canvas properties or a combination of both. For example, let's set the Canvas.Left and Canvas.Top values of the Ok button to 0 and the Margin.Left and Margin.Top values to the values that were set for left and top. This is what Blend does when we switch the layout from canvas to grid, to keep the element positioning same. The following screenshot shows this:

3. If we want to reposition any of the elements, we simply select and drag them, and Blend automatically changes the Canvas.Top and Canvas.Left properties.

**What just happened?**

We changed LayoutRoot in the window from grid to canvas and saw the different positioning patterns followed by the two layout panels.

**StackPanel**

StackPanel is used to stack elements one after another either vertically or horizontally.
Time for action – using StackPanel

We will add a few buttons to StackPanel and learn how StackPanel works:

1. To see how it works, let’s go ahead and add a new window to our project, and let’s name it StackPanelLayout.xaml. Go to the Assets panel and search for StackPanel. Then, drag and drop it onto LayoutRoot.

2. Now, search for the buttons in the Assets panel, and drop three of them onto StackPanel. You will see that, as you add these buttons onto StackPanel, they are added vertically one after another. You can set a height, width, and color for these buttons. Make sure that your code looks similar to the one shown in the following screenshot:

3. We can also arrange the elements with StackPanel horizontally by setting the Orientation property of StackPanel to Horizontal, as shown here:
What just happened?

We saw how StackPanel behaves and how we can use it. We also had a look at the orientation property of the StackPanel.

Other layout containers

Apart from the layout panel that we just talked about, there are other layout containers as well that affect the arrangement of elements in them. However, these containers are not optimized to support complex UI scenarios as the primary layout panels do. The following are the various other layout containers:

- **WrapPanel**: A WrapPanel layout control is similar to a StackPanel layout control, but it allows elements to be placed on multiple lines. When an element overflows off the edge of the panel, it will not be clipped, but instead will be wrapped to the next line. This arrangement of elements could be from left to right or top to bottom. Just like StackPanel, the WrapPanel also provides the option to set the orientation as horizontal or vertical.

- **DockPanel**: DockPanel could be used to describe the overall layout of a simple user interface. DockPanel arranges its children so that each of them fills a particular edge of the panel. If multiple children are docked to the same edge, they simply stack up against that edge in order. This provides an easy docking of elements to the left, right, top, bottom, or centre of the panel. The dock side of an element is defined by the attached property DockPanel.Dock. To fill the remaining space of the panel, we make the LastChildFill property, of the panel, to true.
Layout Panels

- **Border**: This is represented by in the tools panel. It draws the border or background (or both) around its element. A border layout container can contain only one child element; however, that child element can be a layout panel as well.

- **Popup**: This is represented by in the tools panel. It is a window that appears in front of all the other content of the application.

- **ScrollViewer**: This is represented by in the tools panel. It allows the scrolling of its child. However, the child of ScrollViewer can be a panel that scrolls other child elements. We can control scrollbars to be visible, not visible, or automatically visible.

- **UniformGrid**: This is represented by in the tools panel. It arranges its child elements within equal grid regions. It is not an extension of the Grid panel but more of a tiling layout container as it creates equal spacing between each element that it contains, based on the specified rows and columns.

- **ViewBox**: This is represented by in the tools panel. It scales its child element. It can contain only a single element as its child; however, we can place a layout panel with multiple child elements inside it to scale multiple elements.

Building user interfaces

By now, you must have got some idea about the integrated development environment of Expression Blend. You should have a fair idea of a few of the panels available to us to create our user interface and applications and where and how you can find the required control.

Blend provides us with the capability to design and develop a rich user experience. Blend is a great tool because it allows us to build the user interface without coding or knowledge of XAML.

Generally, to build user interfaces, we use one or more of the layout controls to structure our page and application. Then, we place the various in-built controls and custom controls to make up the user interface of our application. In the next chapters of this book, we will create various applications and put the various capabilities of Blend to test.
Pop quiz

Q1. How can we show the border lines of the grid layout?
   1. ShowGridLines = "True".
   2. GridLine = "True".
   3. NoGridLines = "False".
   4. ShowGridLine = "True".

Q2. What are the various layout panels in WPF?
   1. StackPanel & Grid.
   2. Canvas.
   4. All.

Summary

In this chapter, we looked at various layout panels, including Grid, Canvas, StackPanel, WrapPanel, DockPanel, and ViewBox.

In the next chapter, we will take a look at XAML and help you understand how it works.
Where to buy this book


Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.