



Learn by doing: less theory, more results

Blend for Visual Studio 2012 by Example

Leverage the power of Blend to create, modify, and reuse applications and components for Windows using a practical, hands-on guide

Beginner's Guide

Abhishek Shukla

[PACKT]
PUBLISHING

www.allitebooks.com

Blend for Visual Studio 2012 by Example Beginner's Guide

Leverage the power of Blend to create, modify, and reuse applications and components for Windows using a practical, hands-on guide

Abhishek Shukla

[PACKT]
PUBLISHING

BIRMINGHAM - MUMBAI

Blend for Visual Studio 2012 by Example Beginner's Guide

Copyright © 2015 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: July 2015

Production reference: 1230715

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-84969-388-2

www.packtpub.com

Credits

Author

Abhishek Shukla

Project Coordinator

Bijal Patel

Reviewers

Nicholas Armstrong

Mattias Cibien

Alexey Tcherniak

Proofreader

Safis Editing

Indexer

Monica Ajmera Mehta

Acquisition Editor

Kevin Colaco

Production Coordinator

Conidon Miranda

Content Development Editor

Athira Laji

Cover Work

Conidon Miranda

Technical Editors

Vijin Boricha

Humera Shaikh

Copy Editor

Sarang Chari

About the Author

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.

He blogs at <http://www.abhishekshukla.com>.

The organizations he's worked for include Infosys, Bengaluru, India; Sapient, Noida, India; and Cognizant, Milwaukee, US.

This book would never have been possible without the unending support and love of my wife, Easha. Most of the work that I did for this book was done on weekends, nights, vacations, and at other times inconvenient to my family. I want to thank my parents for always helping me follow my ambitions throughout my life, especially my mother, who always spoke only positive things about my work.

I would also like to thank Packt Publishing for showing faith in me and giving me the opportunity to write this book. I would also like to thank everyone who took time out of their busy lives and provided reviews and feedback on the book.

About the Reviewers

Nicholas Armstrong is a software developer and technology enthusiast currently living in Waterloo, Ontario. A graduate in computer engineering at the University of Waterloo (BASC and MASC), Nicholas is currently VP Engineering at Pravala Networks, a start-up focused on improving multinetwork experiences on mobile devices and connected vehicles. Nicholas has traveled throughout Asia and North America to interact with mobile operators, OEMs, automotive suppliers, and other technology companies.

Nicholas's development interests include user experience, interface design, high-performance web applications, and web services. His recent work has focused on delivering high-performance web services to drive mobile clients and web applications on a large scale and single-page web applications built on top of these services. Over the course of his career, Nicholas has worked extensively with Node.js, Android, WPF, SQL, and .NET and has experience in numerous other languages, platforms, tools, and environments.

Learn more about Nicholas at nicholasarmstrong.com.

Mattias Cibien is a C# programmer with a passion for .NET technologies. After graduation, he started working for a company in Milan (Italy) that specialized in Microsoft technologies. After 2 years, he moved on to work for a famous Italian web company.

His primary skills are in C# (WPF, WCF, and MVC), but he has also worked on other technologies, such as C++. His main interest is in 3D technologies, such as Microsoft XNA (MonoGame right now), DirectX, and OpenGL. To know more about Mattias, visit <http://mattiascibien.net>.

I'd like to thank Packt Publishing for letting me review my first book, my wife for supporting me and my passions, and the guys from the university with whom I started doing serious programming.

Alexey Tcherniak is a UI/UX designer and a frontend developer with broad experience in creating desktop, web, and mobile applications. He has been working in the IT industry for over 15 years.

After spending several years building websites and creating graphic arts, his focus shifted to UI/UX design, and currently, he is specializing mainly in .NET-based desktop and mobile applications. He uses C#/XAML to design and develop WPF projects, taking on every project with enthusiasm. Also, Alexey is still a graphics designer, creating icons and illustrations for commercial and free use.

Today, he lives a digital nomad's life and enjoys traveling Europe with his wife and two wonderful children, remotely serving businesses from all over the world. You can find out more about the projects he has participated in at www.alexeytcherniak.com.

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit www.PacktPub.com.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<https://www2.packtpub.com/books/subscription/packtlib>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why subscribe?

- ◆ Fully searchable across every book published by Packt
- ◆ Copy and paste, print, and bookmark content
- ◆ On demand and accessible via a web browser

Free access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface	vii
Chapter 1: Getting Started with Blend	1
Blend for Visual Studio 2012	1
Downloading Blend	2
Time for action – installing Microsoft Blend	3
Creating your first application in Blend	4
Time for action – creating a project in Blend using an existing template	5
The fundamental pieces of the Blend IDE	6
The tools panel	8
Time for action – adding TextBlock	10
Time for action – adding text to TextBlock	13
Brushes	13
The solid color brush	14
Time for action – changing the color of the text	15
The gradient brush	15
Time for action – changing the background color of the grid	16
Linear and radial gradients	18
The tile brush	18
Time for action – changing the background of the grid	18
Time for action – running the application	19
Time for action – integrating the project into Visual Studio	20
Using help and documentation	21
Summary	22
Chapter 2: Layout Panels	23
Grid	24
Time for action – creating a Run window using grid	25
Canvas	32
Time for action – using canvas	32

StackPanel	33
Time for action – using StackPanel	34
Other layout containers	35
Building user interfaces	36
Summary	37
Chapter 3: Working with XAML	39
The basics of XAML	40
Time for action – taking a look at XAML code	41
Time for action – adding other namespaces in XAML	42
Naming elements	43
The code-behind class	44
Time for action – using a named element in a code-behind class	44
Default properties	45
Expressing properties as attributes	45
Time for action – adding elements in XAML by hand-coding	45
Non-attribute syntax	46
Time for action – defining the gradient for the grid	46
Comments in XAML	47
Styles in XAML	47
Defining a style	47
Time for action – defining a style in XAML	48
Using a style	48
Time for action – using a style in XAML	48
Where to go from here	49
Summary	50
Chapter 4: Styles and Templates	51
Creating and using styles	51
An introduction to styles	52
Time for action – creating a resource	53
The resource dictionary	55
Simple styles	56
Creating a simple styled control	56
Changing colors	58
Changing styles	58
Changing control templates	59
Style specification	59
Specifying TargetType of a style	60
Specifying the key for a style	60
Application skinning	61
Time for action – creating resource dictionaries	62
Templates	64

Editing the template	64
Time for action – editing the template	64
Merged dictionaries	67
Summary	68
Chapter 5: Behaviors and States in Blend	69
<hr/>	
An introduction to behavior objects	69
Adding built-in behaviors	70
Types of built-in behaviors	70
Time for action – adding a storyboard	70
Conditional behaviors	73
Data state behaviors	74
Motion behaviors	74
Visual states	76
Visual State Manager	76
Time for action – modifying with visual states	77
Summary	83
Chapter 6: Understanding Animation and Storyboards	85
<hr/>	
Understanding the animation service	86
Storyboards	86
Time for action – adding the storyboard	87
Timelines	90
Timeline recording	90
Properties	91
Animation workspace	91
Time for action – switching workspaces	91
Keyframe	92
Time for action – using keyframes	93
Translation and rotation animation	94
Time for action – using transforms	94
Animation recording symbol	96
Keyframe editing	96
The Timeline zoom feature	97
Storyboard properties	97
XAML for the storyboard	98
Transition between keyframes	101
Easing functions	101
Time for action – using easing functions	101
KeySpline	104
Summary	105

Chapter 7: Understanding DataBinding	107
Understanding dependency properties	107
Understanding the attached property	108
An introduction to DataBinding	109
DataBinding modes	109
The DataBinding model	110
DataBinding properties to control	110
Time for action - DataBinding to one's own property	111
DataBinding control to control	114
Time for action – DataBinding to properties of a different control	114
Using DataSource	117
Time for action – DataBinding to DataSource as a collection	117
Time for action – DataBinding the background with SelectedValue	122
Summary	123
Chapter 8: Vector Graphics	125
An introduction to vector graphics	125
Raster graphics	126
Vector graphics	126
Time for action – zooming in to a WPF control	126
Shapes	129
Time for action – adding a shape	129
Importing graphics	130
Time for action – importing graphics	130
The Line, Pen, and Pencil tools	132
Line	132
Pen	132
Time for action – creating a shape using Pen	132
Pencil	134
Paths	134
Time for action – modifying a Path	134
BitmapScalingMode	137
DPI awareness	138
Summary	139
Chapter 9: User Controls and Custom Controls	141
User control or custom control – which to use and when	142
Understanding and creating a user control	143
Time for action – creating a user control that selects the background color	144
Time for action – adding event handlers	146

Time for action – adding a user control in a window	148
Understanding and creating custom controls	150
Time for action – creating a custom control	150
Summary	155
Chapter 10: Creating Windows Phone Apps	157
Installing Windows Phone SDK	158
An introduction to Windows Phone	158
Guidelines for Windows Phone applications	160
Understanding Windows Phone Emulator	160
Time for action – Windows Phone Emulator	160
Creating a Windows Phone application	162
Time for action – creating a Windows Phone application	162
Exploring the Device panel	170
Testing the application before submitting to the store	172
Time for action – testing our application	172
Submitting our application to the store	176
Time for action – submitting the application	176
Summary	177
Chapter 11: Creating Windows 8 Store Apps	179
Templates	180
Creating Windows Store apps with XAML and C#	181
Time for action – creating a Windows 8 Store app	181
Submitting your app to Windows Store	184
Time for action – submitting the app to Windows Store	185
Stages of app submission	190
Summary	192
Appendix: Pop Quiz Answers	193
Index	197

Preface

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.

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/38820T_Chapter12.pdf.

What you need for this book

You just need a PC with Windows 8 or above and Microsoft Visual Studio 2012.

Who this book is for

This book is aimed at developers and designers who are new to Blend and looking to learn Blend, not just practically, but also conceptually. This book does not assume any knowledge about Blend on the part of developers; however, some experience in design or development might be useful in understanding the concepts faster, but this book explains everything very simply so that you are able to understand everything with little or no effort.

Sections

In this book, you will find several headings that appear frequently (Time for action, What just happened?, Pop quiz, and Have a go hero).

To give clear instructions on how to complete a procedure or task, we use these sections as follows:

Time for action – heading

1. Action 1
2. Action 2
3. Action 3

Instructions often need some extra explanation to ensure they make sense, so they are followed with these sections:

What just happened?

This section explains the working of the tasks or instructions that you have just completed.

You will also find some other learning aids in the book, for example:

Pop quiz – heading

These are short multiple-choice questions intended to help you test your own understanding.

Have a go hero – heading

These are practical challenges that give you ideas to experiment with what you have learned.

Conventions

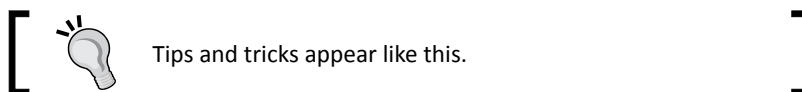
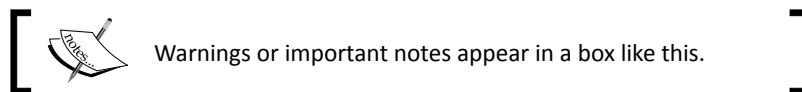
You will also find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Add the name of the application to HelloWorld."

A block of code is set as follows:

```
<Grid.RowDefinitions>
  <RowDefinition/>
  <RowDefinition/>
  <RowDefinition Height="2*" />
  <RowDefinition Height="1.5*" />
</Grid.RowDefinitions>
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Run the installer and make sure you select **Blend for Visual Studio** in the optional features."



Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail feedback@packtpub.com, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files from your account at <http://www.packtpub.com> for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from: https://www.packtpub.com/sites/default/files/downloads/38820T_Graphic_Bundle.pdf.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at questions@packtpub.com, and we will do our best to address the problem.

1

Getting Started with Blend

Blend is a full-featured professional design tool used to create engaging and sophisticated user interfaces for .NET applications with minimal or no code. Whether you are a developer or a designer, Blend offers you capabilities to decrease the delivery time of your .NET applications.

This book is for Blend beginners. In this chapter, we will cover the following topics:

- ◆ How to download and install blend
- ◆ Set up the default environment for application development using Blend
- ◆ Various tools and panels available in the latest version
- ◆ Develop the "Hello World" application using Blend

Blend for Visual Studio 2012

If you are a designer, you could design the visuals using Blend, and Blend will generate the XAML code for it behind the scenes that could be used by the developer. If you are a developer, you still code the XAML by hand as in Visual Studio, but you could also utilize the simple designing capabilities offered by Blend to do your job more efficiently and effectively. This book will teach you how to work with Blend using hands-on examples.

We can develop the following types of applications using Blend:

- ◆ **Windows Presentation Foundation (WPF):** We can design the next generation of Windows client applications utilizing the hardware capabilities of client machines. WPF applications can be both standalone applications as well as browser-hosted applications.
- ◆ **Silverlight:** This is a cross-browser, cross-platform implementation of the .NET framework created to deliver next-generation, rich, interactive media and content over the Web and to develop browser-hosted **Rich Internet Applications (RIAs)**. Silverlight applications can run in browsers as well as in applications.
- ◆ **Windows Phone apps:** We could design the applications for Windows Phones. We will have a look these in *Chapter 10, Creating Windows Phone Apps*.
- ◆ **WPF/Silverlight Prototypes:** We could prototype the application before working on the final design of the application. We will have a look at these in detail in bonus chapter.
- ◆ **Windows Store apps:** Design Metro style applications using XAML or HTML5 and CSS3. We will have a look these in detail in *Chapter 11, Creating Windows 8 Store Apps*.

WPF and Silverlight share a set of features, but their runtime stacks are different. WPF uses the full .NET framework and runs on **Common Language Runtime (CLR)**, whereas Silverlight uses the full .NET framework but executes on the browser-hosted version of CLR.

Downloading Blend

You can buy the full version of Visual Studio at <http://www.microsoft.com/visualstudio/eng/buy>. You can even download the free express edition at <http://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx> to give it a test run.

If you are a student, then you can take advantage of the DreamSpark program from Microsoft at <https://www.dreamspark.com/Student/Software-Catalog.aspx>.

You can also buy MSDN subscription, which offers you the most complete library of Microsoft products and services at <http://www.visualstudio.com/en-us/products/msdn-subscriptions-vs>.

Before you buy or download any version of Visual Studio, you need to decide which types of applications you will develop. To develop all the applications described in this book, we need Visual Studio 2012 Premium or Ultimate. The following is the combination of the various versions of Visual Studio and operating systems and the applications we can develop on them:

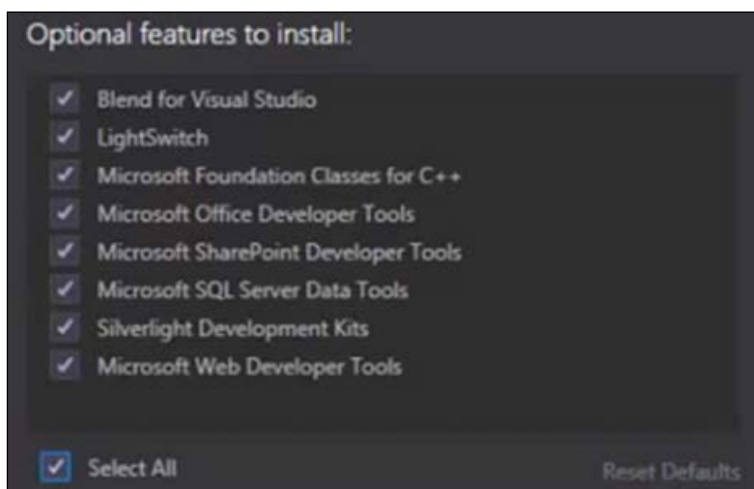
Visual Studio 2012 Update 4	Windows 7	Windows 8
Professional	WPF and Silverlight	WPF, Silverlight and Windows Store
Premium	WPF, Silverlight, and SketchFlow	WPF, Silverlight, SketchFlow, Windows Phone and Windows Store
Ultimate	WPF, Silverlight, and SketchFlow	WPF, Silverlight, SketchFlow, Windows Phone and Windows Store

At the time of writing this book, Blend is available with Visual Studio 2012 only after applying Update 4. You need to download this update from <http://www.microsoft.com/en-in/download/details.aspx?id=39305> and install it to use Blend.

Time for action – installing Microsoft Blend

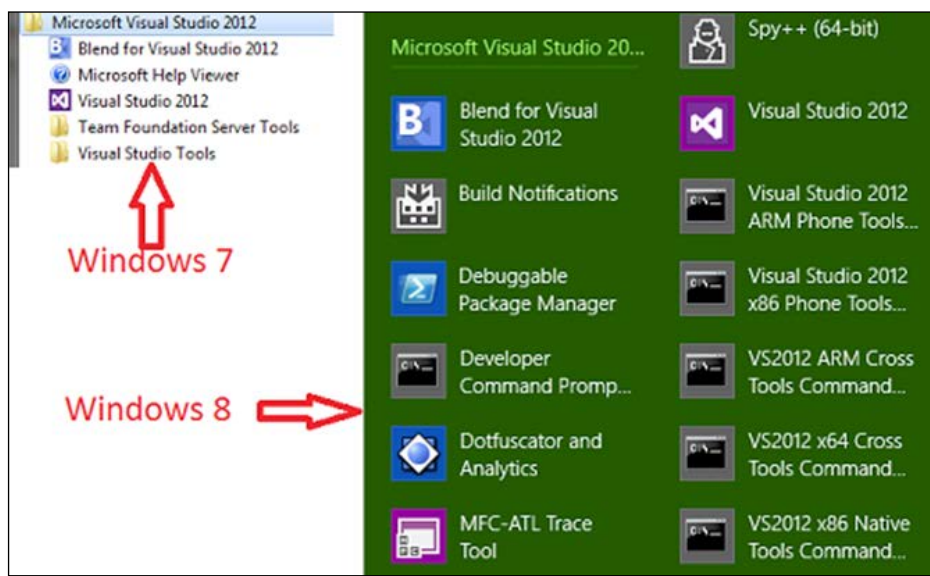
Once you have downloaded Visual Studio, you need to install it as follows:

1. Run the installer and make sure you select **Blend for Visual Studio** in the optional features (if not selected by default), as shown here:



What just happened?

Having installed Visual Studio on our computer, we can see that we have Blend for Visual Studio 2012 along with Visual Studio 2012 in the Start menu. In Windows 7, we can see it in **All Programs | Microsoft Visual Studio 2012**, and in Windows 8, we can see it in Apps under **Microsoft Visual Studio 2012**. This is shown in the following image:



Creating your first application in Blend

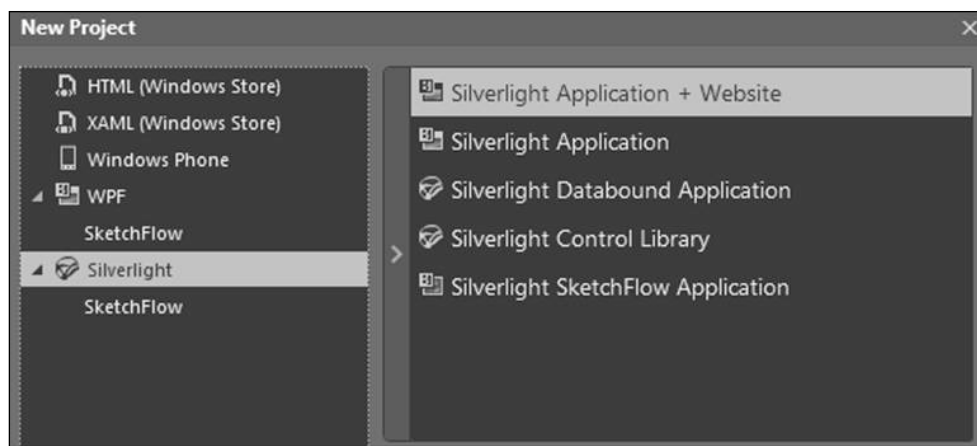
Blend is not a tool for designers who want to use it like Photoshop but is for the task of designing applications. Blend provides simple drag-and-drop options to design the application and a pretty extensive **Properties** panel to customize all the components used. Blend helps us design the complete application, and the output of Blend is ready to use XAML.

XAML (pronounced "zammel") is an XML-based markup language aimed at defining elements in the user interface of a .NET application. It is the language behind the visual presentation of an application that we develop in Blend just as HTML is the language behind the visual presentation of a web page. We will have a look at XAML in detail in *Chapter 3, Working with XAML*.

Time for action – creating a project in Blend using an existing template

Let's create a Silverlight application in Blend:

1. Now, as we have installed Blend, let's go ahead and launch it.
2. If you are on Windows 7, then, from the **Start** menu, go to **All Programs | Microsoft Visual Studio 2012 | Blend for Visual Studio 2012**, and, if you are on Windows 8, then you need to go to **Apps | Microsoft Visual Studio 2012 | Blend for Visual Studio 2012**.
3. We will see the startup screen. Once you have created one or more projects, the screen also shows a list of recent projects for easy access. Go ahead and click on **New Project**.
4. Once we do that, we can see that we have multiple templates available to create different applications in Blend. We can also create new projects in Blend by navigating to **File | New Project** in Blend. The following screenshot shows this:



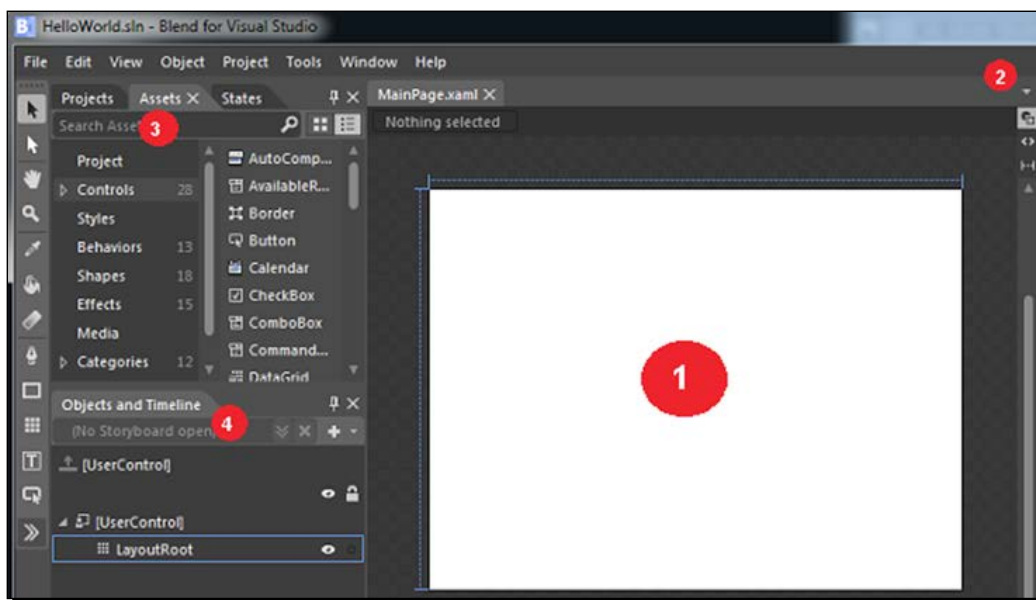
5. Select **Silverlight** in the left panel and **Silverlight Application** in the right panel. Then, add the name of the application to `HelloWorld`. Go ahead and select a location for the project code to reside in. Then, select **Visual C#** as the language and **5.0** as the version and click on **OK**. When we select the individual project on the right panel, you will see the type of application you will create just as we are creating a cross-platform web application here.

What just happened?

We will see that just after creating the project, the blank Blend **integrated development environment (IDE)** gets filled with multiple panels.

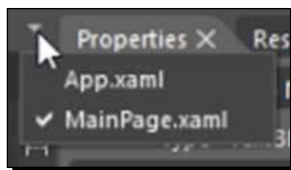
The fundamental pieces of the Blend IDE

We will now navigate through Blend and take a look at the various pieces. We will also see how and where we can find the things you would need to do a specific job. This screenshot represents various pieces of Blend:

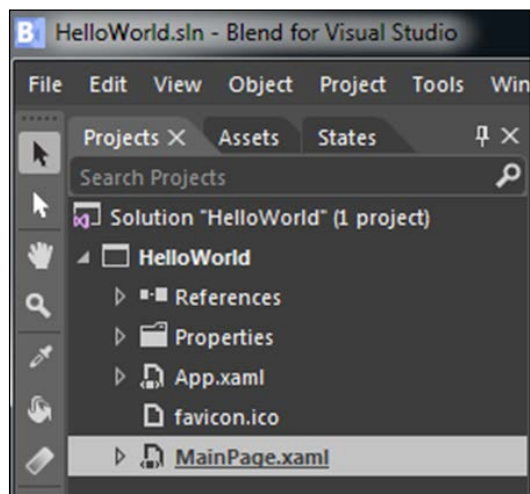


We will explore the pieces of Blend in the following list:

- ◆ **The Art board:** This (highlighted as 1) is where we can see the current design we are working on. It will update as we keep changing the design. Above the art board, we can see the name of the document (`MainPage.xaml`) that we are currently working on. We can open multiple documents in different tabs and navigate between them.
- ◆ **Open documents:** This is highlighted as 2. When we click on the white downward arrow icon in the top-right corner of the art board, we can see the files that are currently open in this project. It is helpful to navigate through them if the list of open files becomes too long to fit in a single row on the screen, as shown here:

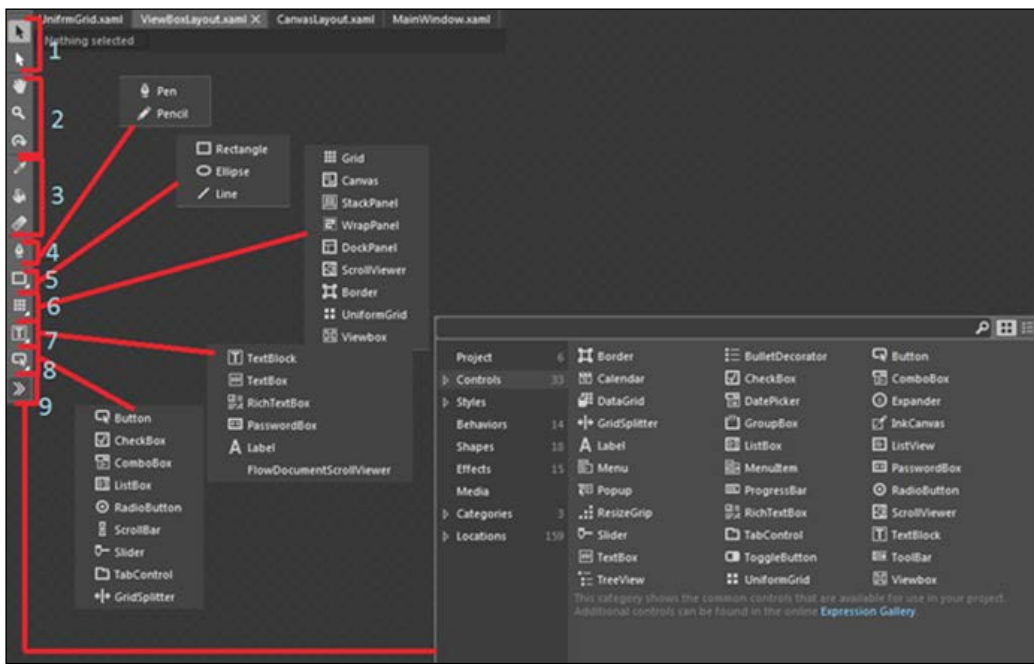


- ◆ **The Assets panel:** This is highlighted as **3**. In the top-left corner, we see the **Assets** panel. This is the place where we have a list of all the controls, styles, media, behaviors, and effects that we could add. We could select the element we want to use or even search for it. We could add the element to the art board by simply grabbing that element and dragging it to the workspace and start working with it. We could also add them to the art board by double-clicking on the element.
- ◆ **The Objects and Timeline panel:** This is highlighted as **4**. We can see the **Objects and Timeline** panel below the **Projects** panel. This panel shows the complete hierarchy of elements in the document that we are working on. We can select the objects that we want to modify.
- ◆ **The Projects panel:** This is highlighted as **5**. Alongside the **Assets** panel, we also see the **Projects** panel. If we click on it, it will show the project we are working on currently. This is the place from where we can view all files of the currently open project. The first item in this panel is a search box where we can search for a project file. This is shown in the following screenshot:



The tools panel

On the extreme left, we can see the tools panel. In this panel, we have the set of common elements that are used to create the UI of our applications. Some of these tools panel items have *multi* buttons (the ones with a little triangle ▾ alongside them). These are the buttons that have multiple functions. So, if we move the mouse cursor to the text icon, as shown in the following image, and press and hold the mouse's left button, we can see the multiple text controls available in the tools panel. If we simply click on the text button, it is the same as selecting the default text control or the last selected text control. There might be a slight difference between the tools that are available in a Silverlight application and a WPF application. The following screenshot shows this:



This is the place where we will get the building blocks of our application. The toolbox can be divided into nine sections:

- ◆ **Selection tools:** This section is highlighted as **1**. These tools allow us to select objects:
 - **Selection:** The black arrow is the selection tool to select any object
 - **Direct selection:** The white arrow is known as the direct selection tool, which is used to select nested objects and paths

- ◆ **View tools:** This section is highlighted as **2**. These tools are used to pan, zoom, and orbit the camera on the art board. The camera orbit tool is not available in a Silverlight project:
 - **Pan:** This is used to position the art board in the workspace area. We can double-click on the pan icon to center the art board in the workspace available for the art board.
 - **Zoom:** This is used to see the different views of the art board and objects. We can double-click on the zoom icon to resize the current document to its actual size (100 percent zoom).
 - **Camera orbit:** This is used to position the camera face for a 3D object.
- ◆ **Brush tools:** This section is highlighted as **3**. These tools are used to work with the various visual attributes, such as foreground, background, and so on. The brush tools are as follows:
 - **Eyedropper:** This allows us to select our color of choice by clicking on any element inside Blend or even elsewhere
 - **Paint bucket:** This allows us fill an area with a brush
 - **Gradient:** This tool helps us to add the gradient effect to the control
- ◆ **Path tools:** This section is highlighted as **4**. The pen and pencil tools are both path tools:
 - **Pencil:** This provides the option to create freeform graphics
 - **Pen:** This provides an easier way to draw complex shapes
- ◆ **Shape tools:** This section is highlighted as **5**. The rectangle, ellipse, and line are used to draw the respective shapes onto the art board.
- ◆ **Layout panels:** This section is highlighted as **6**. These are the various layout panels that help us to design the layout structure of the application screen.
- ◆ **Text controls:** This section is highlighted as **7**. Text controls allow us to display text to the user or take text input from the users.
- ◆ **Common controls:** This section is highlighted as **8**. These are the various input controls available in WPF/Silverlight and are the ones that are most used while developing an application. The other controls can be accessed using the **Assets** toolbar menu described next.

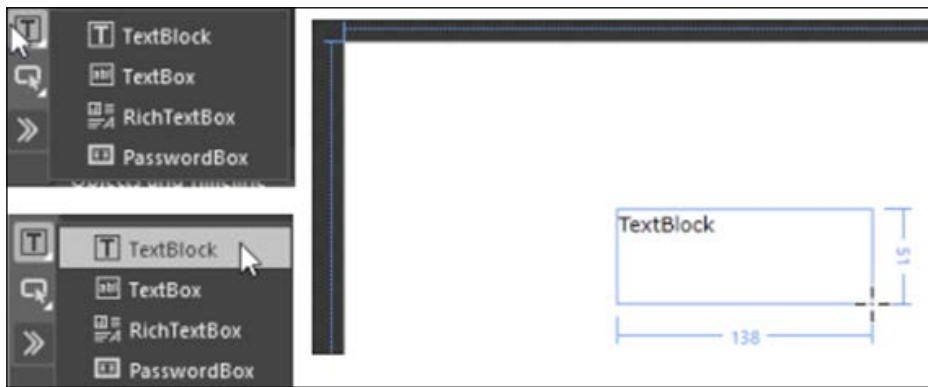
- ◆ **Assets:** This toolbar menu is highlighted as 9. This is the place where we can find all the controls:
 - **UI components:** These include button, label, textbox, menu, and listbox, among others and are referred to as controls
 - **Styles:** This is a group of property settings that determines how a control will appear
 - **Effects:** This is an easy-to-use API to create a graphical effect
 - **Behaviors:** These are reusable code packages that can be added to any object and then fine-tuned by changing their properties

Styles, effects, and behaviors allow us to enhance the controls and add more interactivity. We will have a look at each of these in detail in later chapters.

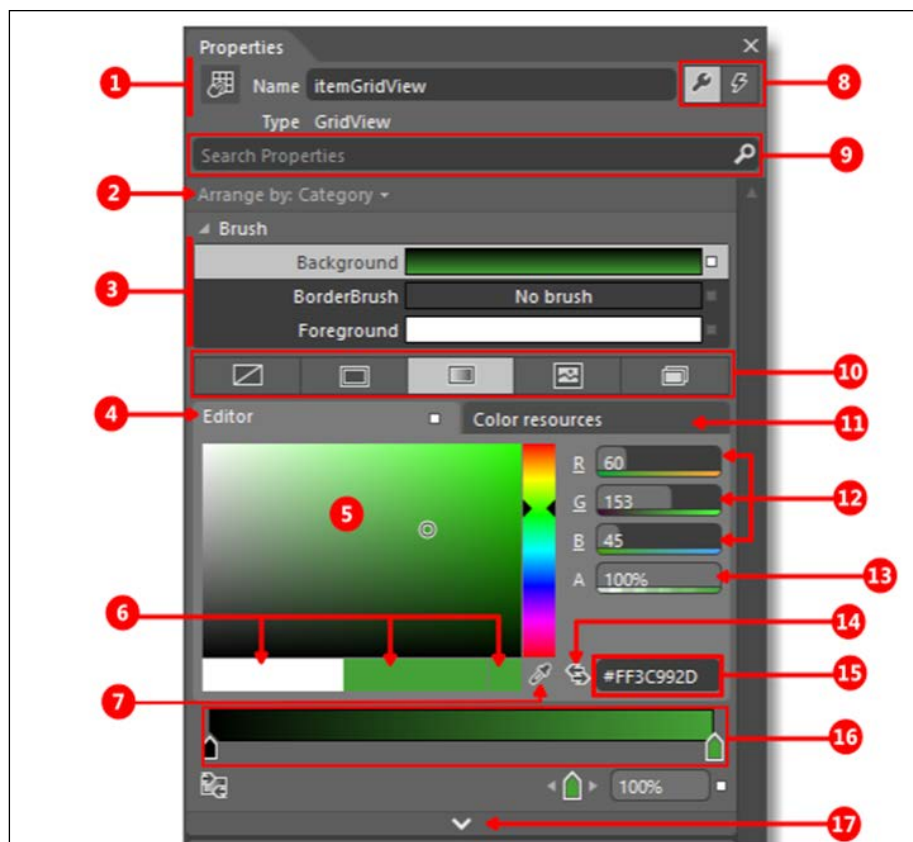
Time for action – adding TextBlock

Let's add **TextBlock** to the art board.

1. Left-click and hold the text tool icon until we see the popup to select one of the text controls.
2. Select **TextBlock** by clicking on it.
3. Move the mouse to the art board, press the mouse's left button down, and, without releasing the mouse button, drag the mouse diagonally to create **TextBlock**. The following screenshot shows this:



4. Once we do that, we can see multiple things. The first thing that we notice is that we have the **Properties** panel filled with various options to configure **TextBlock**. Before we move on, let's take a look at the **Properties** panel in detail:



- 1: This shows **Name** and **Type** of the element selected.
- 2: This allows us to sort the properties by **Name**, **Source**, or **Category**.
- 3: This allows us to select the **Fill**, **Background**, and **Stroke** brushes for geometry elements, such as **Rectangle**, **Ellipse**, and **Path**, and **BorderBrush**, **Foreground**, and **Background** for elements inherited from **UIElements**, such as **TextBlock**, **Button**, and so on.
- 4: This is used to select the solid and gradient brushes.
- 5: This is used to select the color of choice by moving this slider.
- 6: Here, the three sections show initial, current, and last color respectively.

- **7:** Eyedropper can be used to select any color within Blend or outside Blend. Different eyedroppers are available depending on where the solid and gradient brushes are selected.
 - **8:** This allows us to switch between events and properties available for the selected element.
 - **9:** This allows us to search and reach the properties faster.
 - **10:** This is used to switch between no brush, solid brush, gradient brush, tile brush, and brush resource.
 - **11:** This shows the color resources available.
 - **12:** This shows the RGB equivalent values (0–255) of the selected color. RGB values can also be set here.
 - **13:** This shows the alpha value and could also be set.
 - **14:** This is used to convert colors to resources.
 - **15:** This shows and edits the hex equivalent value of the selected color.
 - **16:** This shows the gradient slider, along with the various gradient stops.
 - **17:** This shows the other properties available for the element.
- 5.** Also, you will notice that, in the **Objects and Timeline** panel, we have a new element named **TextBlock** below **LayoutRoot**.
- 6.** We can rename the control from the **Objects and Timeline** panel.
- 7.** Also, a small * sign appears with the `MainPage.xaml` filename in the projects panel. This means that we have made some modifications in the file, but the changes are not yet saved.

What just happened?

We added our first element to the panel, and that is **TextBlock**. Similarly, we can add any number of controls we want to the panel.

Time for action – adding text to TextBlock

We can see that the default text in **TextBlock** is selected. Perform the following steps to add text to **TextBlock**:

1. Let's type `Hello World` inside **TextBlock** and press the *Esc* key. We can see that we come out of **TextBlock**. If we need to go back into editing mode, we should double-click **TextBlock**.
2. In the tools panel, we see two different kinds of arrow buttons. One of them is the selection arrow (black arrow), and the other one is the direct selection arrow (white arrow). The selection arrow helps us select the element and make changes in the sizing of the element, whereas the direct selection arrow helps us make changes in the positioning of the element. So, click on any of these selection arrows and then on the textbox in the art board, and it will select the properties of **TextBlock** in the **Properties** panel. We can now configure the various properties of **TextBlock**, such as the foreground color, visibility, height, width, alignment, text font, and so on.

What just happened?

We modified the text inside **TextBlock**.

Brushes

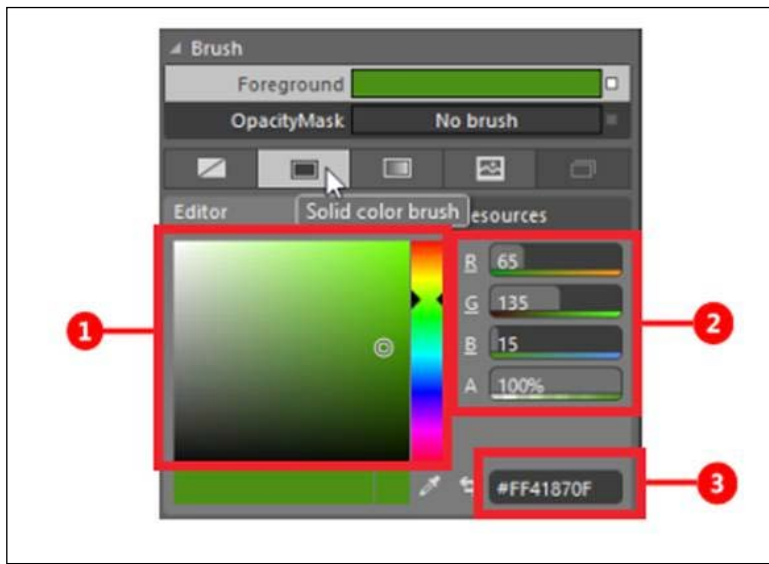
A brush paints an area with its output, and different brushes have different output. A brush can be used to describe the fill of a shape, foreground of a text, or background of a control. We can define these brushes using a solid color or a complex set of patterns and images. Most of the visual objects allow us to specify the brush to paint them. The following is a list of common controls, with their properties, on which we can use a brush:

Class	Brush properties
Border	BorderBrush and Background
Control	BorderBrush and Background
Panel	Background
Pen	Brush
Shape	Fill and Stroke
TextBlock	Background and Foreground

We have the following types of brushes.

The solid color brush

Solid color brush paints an area with a solid color. We can specify **Solid color brush** for a control in multiple ways. The following screenshot represents the options available with **Solid color brush**:



These options are discussed further in the following numbered list:

- ◆ 1: The color editor allows us to select the color by dragging around the circular marker
- ◆ 2: We can specify the alpha, blue, green, and red channels for the color
- ◆ 3: We can specify the hex value of the color

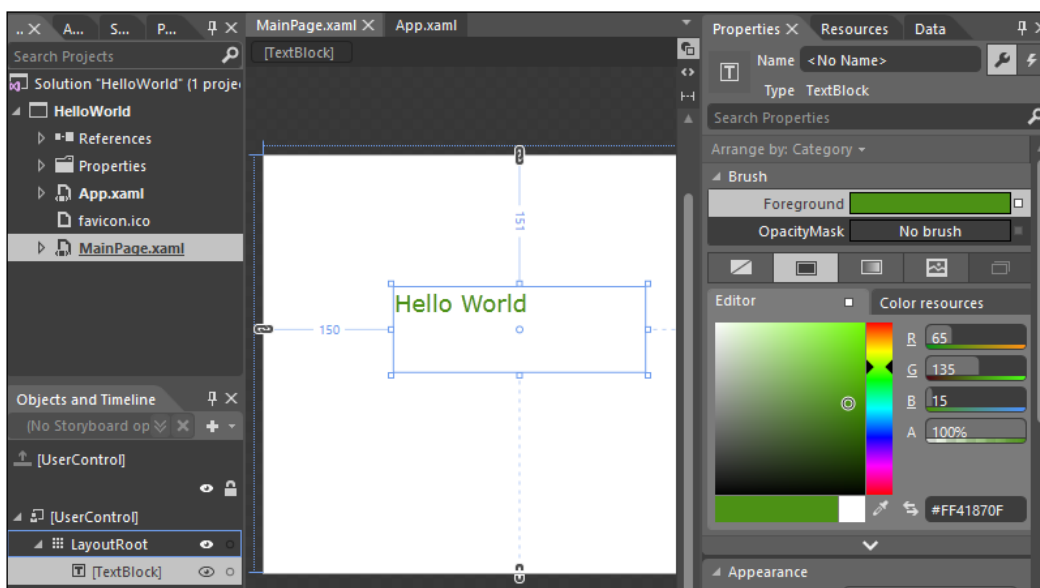


When we set the color using any one of these ways, the other two values are automatically updated to reflect that change as well.

Time for action – changing the color of the text

Let's assign a color to the text we are displaying:

1. Select **TextBlock** by clicking on it, and then, on the **Properties** panel, select **Foreground**. When we do that, **Editor** becomes visible.
2. Click on the solid color brush and move the circular marker to a shade of green. You can set the values of the red, green, and blue channels as well.



What just happened?

We just changed the color of the text of **TextBlock** to solid color brush.

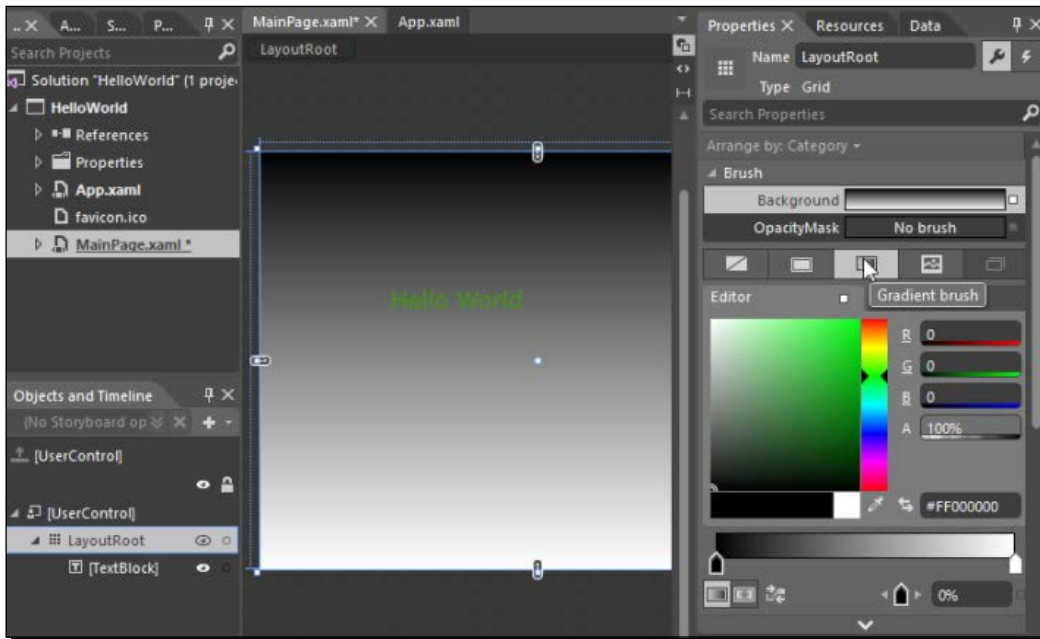
The gradient brush

The gradient brush provides us with the option to specify a sequence of colors for our element. We use `GradientStop` objects to specify the colors in the gradient and their positions. We could have any number (a minimum of two) of `GradientStop` with the same color or a different color in `GradientBrush`.

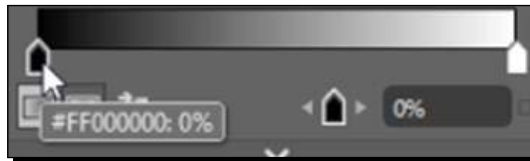
Time for action – changing the background color of the grid

Let's change the background color of the grid:

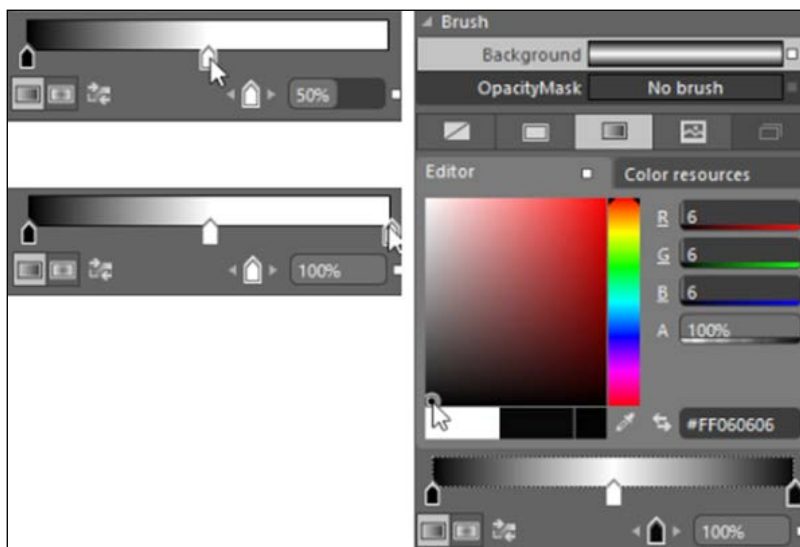
1. Select `LayoutRoot` and then, on the **Properties** panel, select **Background**. When we do that, **Editor** becomes visible.
2. Click on the gradient brush. When we do that, we get to see the default gradient brush offered by Blend. It has two gradient stops: one at offset 0 (Black) and one at offset 100 (white). The following screenshot shows this:



3. When we move the mouse over these gradient stops, we see the color of the stop and the position of the gradient stop on the gradient slider. The following screenshot shows this:



4. Move the white gradient stop to the center by clicking and dragging it, and then add a new gradient stop at the end of the gradient slider by clicking on the gradient slider. Now, change the color of this gradient stop to black. The following screenshot depicts this:



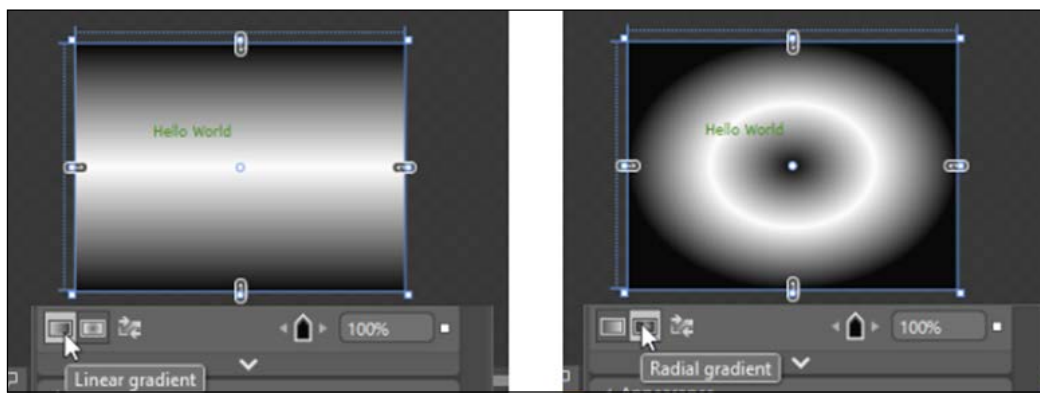
5. When we slide these markers, we can see the gradient shift along with that. To remove a marker, we can either click on a marker and press the *delete* key or drag the gradient stop off the bottom of the gradient slider.

What just happened?

We just changed the color of the background of the grid to the gradient brush.

Linear and radial gradients

Our gradient brush could be linear or radial. The one that we see here is the linear gradient brush, which blends two or more colors across a line, that is, the gradient axis. The radial gradient brush blends two or more colors across a circle. The following screenshot shows this:



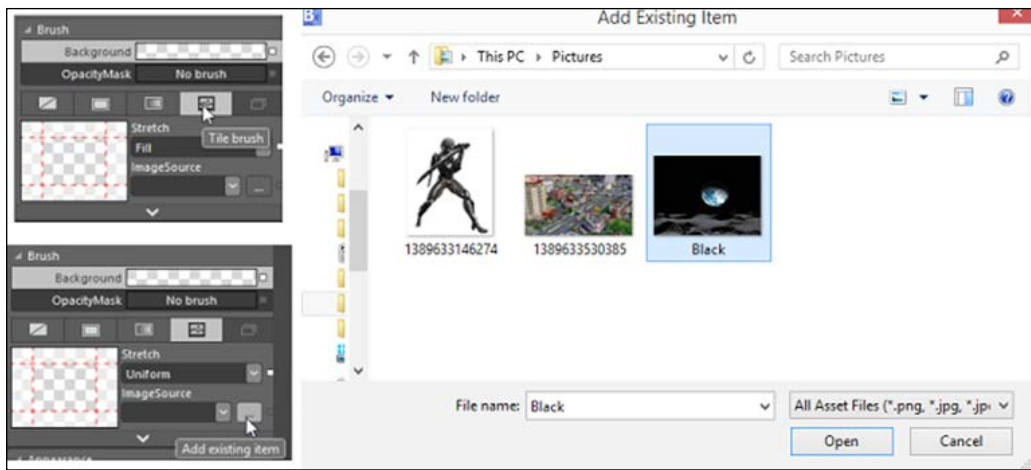
The tile brush

The tile brush paints an area with a repeating image or pattern. We can create a tile brush from an image brush, drawing brush, or visual brush. Here, we will use an image brush to paint an area using an image.

Time for action – changing the background of the grid

Let's change the background of the grid using an image brush:

1. Select **LayoutRoot**, and then, on the **Properties** panel, select **Background**. When we do that, the editor becomes visible.
2. Click on the tile brush. When we do that, we see that the background of the grid is reset and we have the option to select the stretch and source of the image.
3. Click on the **Browse** button next to **ImageSource**, and browse and select an image we want to use for the image brush for the background of the grid. This sequence of steps is shown here:



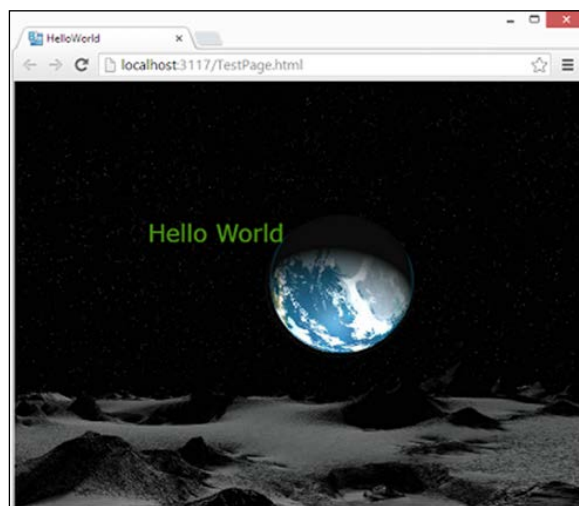
What just happened?

We just changed the color of the background of the grid to the image brush

Time for action – running the application

Let's run the application that we developed:

1. In the **Project** menu, click on **Run Project**. Alternatively, we can use *F5* or *Ctrl + F5*.
2. When we perform any of these actions, we can see the project being built and then our application in the default web browser. The following screenshot shows this:





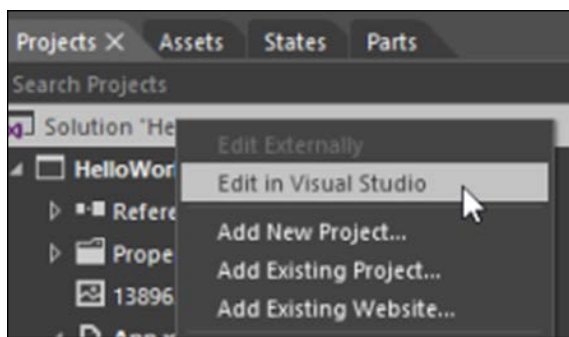
We see a browser because we are running a Silverlight application. If we were running a WPF application, then we would see a window.

What just happened?

We just ran the application for the first time.

Time for action – integrating the project into Visual Studio

1. Right-click on the solution file, and select **Edit** in Visual Studio. It will open the same project in Visual Studio 2012. Now, open `MainPage.xaml`:



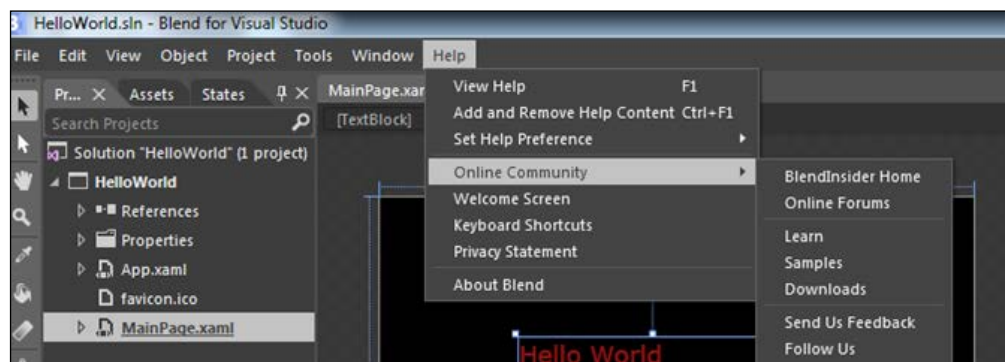
2. Go back to Blend, change the foreground color of **TextBlock** to red, and save the changes.
3. Go to Visual Studio, and you will see a prompt. This shows that the files that we have been working on have been modified in Blend and we need to reload it to work with the updated version.
4. The same thing would happen if we edited a file in Visual Studio and saved it. Blend will prompt us to reload the file to start working with the latest version of the file.

What just happened?

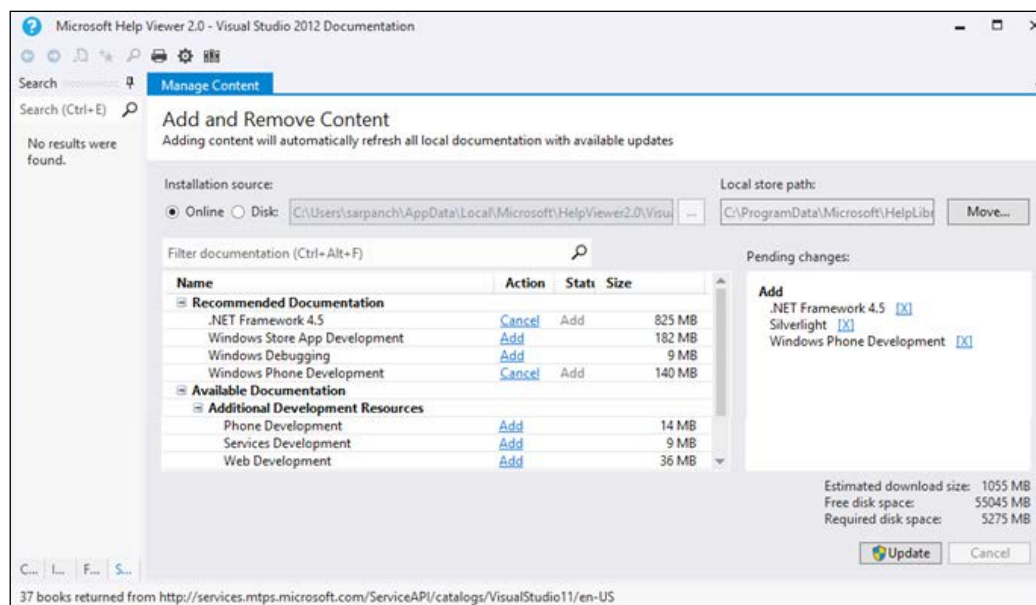
We had a look at the integration of Blend into Visual Studio, and when we change and save one or more files in any of the IDEs, we will be notified of the changes in the other.

Using help and documentation

We can navigate to help from the **Help** menu or by pressing *F1*:



Pressing *F1* will lead us to contextual help, which will open **Design Windows Store apps using Blend for Microsoft Visual Studio**. *Ctrl + F1* will allow us to manage the help content installed on the local machine. The following screenshot shows this:



Have a go hero

Drag and drop more controls onto the art board. Change their various properties, and run the application.

Pop quiz

Q1. How to run a project in Blend?

1. F4 or Ctrl + F4.
2. F5 or Ctrl + F5.
3. F6.
4. F8.

Summary

We installed Blend for Visual Studio 2012 and created the "Hello World" application using Blend. You specifically learned about the versions of Blend 2012 available, Blend IDE layout, and project templates available in Blend. You also saw how to create a project in Blend, use the brush tool, and run a project in Blend.

Now that we have the basics of Blend ready to create an application, we will look into layouts and controls in the next chapter.

2






Layout Panels


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.

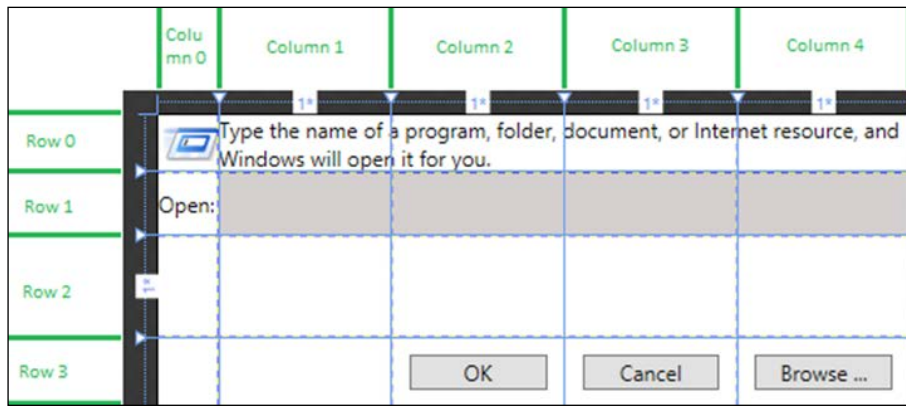
 Only the Grid, Canvas, StackPanel, and ScrollViewer panels are available in a Silverlight project.

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 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.



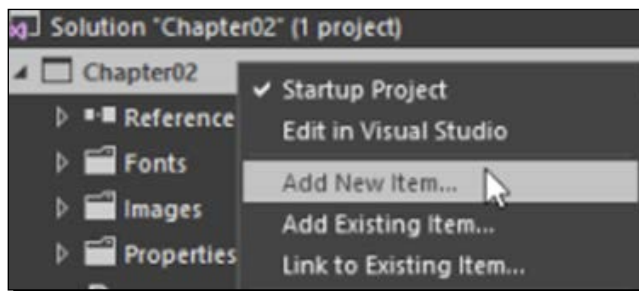
When we don't explicitly define any rows or columns, even then the grid layout has `RowDefinition` and `ColumnDefinition`. This takes up the entire space inside the grid layout in one cell.

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.

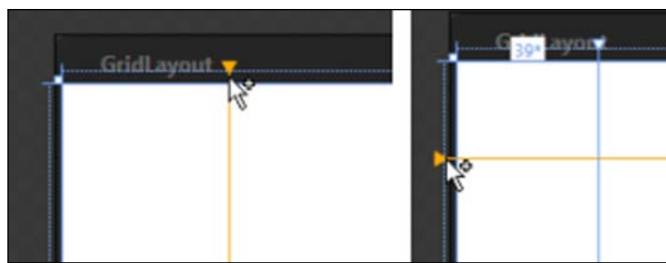
Time for action – creating a Run window using grid

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

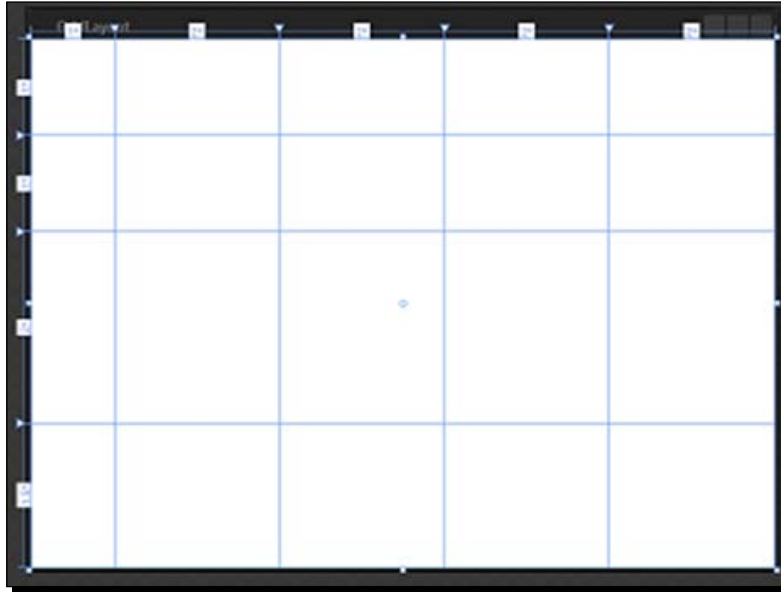
1. Create a new WPF project and name it `Chapter02`.
2. Right-click on the project name and select **Add New Item....** Add a new window to the project and name it `GridLayout.xaml`. This is shown in the following screenshot:



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:



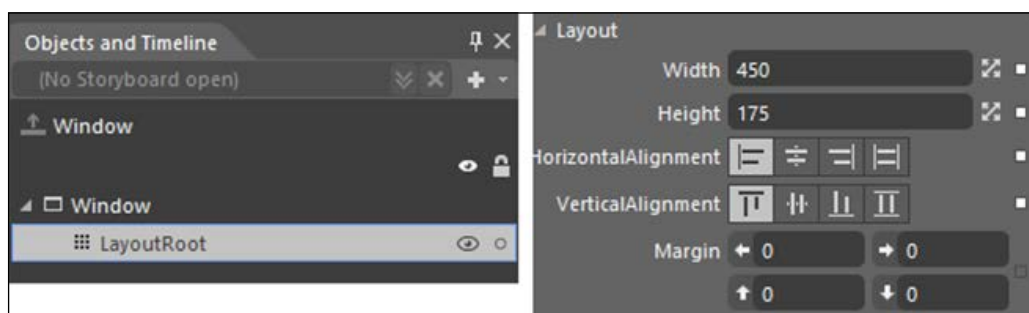
While adding columns, if the divider does not land up where we want it, then we can hover the mouse above the column divider, and the cursor will change to show the column divider to be moved. Also, when we hover the mouse just above the grid area within a column, we will have multiple options to change 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:

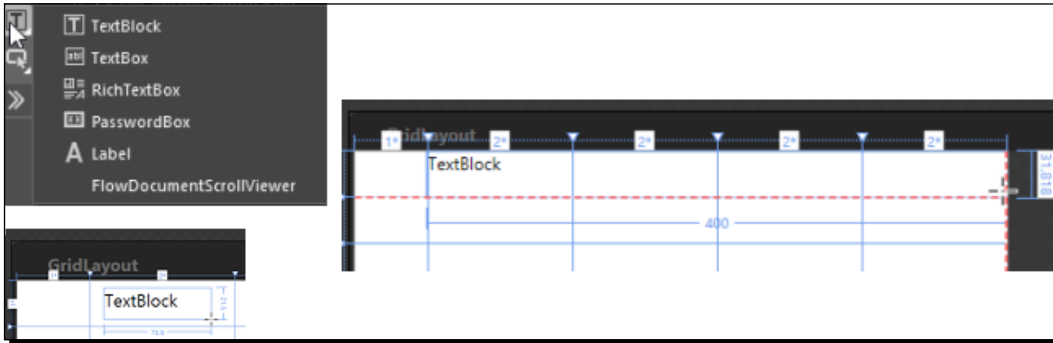


5. Select **LayoutRoot** from the **Objects and Timeline** panel, and, in the **Properties** panel, set **Height** and **Width** of the grid layout as **450** and **175** respectively. This will make the grid layout match the size of the run command.
6. Also, set **HorizontalAlignment** and **VerticalAlignment**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

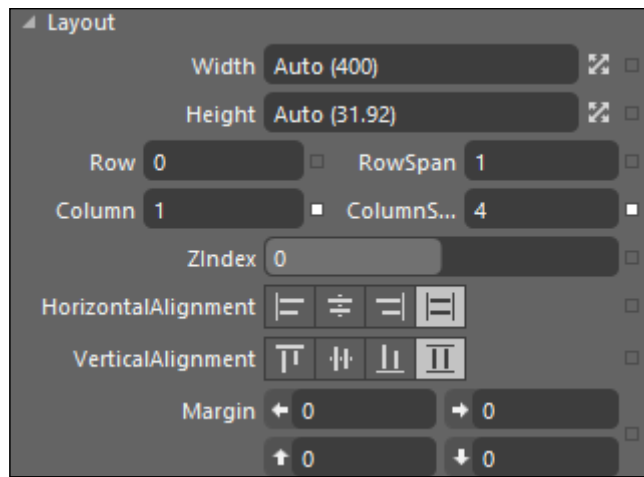


7. After defining the grid layout's row and columns, we will add some content in the grid layout.

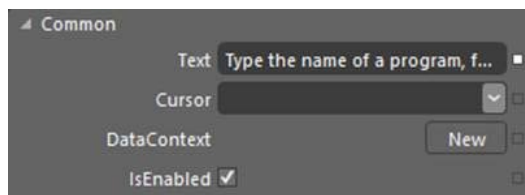
8. Go to the text icon in the tools panel and press the left button of the mouse for a few milliseconds and we will see the flyout menu to select the text tool.
9. Select the **TextBlock** option and draw **TextBlock** in the grid layout by moving the mouse to the first column and then pressing the left 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:



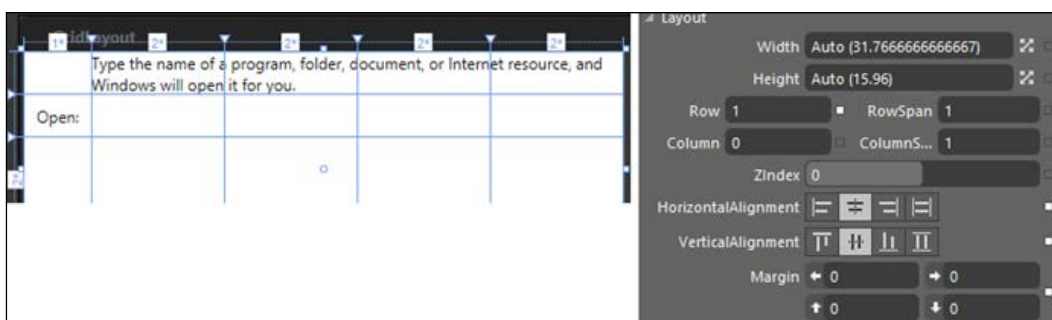
10. Select the **TextBlock** option, move to the **Properties** panel, and make sure that we have **Row** and **Column** set to 0 and 1 respectively.
11. Also, we have **RowSpan** set to 1 and **ColumnSpan** set to 4. **RowSpan** specifies the number of rows the element will take, and **ColumnSpan** 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:



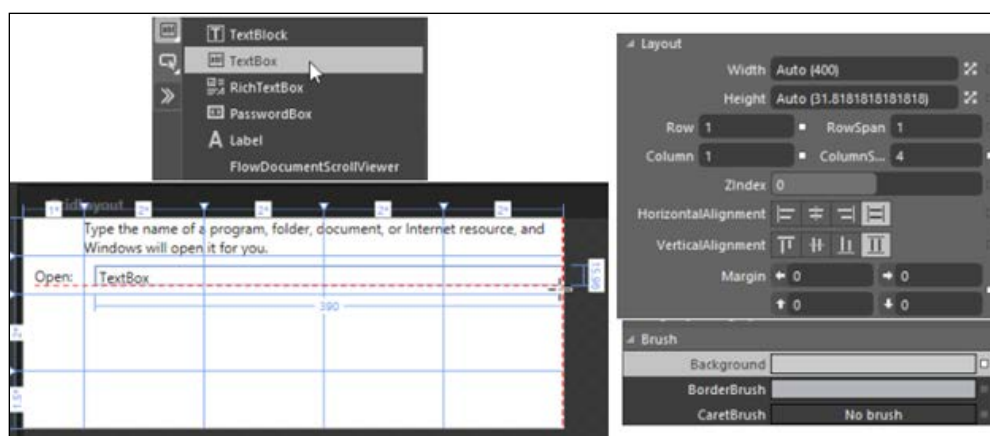
- 13.** With **TextBlock** selected in the **Properties** panel, set the text of **TextBlock** to **Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.** This is shown in the following screenshot:



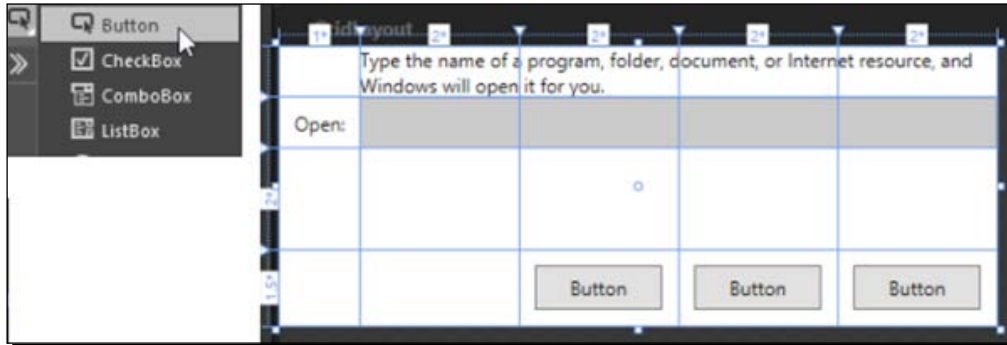
- 14.** Now, let's create **TextBlock** in the first row and zeroth column. Also, we will change the text of **TextBlock** to **Open:**



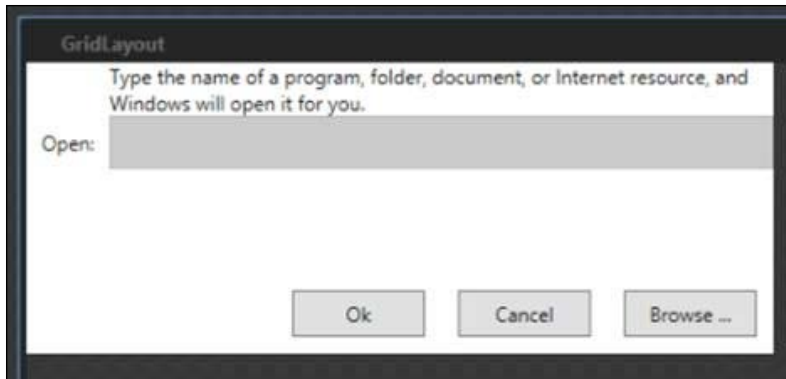
- 15.** Now, we will go back to the tools panel and select **TextBox**, this time from the text controls, and add it to the first column and first row of the grid layout. Then, we will change the **ColumnSpan** property of **TextBox** to span through four columns. We will also change the **Background** 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:



- 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:



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:

```
<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>
```



Downloading the example code

You can download the example code files from your account at <http://www.packtpub.com> for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

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`.

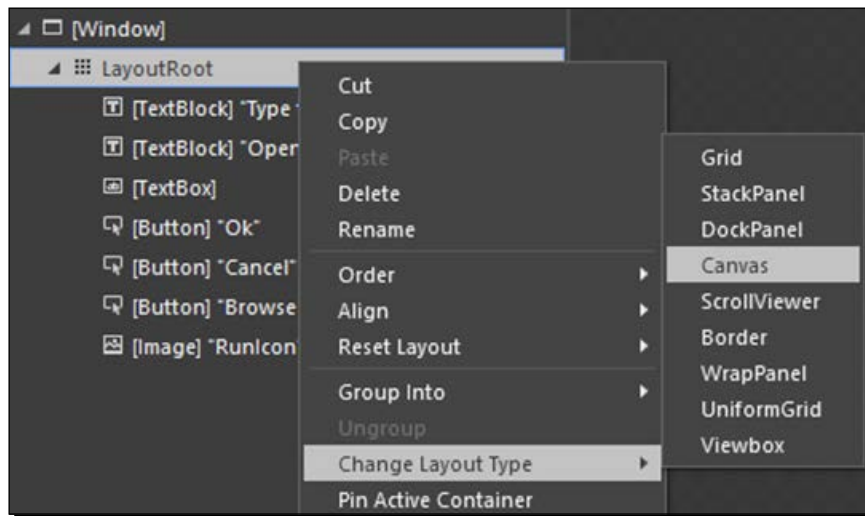
Canvas

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

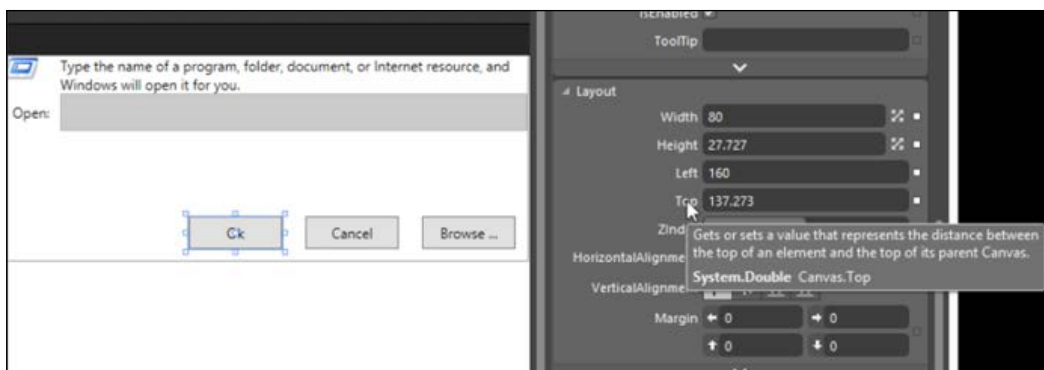
Let's use the same example that we used in the last section:

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:

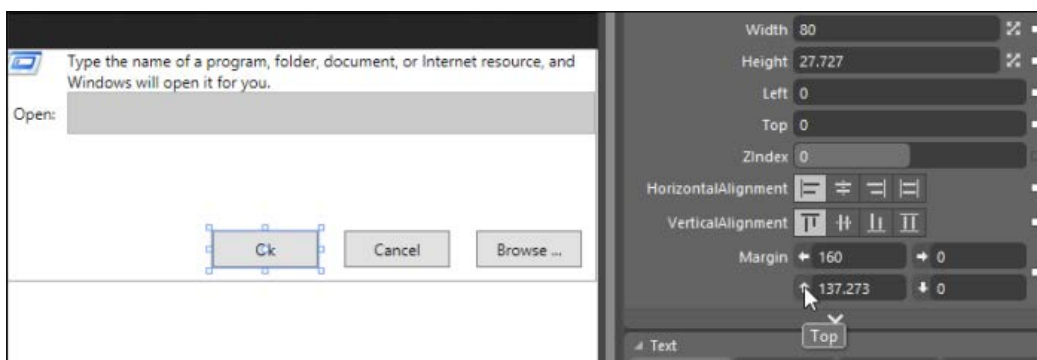


We will notice a few things about the layout:

- 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.

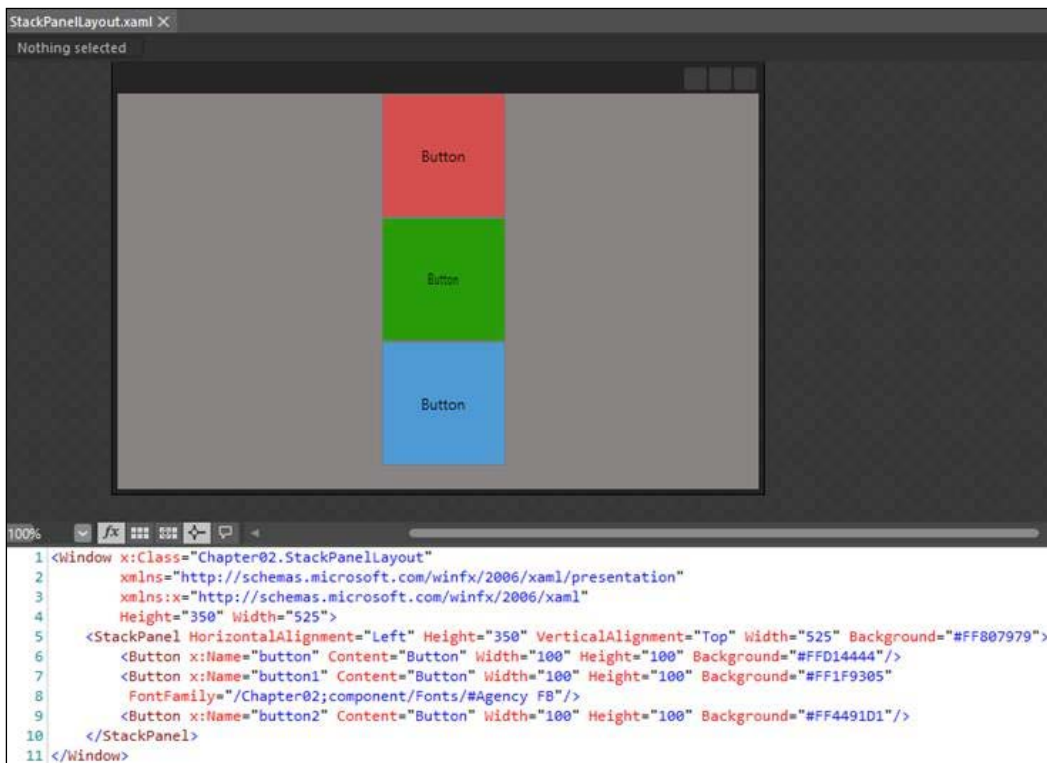
StackPanel

`StackPanel` is used to stack elements one after another either vertically or horizontally.

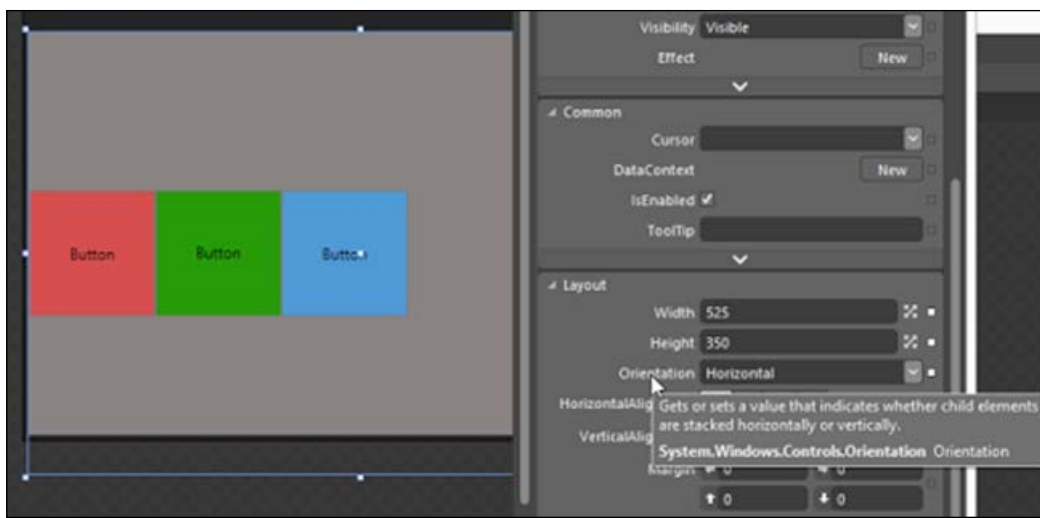
Time for action – using StackPanel

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.

- ◆ **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.
3. DockPanel & Wrap Panel.
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.

3

Working with XAML

XAML is also known as Extensible Application Markup Language. XAML is a generic language like XML and can be used for multiple purposes and in the WPF and Silverlight applications. XAML is majorly used to declaratively design user interfaces.

In the previous chapter, we had a look at the various layout controls and at how to use them in our application. In this chapter, we will have a look at the following topics:

- ◆ The fundamentals of XAML
- ◆ The use of XAML for applications in Blend

An important point to note here is that almost everything that we can do using XAML can also be done using C#. The following is the XAML and C# code to accomplish the same task of adding a rectangle within a canvas. In the XAML code, a canvas is created and an instance of a rectangle is created, which is placed inside the canvas:

- ◆ **XAML code:** In the following code, we declare the `Rectangle` element within `Canvas`, and that makes the `Rectangle` element the child of `Canvas`. The hierarchy of the elements defines the parent-child relationship of the elements. When we declare an element in XAML, it is the same as initializing it with a default constructor, and when we set an attribute in XAML, it is equivalent to setting the same property or event handler in code. In the following code, we set the various properties of `Rectangle`, such as `Height`, `Width`, and so on:

```
<Canvas>
  <Rectangle Height="100" Width="250" Fill="AliceBlue"
    StrokeThickness="5" Stroke="Black" />
</Canvas>
```

- ◆ **C# code:** We created `Canvas` and `Rectangle`. Then, we set a few properties of the `Rectangle` element and then placed `Rectangle` inside `Canvas`:

```
Canvas layoutRoot = new Canvas();
Rectangle rectangle = new Rectangle();

rectangle.Height = 100;
rectangle.Width = 250;
rectangle.Fill = Brushes.AliceBlue;
rectangle.StrokeThickness = 5;
rectangle.Stroke = Brushes.Black;

layoutRoot.Children.Add(rectangle);

AddChild(layoutRoot);
```

We can clearly see why XAML is the preferred choice to define UI elements—XAML code is shorter, more readable, and has all the advantages that a declarative language provides. Another major advantage of working with XAML rather than C# is instant design feedback. As we type XAML code, we see the changes on the art board instantly. Whereas in the case of C#, we have to run the application to see the changes. Thanks to XAML, the process of creating a UI is now more like visual design than code development.

When we drag and drop an element or draw an element on the art board, Blend generates XAML in the background. This is helpful as we do not need to hand-code XAML as we would have to when working with text-editing tools.



Generally, the order of attaching properties and event handlers is performed based on the order in which they are defined in the object element. However, this should not matter, ideally, because, as per the design guidelines, the classes should allow properties and event handlers to be specified in any order. Wherever we create a UI, we should use XAML, and whatever relates to data should be processed in code. XAML is great for UIs that may have logic, but XAML is not intended to process data, which should be prepared in code and posted to the UI for displaying purposes. It is data processing that XAML is not designed for.

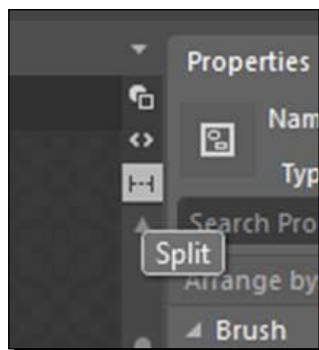
The basics of XAML

Each element in XAML maps to an instance of a .NET class, and the name of the element is exactly the same as the class. For example, the `<Button>` element in XAML is an instruction to create an instance of the `Button` class. XAML specifications define the rules to map the namespaces, types, events, and properties of object-oriented languages to XML namespaces.

Time for action – taking a look at XAML code

Perform the following steps and take a look at the XAML namespaces after creating a WPF application:

1. Let's create a new WPF project in Expression Blend and name it `Chapter03`.
2. In Blend, open `MainWindow.xaml`, and click on the split-view option so that we can see both the design view as well as the XAML view. The following screenshot shows this:



3. You will also notice that a grid is present under the window and there is no element above the window, which means that `Window` is the root element for the current document. We see this in XAML, and we will also see multiple attributes set on `Window`:

```
<Window x:Class="Chapter03.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx
  /2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525">
```

- We can see in the preceding code that the XAML file has a root element. For a XAML document to be valid, there should be one and only one root element. Generally, we have a window, page, or user control as the root element. Other root elements that are used are `ResourceDictionary` for dictionaries and `Application` for application definition.
- The `Window` element also contains a few attributes, including a class name and two XML namespaces. The three properties (`Title`, `Height`, and `Width`) define the caption of the window and default size of the window. The class name, as you might have guessed, defines the class name of the window. The `xmlns` attribute is a reserved attribute in the world of XML to declare namespaces.

- There are two namespaces specified in `MainWindow.xaml` by default. XAML relies on XML namespaces such as these to understand the elements.

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

`http://schemas.microsoft.com/winfx/2006/xaml/presentation` is the default namespace. The default namespace allows us to add elements to the page without specifying any prefix. So, all other namespaces defined must have a unique prefix for reference. For example, the namespace `http://schemas.microsoft.com/winfx/2006/xaml` is defined with the prefix `:x`. The prefix that is mapped to the schema allows us to reference this namespace just using the prefix instead of the full-schema namespace.

- XML namespaces don't have a one-to-one mapping with .NET namespaces. WPF types are spread across multiple namespaces, and one-to-one mapping would be rather cumbersome. So, when we refer to the `presentation`, various namespaces, such as `System.Windows` and `System.Windows.Controls`, are included.
- All the elements and their attributes defined in `MainPage.xaml` should be defined in at least one of the schemas mentioned in the root element of XAML; otherwise, the XAML document will be invalid, and there is no guarantee that the compiler will understand and continue.
- When we add `Rectangle` in XAML, we expect `Rectangle` and its attributes to be part of the default schema:

```
<Rectangle x:Name="someRectangle" Fill="AliceBlue"/>
```

What just happened?

We had a look at the XAML namespaces that we use when we create a WPF application.

Time for action – adding other namespaces in XAML

In this section, we will add another namespace apart from the default namespace:

- 1.** We can use any other namespace in XAML as well. To do that, we need to declare the XML namespaces the schemas of which we want to adhere to. The syntax to do that is as follows:

```
xmlns:Prefix = "clr-namespace:Namespace;assembly=AssemblyName"
```



Prefix is the XML prefix we want to use in our XAML to represent that namespace. For example, the XAML namespace uses the `:x` prefix.

Namespace is the fully qualified .NET namespace.

AssemblyName is the assembly where the type is declared, and this assembly could be the current project assembly or a referenced assembly.

2. Open the XAML view of `MainWindow.xaml` if it is not already open, and add the following line of code after the reference to `xmlns:x`. With this reference, we can access the types in the system namespace using the `system` prefix:

```
xmlns:system="clr-namespace:System;assembly=mscorlib"
To create an instance of an object we would have to use this
namespace prefix as
<system:Double></system:Double>
```

3. We can access the types defined in the current assembly by referencing the namespace of the current project:

```
xmlns:local="clr-namespace:Chapter03"
```

To create an instance of an object we would have to use this namespace prefix as

```
<local:MyObj></local:MyObj>
```

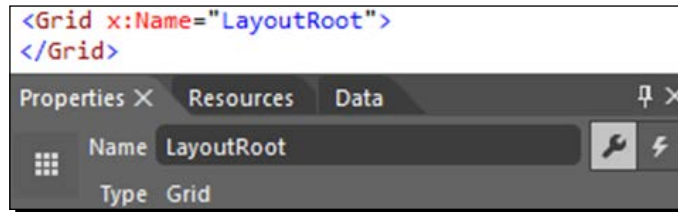
What just happened?

We saw how we can add more namespaces in XAML apart from the ones present by default and how we can use them to create objects.

Naming elements

In XAML, it is not mandatory to add a name to every element, but we might want to name some of the XAML elements that we want to access in the code or XAML. We can change properties or attach the event handler to, or detach it from, elements on the fly.

We can set the name property by hand-coding in XAML or setting it in the properties window. This is shown in the following screenshot:



The code-behind class

The `x:Class` attribute in XAML references the code-behind class for XAML. You would notice the `x:`, which means that the `Class` attribute comes from the XAML namespace, as discussed earlier. The value of the attribute references the `MainWindow` class in the `Chapter03` namespace. If we go to the `MainWindow.xaml.cs` file, we can see the partial class defined there. The code-behind class is where we can put C# (or VB) code for the implementation of event handlers and other application logic. As we discussed earlier, it is technically possible to create all of the XAML elements in code, but that bypasses the advantages of having XAML.

So, as these two are partial classes, they are compiled into one class. So, as C# and XAML are equivalent and both these are partial classes, they are compiled into the same IL.

Time for action – using a named element in a code-behind class

1. Go to `MainWindow.xaml.cs` and change the background of the `LayoutRoot` grid to Green:

```
public MainWindow()  
{  
    InitializeComponent();  
    LayoutRoot.Background = Brushes.Green;  
}
```

2. Run the application; you will see that the background color of the grid is green.

What just happened?

We accessed the element defined in XAML by its name.

Default properties

The content of a XAML element is the value that we can simply put between the tags without any specific references. For example, we can set the content of a button as follows:

```
<Button Content="Some text" /> or <Button > Some text </Button>
```



The default properties are specified in the help file. So, all we need to do is press *F1* on any of our controls to see what the value is.

Expressing properties as attributes

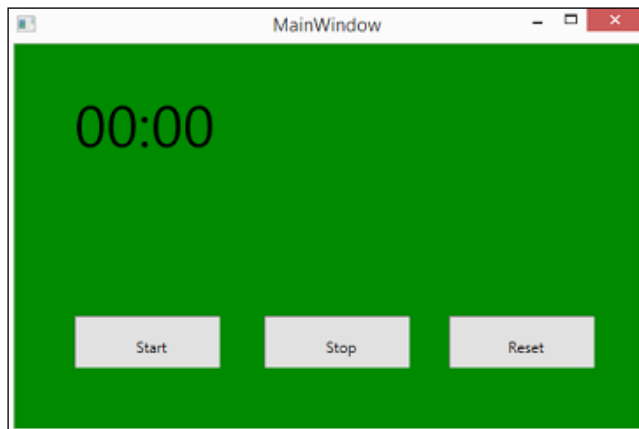
We can express the properties of an element as an XML attribute.

Time for action – adding elements in XAML by hand-coding

In this section, instead of dragging and dropping controls, we will add XAML code:

1. Move `MainWindow.xaml` to XAML and add the code shown here to add `TextBlock` and three buttons in `Grid`:

```
<Grid x:Name="LayoutRoot">
<TextBlock Text="00:00" Height="170" Margin="49,32,38,116"
Width="429" FontSize="48"/>
  <Button Content="Start" Height="50" Margin="49,220,342,49"
Width="125"/>
  <Button Content="Stop" Height="50" Margin="203,220,188,49"
Width="125"/>
  <Button Content="Reset" Height="50" Margin="353,220,38,49"
Width="125"/>
</Grid>
```



2. We set a few properties for each of the elements. The property types of the various properties are also mentioned. These are the .NET types to which the type converter would convert them:
 - **Content:** This is the content displayed onscreen. This property is of the `Object` type.
 - **Height:** This is the height of the button. This property is of the `Double` type.
 - **Width:** This is the width of the button. This property is of the `Double` type.
 - **Margin:** This is the amount of space outside the control, that is, between the edge of the control and its container. This property is of the `Thickness` type.
 - **Text:** This is the text displayed onscreen. This property is of the `String` type.
 - **FontSize:** This is the size of the font of the text. This property is of the `Double` type.

What just happened?

We added elements in XAML with properties as XML attributes.

Non-attribute syntax

We will define a gradient background for the grid, which is a complex property. We already had a look at the `LinearGradient` brush in *Chapter 1, Getting Started with Blend*, and that is what we will use here. Notice that the code in the next section sets the background property of the grid using a different type converter this time. Instead of a string-to-brush converter, the `LinearGradientBrush` to brush type converter would be used.

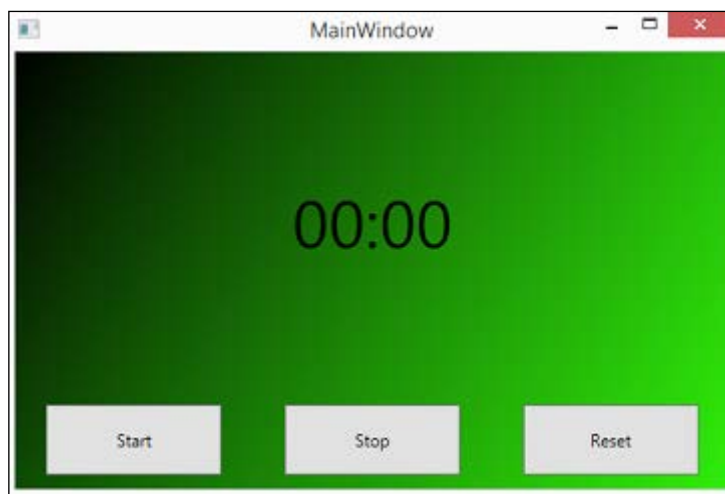
Time for action – defining the gradient for the grid

1. Add the following code inside the grid.
2. We have specified two gradient stops. The first one is black and the second one is a shade of green.
3. We have specified the starting point of `LinearGradientBrush` as the top-left corner and the ending point as the bottom-right corner, so we will see a diagonal gradient:

```
<Grid x:Name="LayoutRoot">
  <Grid.Background>
    <LinearGradientBrush EndPoint="1,1" StartPoint="0,0">
      <GradientStop Color="Black"/>
```

```
<GradientStop Color="#FF27EC07" Offset="1"/>
</LinearGradientBrush>
</Grid.Background>
```

The following is the output of the preceding code:



Comments in XAML

Using `<!-- -->` tags, we can add comments in XAML just as we add comments in XML. Comments are really useful when we have complex XAML with lots of elements and complex layouts:

```
<!-- TextBlock to show the timer -->
```

Styles in XAML

We would like all the buttons in our application to look the same, and we can achieve this using styles. Here, we will just see how styles can be defined and used in XAML, but we will have a look at styles in detail in *Chapter 4, Styles and Templates*.

Defining a style

We will define a style as a static resource in the window. We will go through resources in detail in *Chapter 4, Styles and Templates*. We can move all the properties we define for each button to that style.

Time for action – defining a style in XAML

Add a new `<Window.Resources>` tag in XAML, and then add the code for the style, as shown here:

```
<Window.Resources>
  <Style TargetType="Button" x:Key="MyButtonStyle">
    <Setter Property="Height" Value="50" />
    <Setter Property="Width" Value="125" />
    <Setter Property="Margin" Value="0,10" />
    <Setter Property="FontSize" Value="18"/>
    <Setter Property="FontWeight" Value="Bold" />
    <Setter Property="Background" Value="Black" />
    <Setter Property="Foreground" Value="Green" />
  </Style>
</Window.Resources>
```

We defined a few properties on style that are worth noting:

- ◆ `TargetType`: This property specifies the type of element to which we will apply this style. In our case, it is `Button`.
- ◆ `x:Key`: This is the unique key to reference the style within the window.
- ◆ `Setter`: The setter elements contain the property name and value.

What just happened

We defined a style for a button in the window.

Using a style

Let's use the style that we defined in the previous section. All UI controls have a style property (from the `FrameworkElement` base class).

Time for action – using a style in XAML

1. To use a style, we have to set the style property of the element as shown in the following code. Add the same style code to all the buttons:
 - We set the style property using curly braces (`{ }`) because we are using another element.
 - `StaticResource` denotes that we are using another resource.

- `MyButtonStyle` is the key to refer to the style. The following code encapsulates the style properties:

```
<Button x:Name="BtnStart" Content="Start"
Grid.Row="1" Grid.Column="0"
Style="{StaticResource MyButtonStyle}"/>
```

The following is the output of the preceding code:



What just happened?

We used a style defined for a button.

Where to go from here

This chapter gave a brief overview of XAML, which helps you to start designing your applications in Expression Blend. However, if you wish know more about XAML, you could visit the following MSDN links and go through the various XAML specifications and details:

- ◆ XAML in Silverlight: [http://msdn.microsoft.com/en-us/library/cc189054\(v=vs.95\).aspx](http://msdn.microsoft.com/en-us/library/cc189054(v=vs.95).aspx)
- ◆ XAML in WPF: <http://msdn.microsoft.com/en-us/library/>

Pop quiz

Q1. How can we find out the default property of a control?

1. F1.
2. F2.
3. F3.
4. F4.

Q1. Is it possible to create a custom type converter?

1. Yes.
2. No.

Summary

We had a look at the basics of XAML, including namespaces, elements, and properties. With this introduction, you can now hand-edit XAML, where necessary, and this will allow you to tweak the output of Expression Blend or even hand-code an entire UI if you are more comfortable with that.

In the next chapter, we will have a look at styles and templates.

4

Styles and Templates

A style is a group of property settings that determines how a control will appear. We see styles in almost all of the applications. In WPF and Silverlight, styles are nothing more than collections of property setters, and, most commonly, these properties will be the ones that relate to the appearance of the objects.

In the previous chapter, we talked about XAML, and now, in this chapter, we will have a look at styles and templates.

In any type of application, styles bring consistency across the whole application. We can very easily create new styles, modify the existing ones, or create a style based on the built-in styles.

Creating and using styles

We can design the user interfaces for our application with Blend using the default or system controls. We can style these controls to our liking and make them behave the way we want using styles and templates in WPF and Silverlight.

A style, in simple terms, can be a set of property values that can be shared across multiple instances of the same or different elements. Using a style, we can do the following:

- ◆ Modify the default values of the properties of an element, such as specifying the background, height, width, and so on
- ◆ Specify a default behavior of the element, such as defining a trigger so that when the mouse enters the element, the background color of the element changes

A style looks somewhat similar to the following code; we will have a look in detail at how we can create, modify, and use styles in this chapter:

```
<Style x:Key="ButtonStyle1" TargetType="{x:Type Control}">
  <Style.Setters>
    <Setter Property="Height" Value="50"/>
    <Setter Property="Width" Value="200"/>
    <Setter Property="Background" Value="Green"/>
    <Setter Property="Foreground" Value="White"/>
  </Style.Setters>
</Style>
```

An introduction to styles

A resource is a reusable value. It can be a brush, a gradient, or even an animation. It is a markup extension that could have a key (not mandatory). The key is the item used to access a resource at the markup or code level. We could apply a resource to a single element, such as a button, everything in a window, or the entire application, depending on the scope at which the resource is defined. We can define a resource at:

- ◆ **The application level:** The resource is accessible to the elements in the application
- ◆ **The document level:** The resource is accessible to all the elements in the window, page, or user control
- ◆ **The element level:** The resource is accessible only to that element and its child elements

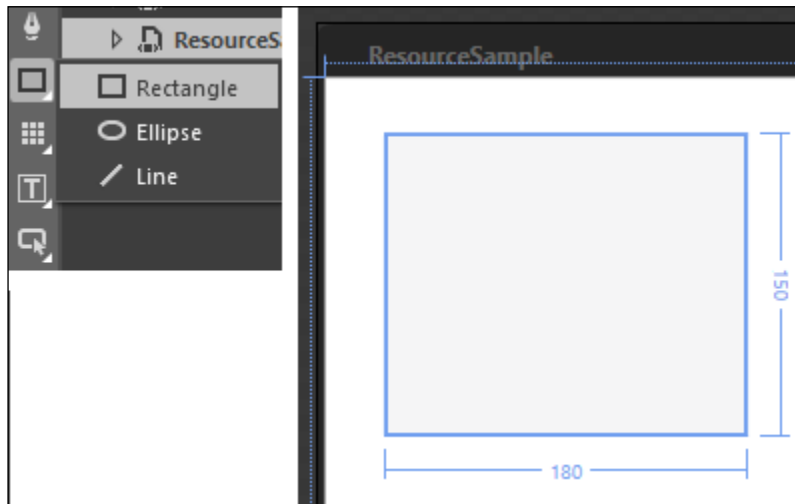
WPF looks for the resource first at the element level, then at the parent level, and finally, at the application level. It is a good practice to have a unique key defined for each resource; however, if the key is not unique, then the first resource encountered with the key is used.

There are two types of resources:

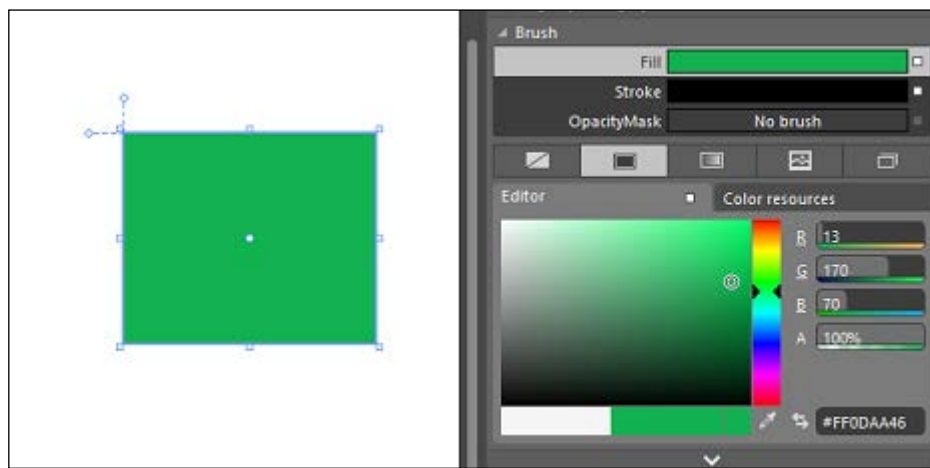
- ◆ **Static resources:** These resources are retrieved once and then used for the lifetime of the resource.
- ◆ **Dynamic resources:** These resources are retrieved every time they are used. Dynamic resources are loaded every time the resource changes and also track the changes in the resource at runtime. Dynamic resources are not loaded until they are used. A dynamic resource does not need to be defined before referencing.

Time for action – creating a resource

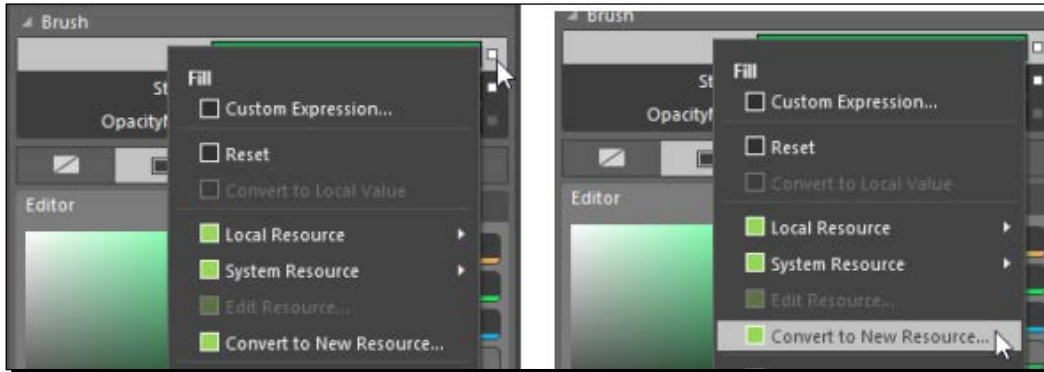
1. Add a new window to the project and name it `ResourceSample.xaml`.
2. After you click on **OK**, select **Rectangle** from the **Tools** panel, and draw a rectangle onto the art board. This is shown in the following screenshot:



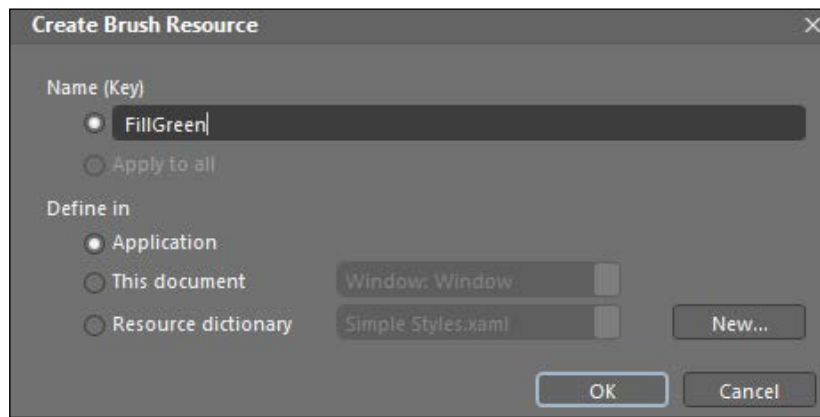
3. Now, add **Fill** to the rectangle from the **Properties** panel, as shown here:



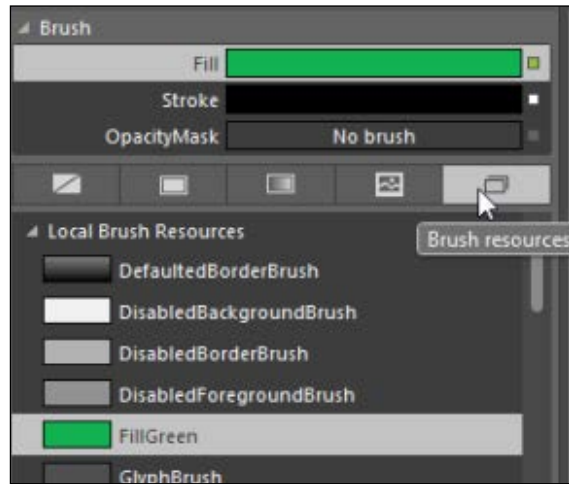
4. Click on the white square at the end of the **Fill** property and select **Convert to New Resource...**. This is shown in the following screenshot:



5. This will present us with an option to create a new resource. We will select the scope of the resource as **Application**, name the resource `FillGreen`, and click on **OK**. The following screenshot depicts this:



- Now, if we take a look at the **Brush resources** available to us, we will see that the new resource we created is also available to us, as shown here:



What just happened?

We created a resource in Blender.

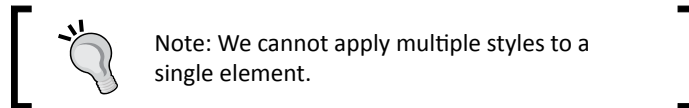
The resource dictionary

A resource dictionary is a collection of all the resources (styles, templates, animations, and so on) that can be used within the application. Resource dictionaries can be merged. Resources can't be merged.

We will work with a resource dictionary later in the chapter.

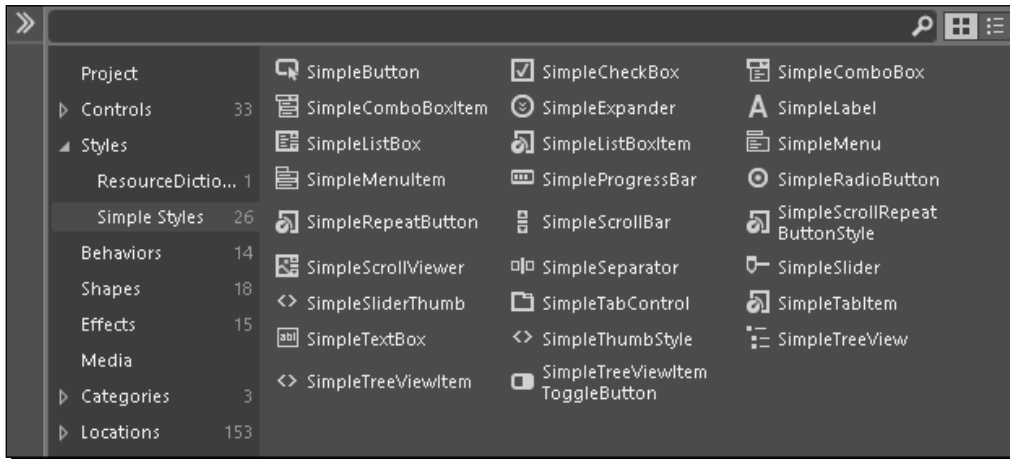
Simple styles

Simple styles are available in Blend for a set of common system controls (button, label, and so on). The reason why Blend provides this functionality is that it helps users who have little or no knowledge of WPF styles and templates and who might break the functionality of the controls while modifying the control. This set of resources could be sufficient in certain cases to give a unique look and feel to our application.



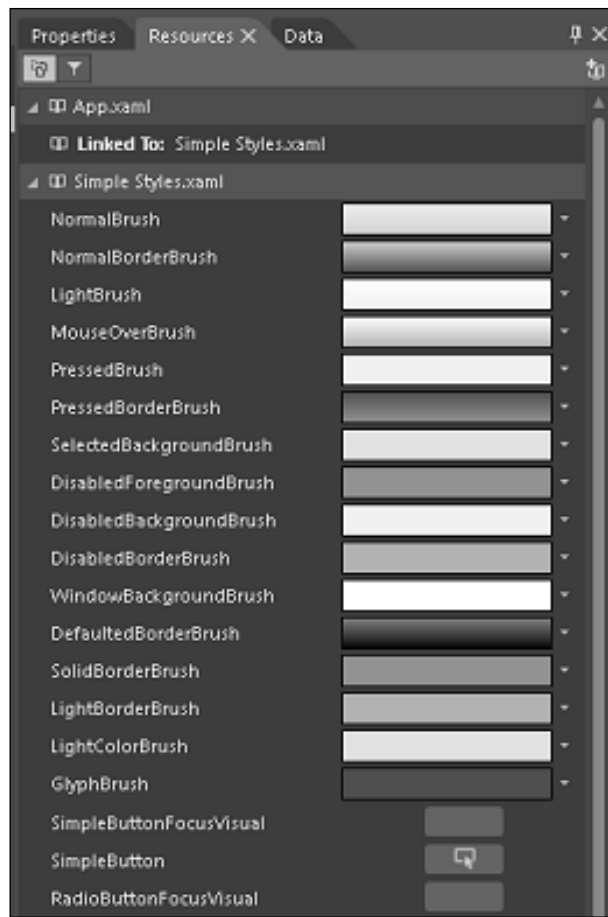
Creating a simple styled control

We can access **Simple Styles** in the **Assets** panel by clicking on **Simple Styles** in the **Styles** category. To use these styles, we simply need to drag and drop the simple style onto the art board, and an instance of the control is created and the style is applied to it. This is illustrated in much detail in the following screenshot:



Let's drag and drop **SimpleButton** onto the art board. When we do that, a couple of things will happen:

- ◆ An instance of the button control is added onto the art board with a simple style (obviously).
- ◆ A resource dictionary (**Simple Styles.xaml**) is added to the project that contains the styles for all the simple styles, so all the resources are available under the **Resources** panel. This is shown in the following screenshot:

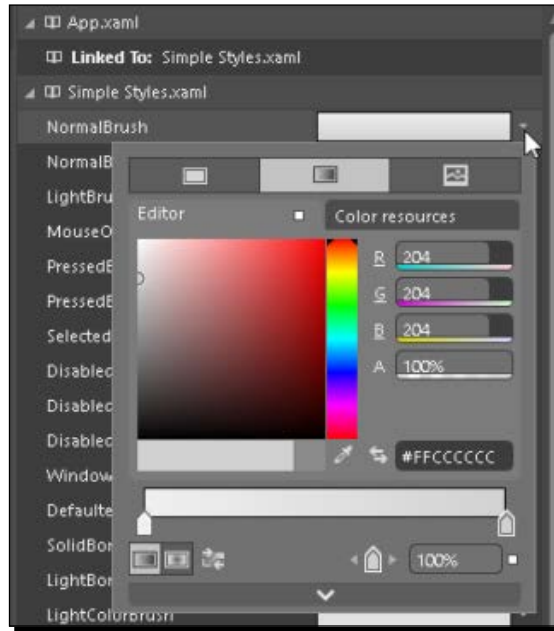


Changing colors

We can change the colors used by the styles. To do that, we need to click on the down arrow next to the color resource in the **Resources** panel and select the brush we want.

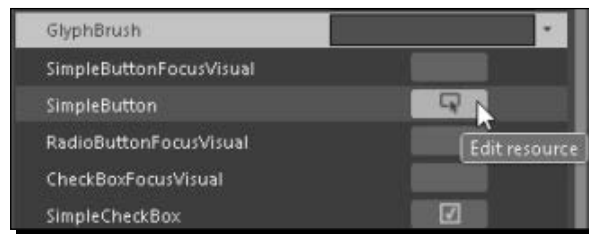


Making a change to a style is different than making a change to a control as changing a style changes every instance of the control to which the style is applied, whereas changing a control only applies the changes to the changed control.



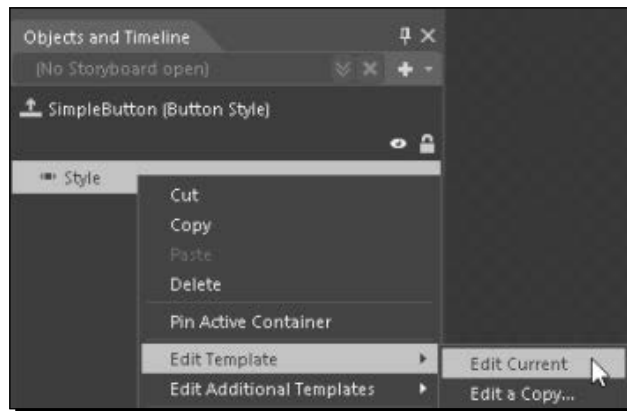
Changing styles

We use style in Blender to specify the properties that are used as defaults by the control on which the style is applied. The properties affect the appearance of the controls. To change any of the simple styles in our resources, we need to click on the **Edit resource** button next to the simple style that we want to edit. Take a look at the following screenshot that shows this in detail:



Changing control templates

The control template defines the appearance of the control. To change a control template, click on **Edit resource** as before, move to the **Objects and Timeline** panel, right-click the style element, move to **Edit Template**, and click on **Edit Current**. This step is shown in intricate detail in the following screenshot:



We can apply these styles and templates to other controls of the same or inherited type as well.

Style specification

The behavior of a style may be different depending on the properties we define in the style.

Specifying TargetType of a style

`TargetType` specifies the type of the element on which the style is targeted. This type of specification is generally used in places where we want all the instances of a particular element to pick the styles by default. In this type of specification, we specify a style with a type rather than a key so that it applies to all the elements of a particular type. We have already seen this in action in the preceding section, where we defined the button style with the scope to apply to all. We defined a style with `TargetType` as `Button`, and the buttons defined in the application will have style applied to them automatically. The following code shows this:

```
<Style TargetType="Button">
    Or
<Style TargetType="{x:Type Button}">
```

There is no difference between these two as WPF internal conversion converts `Button` to `System.Type`, which is a button, and, hence, is equivalent to specifying `x:Type Button`. But it is recommended that you use the explicit specification (`x:Type`).

This approach works because the style will generate a self-key and use the target type that we have specified as a key (`- * +`).

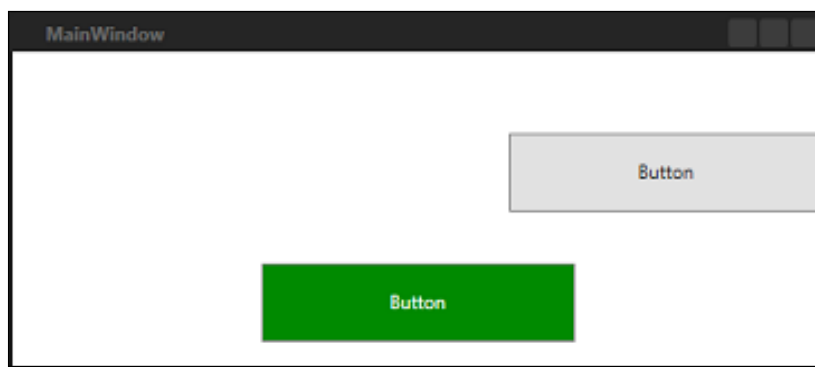
Specifying the key for a style

We can specify the key for a style so that it can be explicitly referenced in the elements to apply the style. When we define a style with a key, that style will not be applied to the elements by default. Along with the key, we also need to specify `TargetType` of the element we want to apply the style to. This will allow us to set the properties specific to that element rather than setting the generic properties of the `FrameworkElement` class. The `FrameworkElement` class is the base class from which all the elements inherit. This is amply illustrated in the following code and depicted well in the screenshot that follows the code:

```
<Grid>
  <Grid.Resources>
    <Style x:Key="ButtonStyle1" TargetType="{x:Type Button}">
      <Style.Setters>
        <Setter Property="Height" Value="50"/>
      </Style.Setters>
    </Style>
  </Grid.Resources>
</Grid>
```

```
<Setter Property="Width" Value="200"/>
<Setter Property="Background" Value="Green"/>
<Setter Property="Foreground" Value="White"/>
<Setter Property="HorizontalAlignment" Value="Center"/>
<Setter Property="VerticalAlignment" Value="Center"/>
</Style.Setters>
</Style>
</Grid.Resources>

<Button Content="Button" Style="{StaticResource ButtonStyle1}" />
<Button Content="Button" />
</Grid>
```

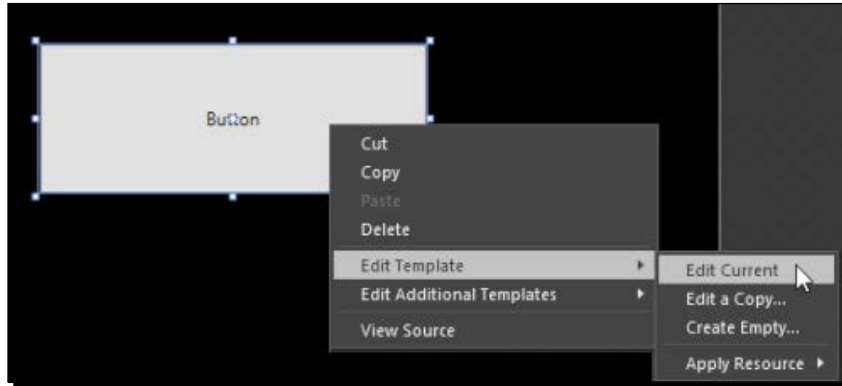


Application skinning

Application skinning refers to changing the visual appearance and behavior of the UI elements in our application. Multiple resources can be stored in a resource dictionary file and then added to multiple projects. By doing this, we can define in a resource dictionary a theme for our application by storing color resources, styles, and templates for common controls. We can have multiple resource dictionaries for the application, and we can choose one dictionary to have a particular style for the application. We can load the styles we want to apply from the merged dictionary collection. We can provide skinning at runtime by loading these dictionaries at runtime.

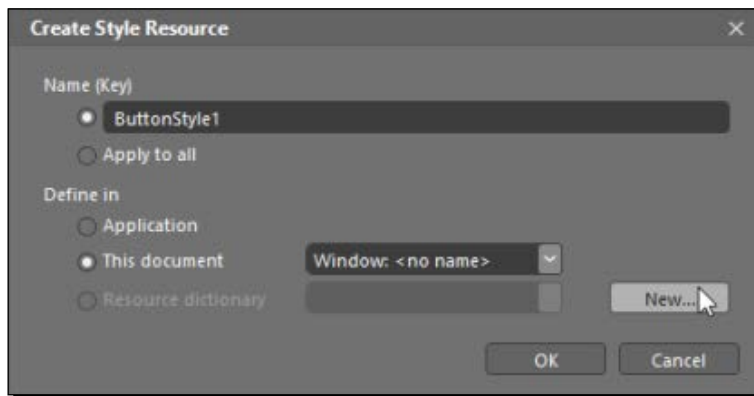
Time for action – creating resource dictionaries

1. Let's drag and drop a button onto the art board. Now right-click on **Button**, and then click on **Edit Template | Edit a Copy...** This is shown in the following screenshot:

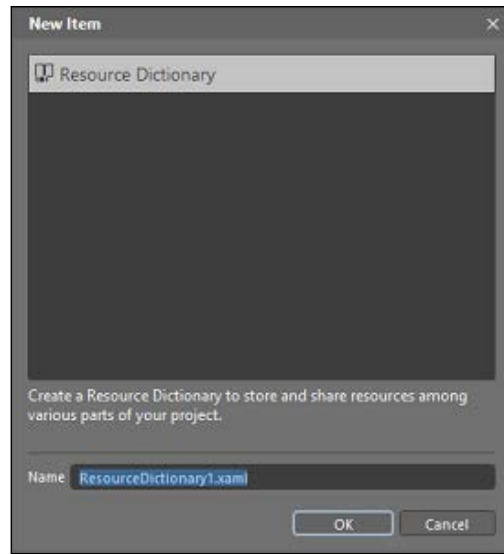


Once we do that, we will see a new window opened that has the option to provide a name for the style resource we are creating and also the option to select the location where this style resource will be placed. However, the resource dictionary section is disabled because there is no resource dictionary present at this moment.

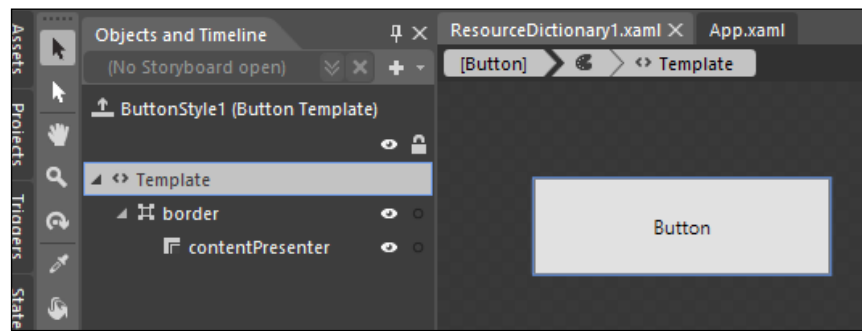
2. So, we will go ahead and create a new resource dictionary by clicking on the **New** button next to **Resource dictionary**. This is precisely shown in the following screenshot:



3. Once we do that, we have an option to provide a name for the resource dictionary. Once we click on **OK**, we move back to the **Create Style Resource** screen. This action is described in the following screenshot:



4. Once we click on OK on the **Create Style Resource** screen, we see that we have moved to `ResourceDictionary1.xaml`. And we are editing **Button Template**. This is encapsulated in this screenshot:



Templates

A template defines the hierarchy of the elements as well as their styles in a control. Using templates, we can modify the structure of the control to which the template is applied. We have two types of templates available in WPF and Silverlight that are as follows:

- ◆ **The control template:** It defines the appearance and structure of the control. For example, we can define the control template of a button to look like a circle but still have its click and other behaviors unaltered.
- ◆ **The data template:** This template specifies a group of characteristics for how data should be displayed. This template is particularly useful when you bind ItemsControl, such as ListBox, to an entire collection. We will have a look at data templates in *Chapter 9, User Controls and Custom Controls*.

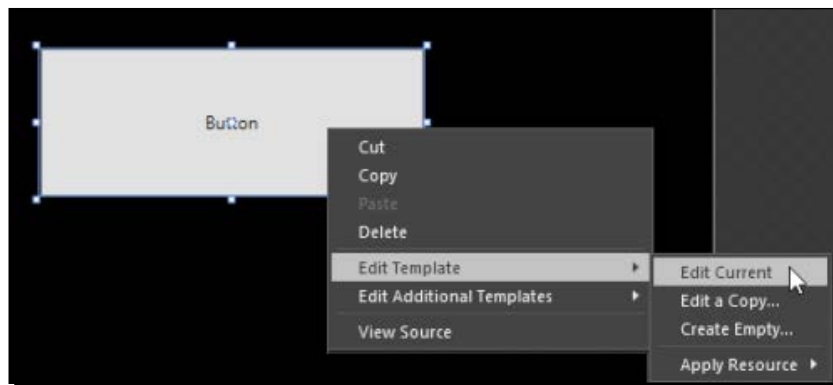
Editing the template

Templates are defined as a way to reuse the structure of objects or elements. We can also add behaviors to a template; this is explained in detail in *Chapter 5, Behaviors and States in Blend*. Templates are resources and can be defined at the application, window, or element level just as in the case of any other resource. Wherever we use the template, a new copy of the template is generated. So, actually, a template behaves like a factory that creates a copy of the UI tree wherever it is used. This is really useful when we do DataBinding or control customization. Once we have created a template, we have the option to modify the template as well.

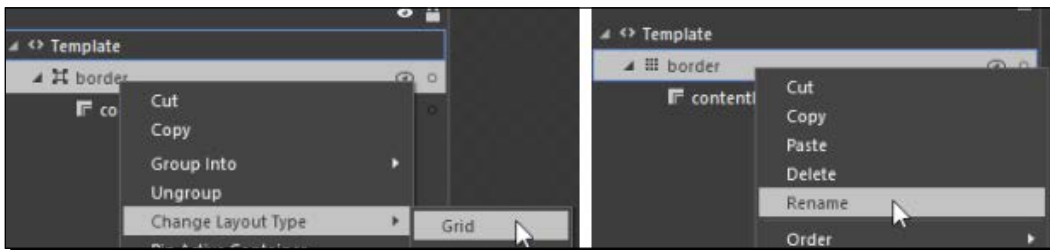
Time for action – editing the template

To edit the template of a button, perform the following steps:

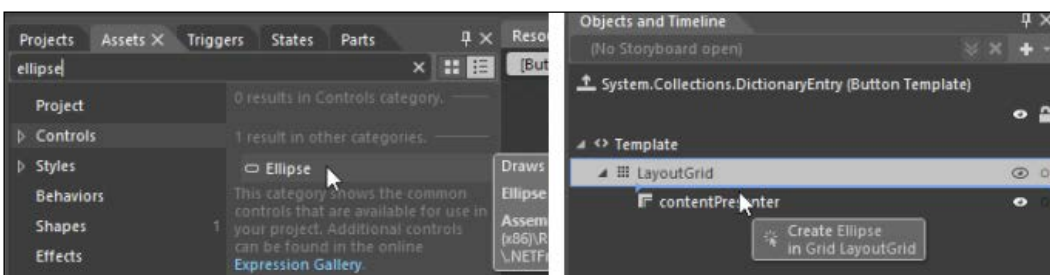
1. To edit the template of a button, we need to right-click on the button and select **Edit Template | Edit Current**. This action is described in great detail in the following screenshot:



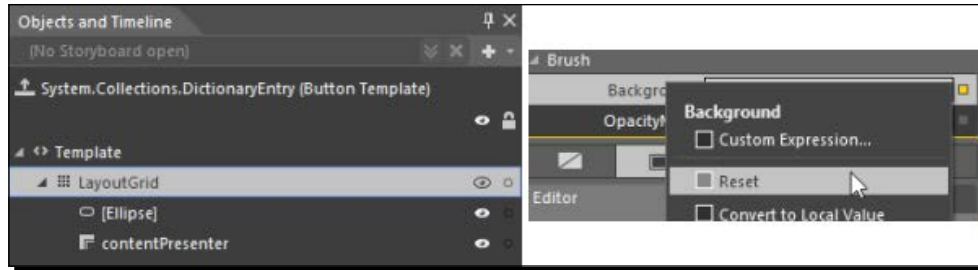
2. We will see the components of the template. We also notice that the presentation of the button is made up of a border with a content presenter in it.
3. **Change Layout Type** from **Border** to **Grid** and rename **Border** to `LayoutGrid`. This is shown in the following screenshot:



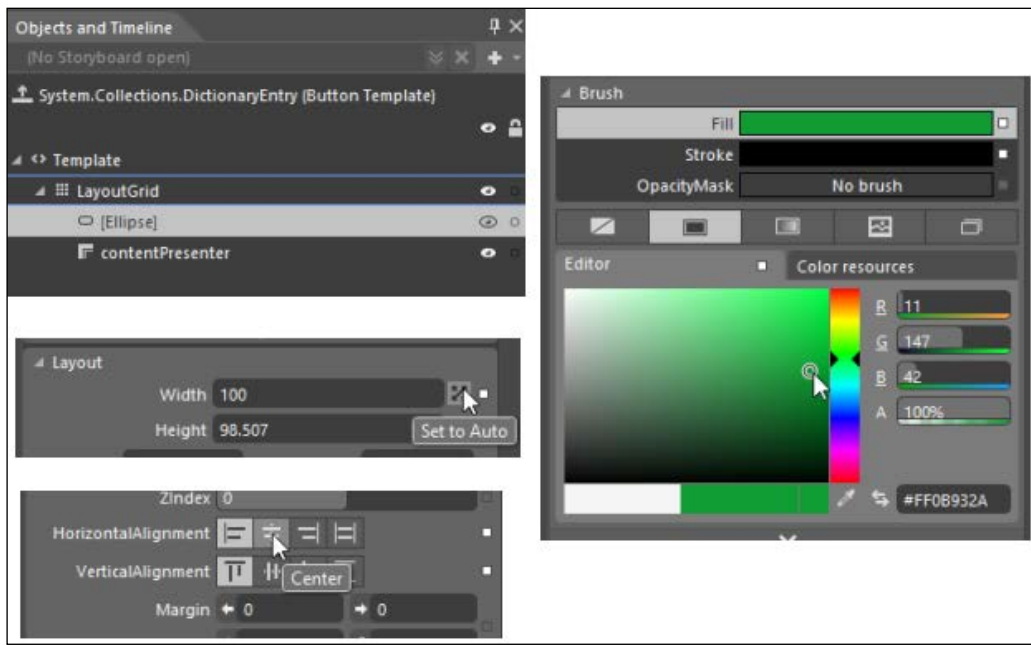
4. Go to the **Assets** panel, search for **Ellipse**, and select and drag **Ellipse** in the grid above `contentPresenter`. This is depicted in the following screenshot:



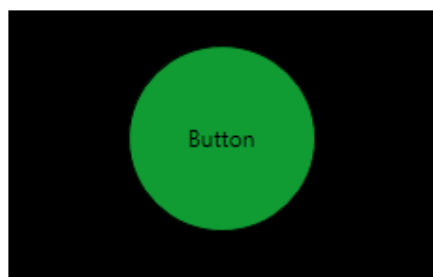
5. Select **LayoutGrid** in the **Objects and Timeline** panel and **Reset** the **Background** property in the **Property** panel. This is amply conveyed to you by the following screenshot:



6. Select **Ellipse** in the **Objects and Timeline** panel, and then change the following properties:
 - Set **Height** and **Width** of **Ellipse** to auto
 - Set **HorizontalAlignment** and **VerticalAlignment** of **Ellipse** to **Center**
 - Set **Fill** of **Ellipse** to a shade of green



7. Now, when we look at the button, it doesn't look similar to what it looked like before. The following screenshot shows the button as it looks like now:



8. Now, this template is present in `ResourceDictionary1.xaml` and available at the application level. This is the default template applied to every button unless we specify otherwise since we did not assign a key to the template.

What just happened?

We edited a template and then used it with a **Button** instance.

Merged dictionaries

If we move our focus back to resource dictionaries, we will see that we can have multiple resource dictionaries that can be merged to provide a themed application. Here is a snapshot of the `App.xaml` file after the creation of `ResourceDictionary1.xaml`:

```
<Application x:Class="StylesTemplates.App"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  StartupUri="StyleTriggers.xaml">
  <Application.Resources>
    <ResourceDictionary>
      <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary Source="ResourceDictionary1.xaml"/>
      </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
  </Application.Resources>
</Application>
```

Using `ResourceDictionaries`, we can provide skinning to an application and use the resources from `ResourceDictionary` in our application.

One important point to note about resource dictionaries is that we can have multiple resources with the same key in different resource dictionaries. When the system is looking for a particular resource in the resource dictionaries, it stops the search at the moment it encounters the first instance of that resource. This may be the reason why a style does not behave the way we expect it to.



An error will be raised if `StaticResource` is not found anywhere in the application.

Pop quiz

Q1. How do I apply a style to only one button?

1. `<Style TargetType="{x:Type Button}" x:Key="AButtonStyle"> </Style>`.
2. `Style TargetType="{x:Type Button}"> </Style>`.
3. `<Style x:Key="AButtonStyle"> </Style>`.

Summary

In this chapter, we had a look at customizing the look of a control using styles and templates.

5

Behaviors and States in Blend

Behaviors are reusable code packages that can be added to any object and then fine-tuned by changing their properties. They let us capture, and play with, events in our XAML without the need to write any code.

In the previous chapter, we had a look at styles and templates in Blend and saw how we could create, modify, and use styles and templates. In this chapter, we will have a look at the following topics:

- ◆ Behaviors
- ◆ Visual states
- ◆ Visual State Manager

An introduction to behavior objects

We can add interactivity to applications in many ways, and behaviors are one of them. By using behaviors, we can move out of the limited scope of *storyboards* and design our applications to respond to users using behaviors available with Blend. We can record the behavior of an object in XAML and then start, play, or pause it. Using behavior, we can modify how the element inherently behaves and how it responds to user actions.



Storyboards are containers to hold animation information. We will have a look at storyboards in detail in *Chapter 6, Understanding Animation and Storyboards*.

Adding built-in behaviors

There are multiple built-in behaviors available in Blend that can be used simply by dragging and dropping onto the element on which we want the behavior. Some behaviors work upon simply dragging and dropping, while others would require property modifications before they can be enabled.

Types of built-in behaviors

There are the following types of built-in behaviors:

- ◆ **Animation behaviors:** We can apply these behaviors to a storyboard or to a transition animation so that it appears smooth.
- ◆ **Conditional behaviors:** We can use conditional behaviors to link an action to an event, the condition of which evaluates to true. We can apply these conditions by modifying properties in the **Properties** panel.
- ◆ **Data behaviors:** We can use these behaviors to interact with data in multiple ways—adding and modifying properties using a data store, firing different actions based on changes in the data store, and applying visual state changes based on the data store.
- ◆ **Motion behaviors:** We can use these behaviors to allow users to control the movement of elements onscreen. The motion behaviors available in WPF are `MouseDownElementBehavior` and `TranslateZoomRotateBehavior`. We will have a look at these later in the chapter.

Animation behaviors

These are the behaviors that we can apply to an animation for it to appear smoother.

Time for action – adding a storyboard

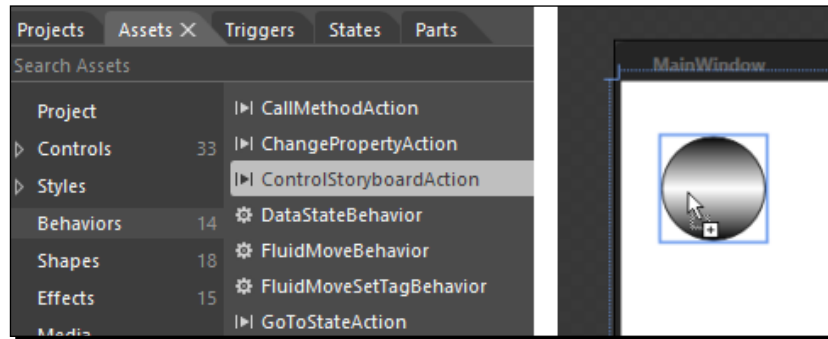
Let's use built-in behaviors. We will make use of the same project that we created in the last chapter and add behaviors to it. Navigate to `Chapter5\Before\Chapter05` in the code bundle of this book.

ControlStoryboardAction

Perform the following steps to use the **ControlStoryboardAction** behavior:

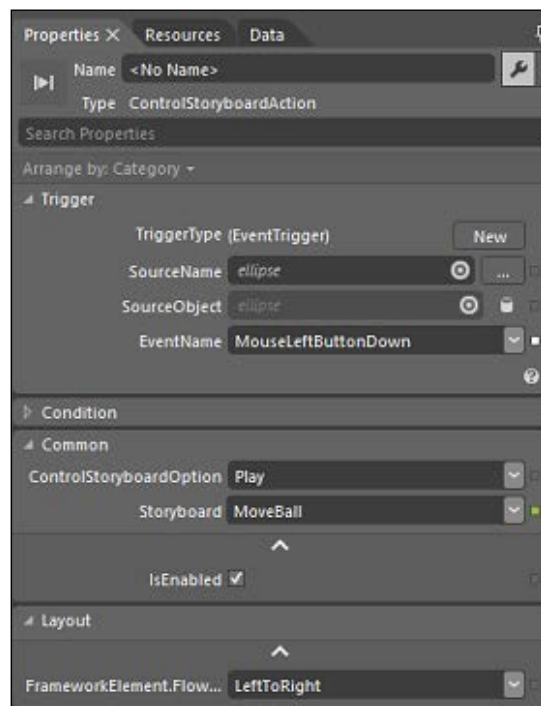
- 1.** Go to the **Assets** panel and select **Behaviors**.
- 2.** We can see the list of available behaviors. We will have a look at a few of these in detail, and the others should be obvious after that.

3. Select the **ControlStoryboardAction** behavior and drag it onto the ball. The following is the output of these actions:



Once we have added the behavior to the ball, we see a couple of things.

4. We see the **Objects and Timeline** panel, and, when we expand the ellipse, we can see that a **ControlStoryboardAction** behavior is added to the ellipse.
5. We see some changes in the **Properties** panel as it now displays the various options to configure the **ControlStoryboardAction** behavior we just added. The following screenshot shows this:

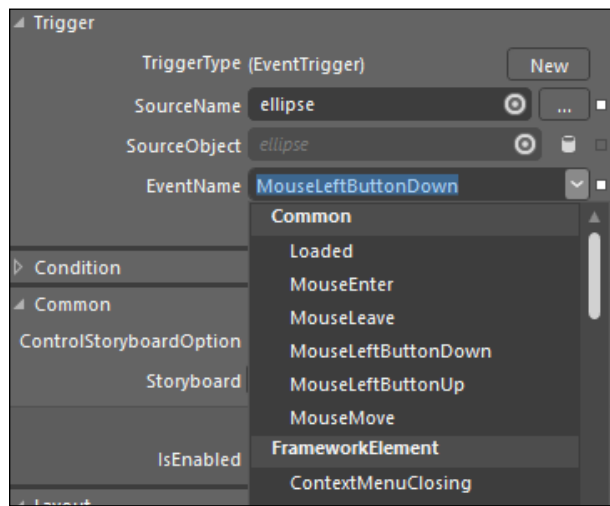


A few of the options available to configure are as follows:

- ◆ The first option that we see is **SourceName**. This is the property that lets us select the element, the events of which would trigger the storyboard. This element could be same as the element that has the storyboard, or it could be any other element. When we click on the ... (browse) button, Blend will show a popup to select the elements, the events of which we could use as a trigger to perform operations on the storyboard. Alternatively, if we click on the circular icon before the browse button, Blend lets us choose the element from the art board just by clicking on it. This is shown by the following screenshot:



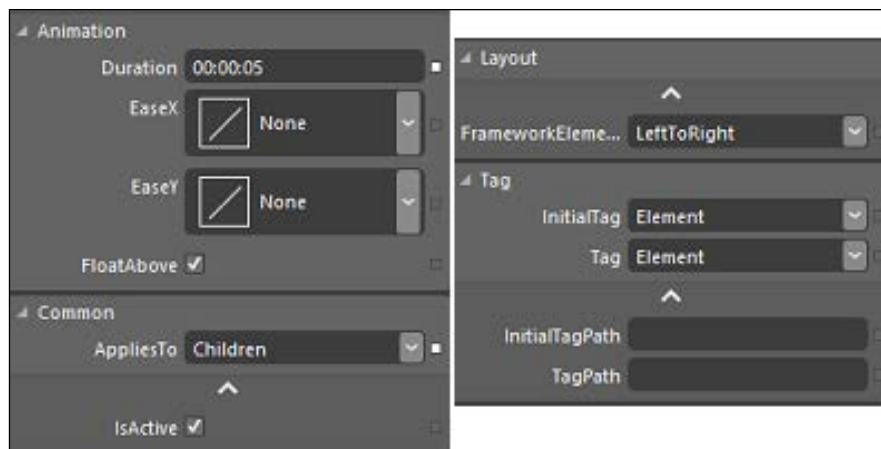
- ◆ The next item that we see is **SourceObject**. This is actually used for DataBinding. If we click the circular button, we can choose the data source; otherwise, we could click on the database button to configure the DataBinding for `EventTrigger`.
- ◆ The next option that we see lets us select **EventName** on which we want our storyboard action to be performed. This will contain a list of events that are available for the source element we selected, including base class events. The following screenshot shows this:



- ◆ The next option that we see is **Condition**. This is the place where we can add certain conditions that match the DataBinding.
- ◆ In the next section, we find the various options to control the storyboard. Also, we have the option to enable or disable the behavior by checking or unchecking the **IsEnabled** checkbox.

FluidMoveBehavior

This type of behavior is particularly useful when we want to smoothly move an element from one position to another. The **Properties** panel of `FluidMovePanel` looks as shown in the next screenshot. We can set the time limit for the behavior, easing functions, and so on. We could also configure it to say whether the behavior is applicable to the element itself or its children. This type of behavior is useful in situations where we are adding or removing elements from a panel or changing the position of an element.



FluidMoveSetTagBehavior

This behavior is generally used to write information to a data store that is used by `FluidMoveBehavior`.

Conditional behaviors

The following are the different types of conditional behaviors:

- ◆ `CallMethodAction`: We can use this behavior to call a method on `DataContext` of an element when an event occurs
- ◆ `ChangePropertyAction`: We can use this behavior to change the property of an object

- ◆ `ControlStoryboardAction`: We can use this behavior to control the state of a storyboard by specifying various actions, such as play, pause, or stopped
- ◆ `GoToStateAction`: We can use this behavior to enable or show a particular visual state
- ◆ `HyperlinkAction`: We can use this action to browse through a website address
- ◆ `InvokeCommandAction`: We can use this behavior to invoke a command that is available in the data source
- ◆ `LaunchUriOrFileAction`: We can use this behavior to browse through a website or launch an application
- ◆ `PlaySoundAction`: We can use this behavior to play a sound file
- ◆ `RemoveElementAction`: We can use this behavior to remove an element from the logical tree
- ◆ `SetDataStoreValueAction`: We can use this behavior to automatically adjust the values of our data store at runtime

Data state behaviors

The following are the different types of data state behaviors:

- ◆ `CallMethodAction`: We can use this behavior to call a method on `DataContext` of an element when an event occurs.
- ◆ `DataStateBehavior`: We can use this behavior to change the visual of an object based on the state of the data that it is bound to. The data state evaluates to true or false, which determines whether the element goes to that state or not.
- ◆ `FluidMoveSetTagBehavior`: This behavior is generally used to write information to a data store that is used by the `FluidMoveBehavior`.
- ◆ `InvokeCommandAction`: We can use this behavior to invoke a command that is available in the data source.
- ◆ `SetDataStoreValueAction`: We can use this behavior to automatically adjust the values of our data store at runtime.

Motion behaviors

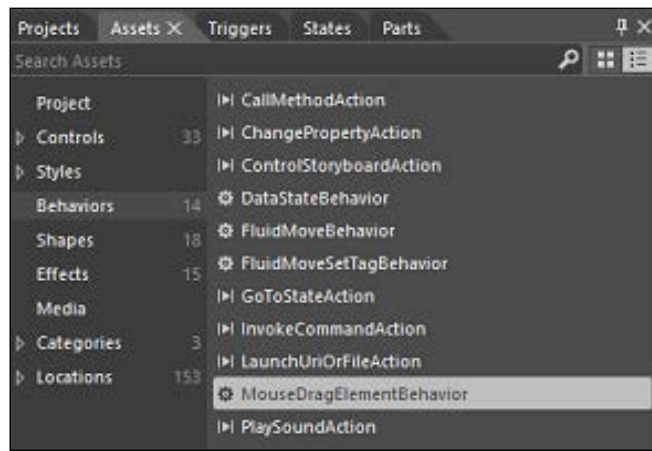
We can use motion behaviors to affect the movements of elements onscreen.

MouseDownElementBehavior

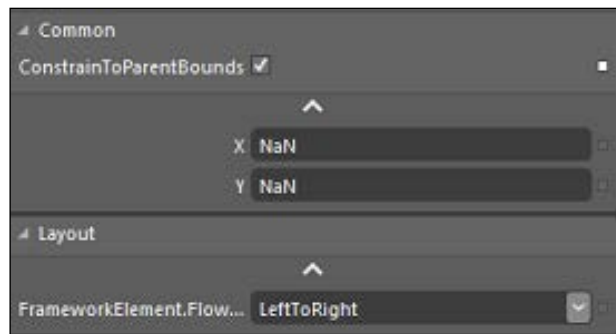
We can use this behavior to enable the user to drag the element within or outside the bounds of the application. To see this behavior in action, we need to disable the `ControlStoryboardAction` behavior. We can do that by unchecking the **IsEnabled** checkbox in the properties of `ControlStoryboardAction`.

TranslateZoomRotateBehavior

This behavior is available only to touchscreens. We can use this behavior to translate, rotate, and scale an element. From the **Assets** panel, select and drag `MouseDragElementBehavior` onto the ball. This is shown in the following screenshot:



Once we do that, we can see a set of properties, as shown in the following screenshot. The important property to note here is `ConstrainToParentBounds`. If this property is enabled, we will not be able to drag the element outside the boundaries of the element's parent, which, in this case, is the grid layout.



Visual states

For predefined visual states in the inbuilt controls, there already exists a mechanism to switch states. For visual states that we create on our own, we will have to provide the logic to switch the visual states. We can define a different visual appearance for each visual state our controls can have. These controls can be user control, window, page, control template, or any of their subclasses, such as `button`.

Visual states are grouped into state groups, which are instances of `VisualStateGroup`. In each state group, it is mandatory to have a default state. A state group contains any visual states that are part of the same logical category, and that cannot be displayed at the same time, whereas the states contained in one group are independent of the states of other groups. This gives us the flexibility of applying any state present in different state groups at any point in time. The only thing that we need to be careful of is the states we are applying at the same time, multiple states should not try to change the same property of an object at the same time. This conflict is detected by Blend within a state group and is notified by displaying a warning icon next to the conflicting state.

The following are the three stages in designing visual states:

- ◆ **Static stage:** This is the stage in which we design the various visual states that we want our control to be in. In this stage, we do not think about anything else, such as the transition between these states.
- ◆ **Transitions stage:** At this stage, we can add transition effects between visual states. We can also configure the various easing effects that are available.
- ◆ **Dynamic stage:** This is the stage in which we can have in-state animations. These are the animations that we add within a state action.

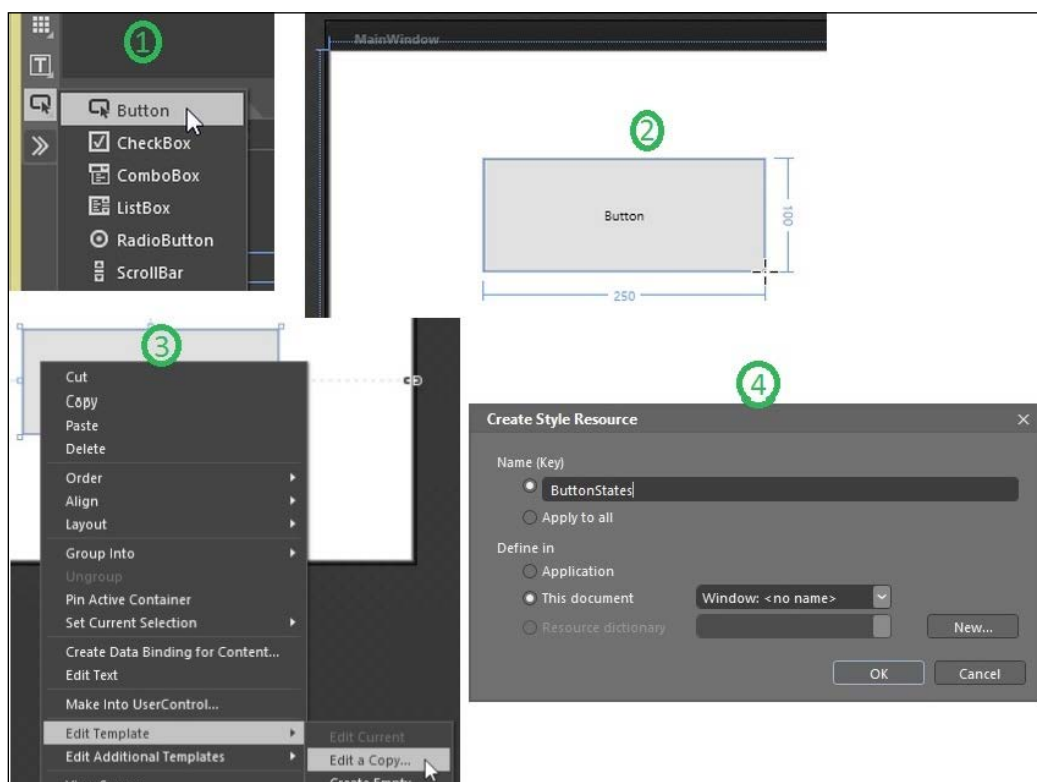
Visual State Manager

Visual State Manager, as its name implies, helps us manage the visual states of controls. It is a simple and powerful means to provide state transitions to controls, while hiding a lot of the animation mechanisms from them. Visual State Manager helps us define the appearance of a control based on the user interaction. Every control has predefined states, and Visual State Manager manages the logic of transitioning between these predefined states. It also allows us to specify the states of various controls. We can have a look at the `VisualStateManager` class in detail at [http://msdn.microsoft.com/en-us/library/system.windows.visualstatemanager\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/system.windows.visualstatemanager(v=vs.110).aspx).

Time for action – modifying with visual states

A button in WPF has default visual states, and we see these different visual states when we interact with the button by clicking on it or by hovering the mouse over it. We will modify these states, add a new state, and see the transition between these states on user interaction.

1. Create a new project in Blend and name it `VSMDemo`. Add a button to the art board. Now right-click on the button, navigate to **Edit Template | Edit a Copy...**, and you would see a dialog box prompting you to give a name to this resource. Give a name and click on **Ok**. The following screenshot shows this:



2. Select the **States** panel, and you will see the default states of the button. There are three default states:

□ **CommonStates**

Normal: This is the state of the button when no interaction is happening with the button.

MouseOver: This is the state when the mouse rolls over the button

Pressed: This is the state when the button is clicked on

Disabled: This is the state when the button is disabled

□ **FocusStates**

Unfocused: This is the state when the button is not in focus

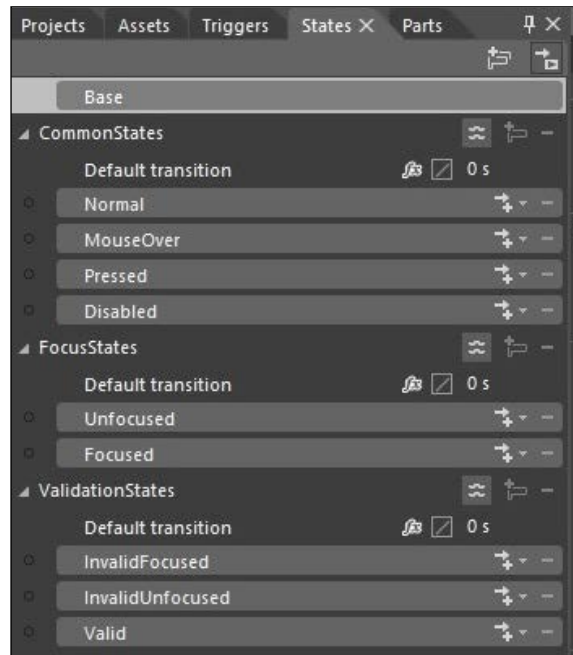
Focused: This is the state when the button is in focus

□ **ValidationStates**

InvalidFocused: This is the state when the button is focused and the validation fails

InvalidUnfocused: This is the state when the validation fails and the button is not in focus

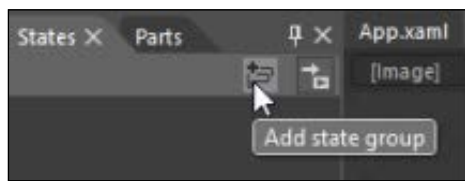
Valid: This is the state when the validation passes



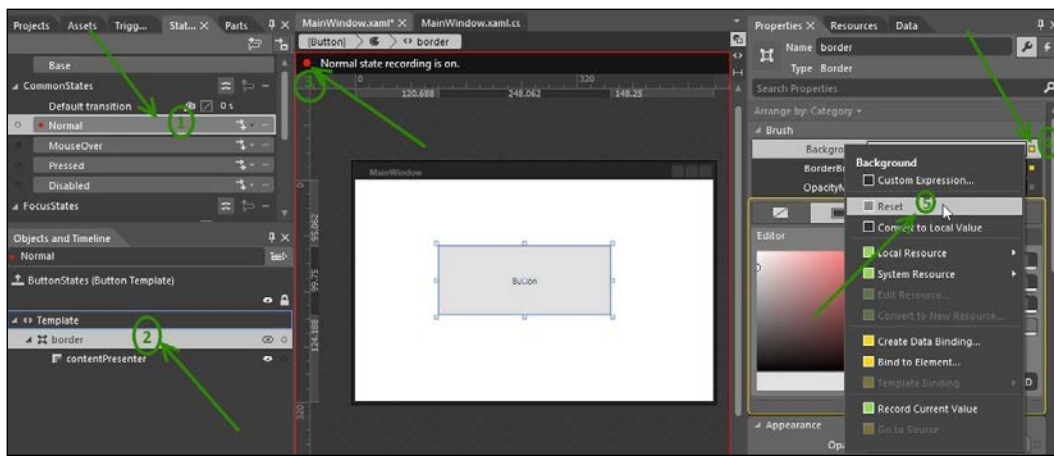


The state groups are mutually exclusive groups of states that a control can be in at the same time. As in the preceding sample, the button could be in the **Normal** and **Unfocused** states, but it cannot be in the **Normal** and **MouseOver** states at the same time.

- The state group for the button already exists, but, if we are not satisfied with the default, we can add a new state group by clicking on the **Add state group** button. This is shown in the following screenshot:



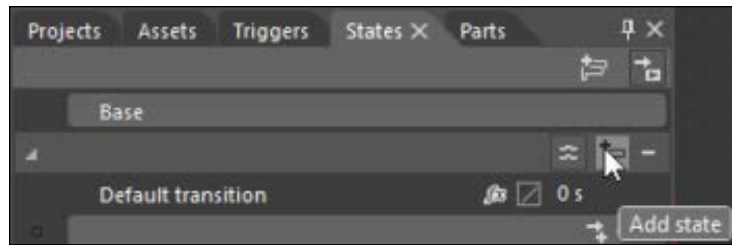
- Let's edit the **Normal** visual state. Select **Normal** under **CommonStates** and then select **border** in the **Objects and Timeline** panel. Notice the red indicator stating **Normal state recording is on**. This indicates that the changes that we are making to the **Normal** state are being recorded. We can toggle it between on and off by clicking on the red indicator. We will change **Background** of the button for this state. To do this, we would need to reset **Template Binding** of **Background** by clicking on the square next to the **Background** property and selecting **Reset**. The following screenshot shows this:



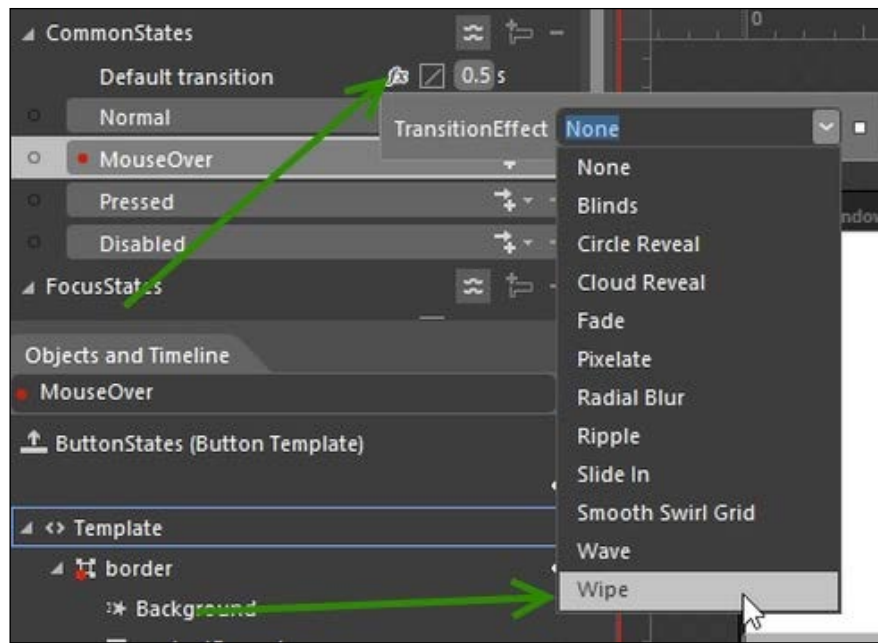
5. Change the background color of the border to a shade of blue, as shown in the following screenshot:



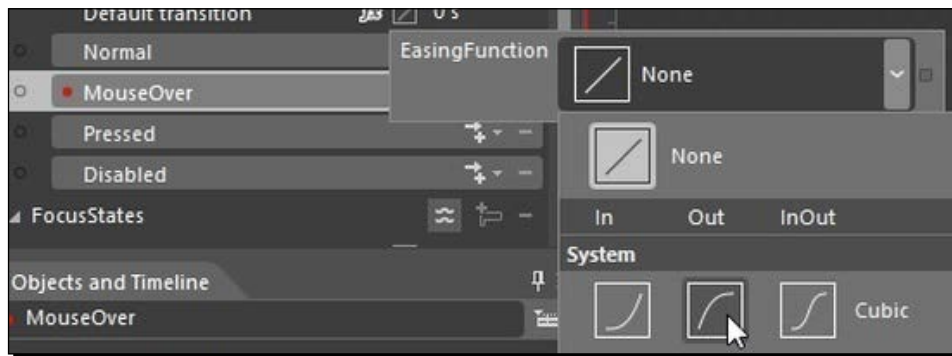
6. Similarly, change the background color of the border of the mouseover visual state now to a shade of green.
7. We could also add new states by clicking on the **Add state** button:



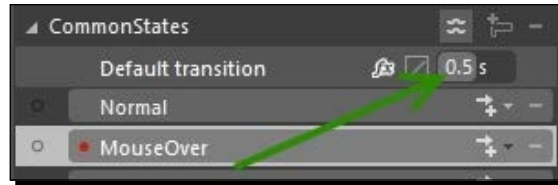
8. Now, let's add a transition effect to between the state changes. Click on **Transition Effect** for **CommonStates** and select **Wipe**. The following screenshot shows this:



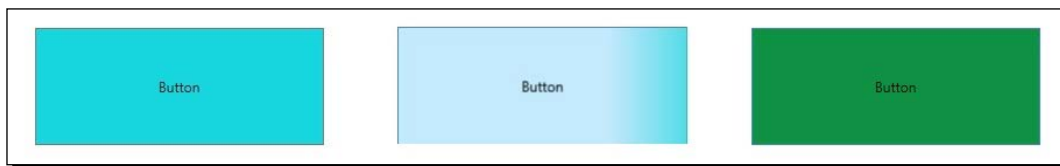
9. We will apply an **Easing Effect** cubic out to make the transition between states smooth, as shown in the following screenshot:



- 10.** If we run the application now, we notice that the state changes are happening instantaneously. We can configure this transition to be smoother by changing the transition duration from **0** sec to **0.5** sec, as shown in the following screenshot:



- 11.** Now, run the application. When we move the mouse over the button, we should see the transition from blue to green, as shown in the following image:



- 12.** When we have a look at the XAML code for visual states, we can see `VisualStateManager` as `CommonStates`. We can also see that we have set the `Background` property of the border panel for the `Normal` and `MouseOver` states. This is shown in the following XAML code:

```
<Border x:Name="border" BorderBrush="{TemplateBinding BorderBrush}" BorderThickness="{TemplateBinding BorderThickness}" SnapsToDevicePixels="true">
  <VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
      <VisualStateGroup.Transitions>
        <VisualTransition GeneratedDuration="0:0:0.5">
          <ei:ExtendedVisualStateManager.TransitionEffect>
            <ee:WipeTransitionEffect/>
          </ei:ExtendedVisualStateManager.TransitionEffect>
          <VisualTransition.GeneratedEasingFunction>
            <CubicEase EasingMode="EaseOut"/>
          </VisualTransition.GeneratedEasingFunction>
        </VisualTransition>
      </VisualStateGroup.Transitions>
      <VisualState x:Name="Normal">
        <Storyboard>
          <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(Panel.Background)" Storyboard.TargetName="border">
            <DiscreteObjectKeyFrame KeyTime="0">
              <DiscreteObjectKeyFrame.Value>
                <SolidColorBrush Color="#FF11D1DA"/>
              </DiscreteObjectKeyFrame.Value>
            </DiscreteObjectKeyFrame>
          </ObjectAnimationUsingKeyFrames>
        </Storyboard>
      </VisualState>
      <VisualState x:Name="MouseOver">
        <Storyboard>
          <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(Panel.Background)" Storyboard.TargetName="border">
            <DiscreteObjectKeyFrame KeyTime="0">
              <DiscreteObjectKeyFrame.Value>
                <SolidColorBrush Color="#FF0A8738"/>
              </DiscreteObjectKeyFrame.Value>
            </DiscreteObjectKeyFrame>
          </ObjectAnimationUsingKeyFrames>
        </Storyboard>
      </VisualState>
    </VisualStateGroup>
  </VisualStateManager.VisualStateGroups>
</Border>
```

What just happened?

We saw the various visual states present in button and modified a couple of them, and those changes are in effect. We also added the transition effect and the easing effect to the common states button. We can add similar or different behaviors to any element.

Have a go hero

Add more property changes in the state, such as font size, gradient background, and so on, to the home button, and add different states to the other buttons as well. Also, apply different transition effects to the visual states.

Pop quiz

Q1. Can an element be in multiple states at the same time?

1. Yes, if each of these multiple states belong to different state groups.
2. Yes, if each of these multiple states belong to same state groups.
3. Yes and these multiple states could belong to same or different state groups.
4. No.

Q2. Is it possible to default the visual states of a control?

1. Yes but cannot be done through Blend.
2. Yes and can be done through Blend.
3. Yes but requires advanced coding skills.
4. Not possible.

Summary

In this chapter, we had a look at the various panels and controls, including behaviors, visual states, and Visual State Manager.

In the next chapter, we will have a look at storyboards and animations.

6

Understanding Animation and Storyboards

Animations create the illusion of a scene by changing a series of different images, and the brain perceives it as a scene.

In the previous chapter, we talked about XAML—how we declare various elements in XAML and how we could assign various properties.

In this chapter, we will cover the following topics:

- ◆ Animations
- ◆ Storyboards

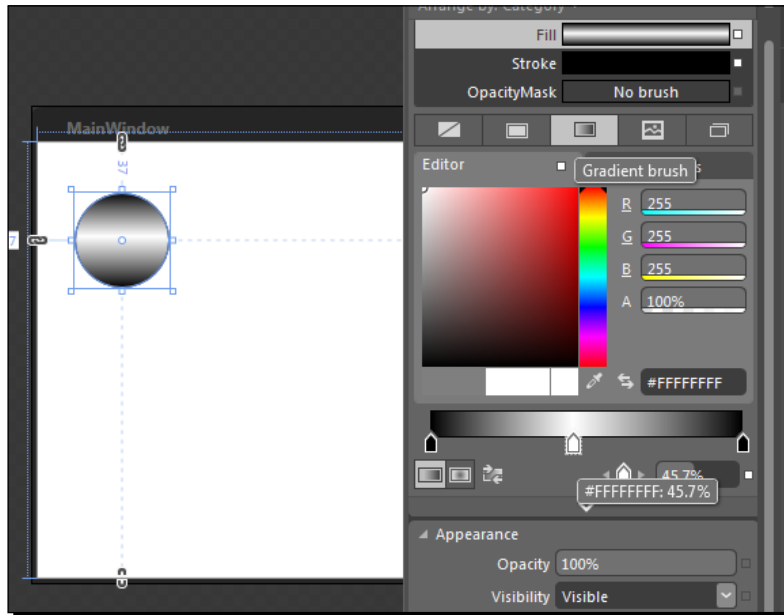
There are two popular animation techniques that are generally used. One is **frame-based animation** and the other is **time-based animation**.

In frame-based animation, the animation that we create is cut into frames and displayed one frame at a time. In a film, the camera does this by recording many photographs (frames) per second, and, when it is played back, it feels like a moving picture. The computer works in a similar way except that the frames can be further apart in time and the computer will animate and interpolate any changes in between. The problem with such kinds of animations is that they become resource intensive when they run.

Animations in WPF and Silverlight are based on **keyframes (time-based animation)**, where we define the start and end points of a visual transition, and the framework interpolates the property changes over time and displays the animation in our application. We will talk about keyframes in detail further on in the chapter.

Understanding the animation service

Let's create a project, and, along the way, we will talk more about animation. Create a new WPF application and name it `Chapter06`. We will start off by creating a simple animation of a ball moving from left to right onscreen. Go over to the **Assets** panel, select **Ellipse**, draw a circle on the screen, and give it a background color, as shown in the following screenshot:



Storyboards

Storyboards are containers to hold animation information. In the storyboard, we specify keyframes on a timeline to mark property changes, such as color, size, and so on, as we are creating the animation, run the animation storyboard to see how it works, and make adjustments. We can also control when, where, and how our storyboards run. The storyboard will determine the state of objects at any time during the animation. The storyboard doesn't change the base value of the property. It saves the current value and shows the animation for the specified duration. Then, it either restores the initial value or holds a specific value until the storyboard is removed. Any property of an object can be a target in the storyboard.

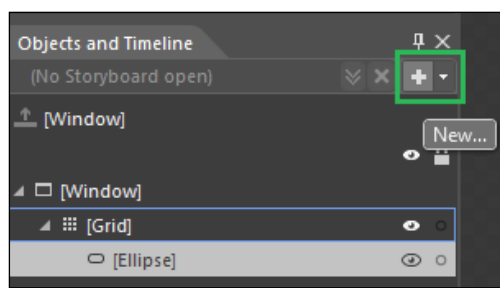
We can find more details about the storyboard class at <http://msdn.microsoft.com/en-us/library/system.windows.media.animation.storyboard.aspx>.



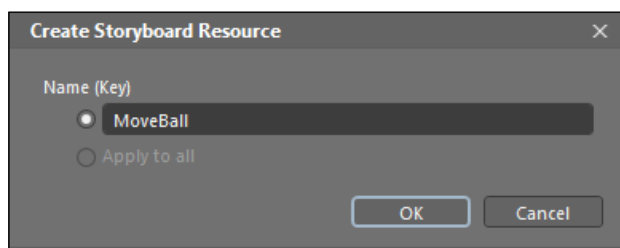
We can also create animations programmatically in code-behind files. This topic is beyond the scope of this book, but we can find more details at <http://msdn.microsoft.com/en-us/library/>.

Time for action – adding the storyboard

1. Go over to the **Objects and Timeline** panel and click on the + button in the top-right corner of the panel, as shown in the following screenshot. This will add a storyboard into our application.

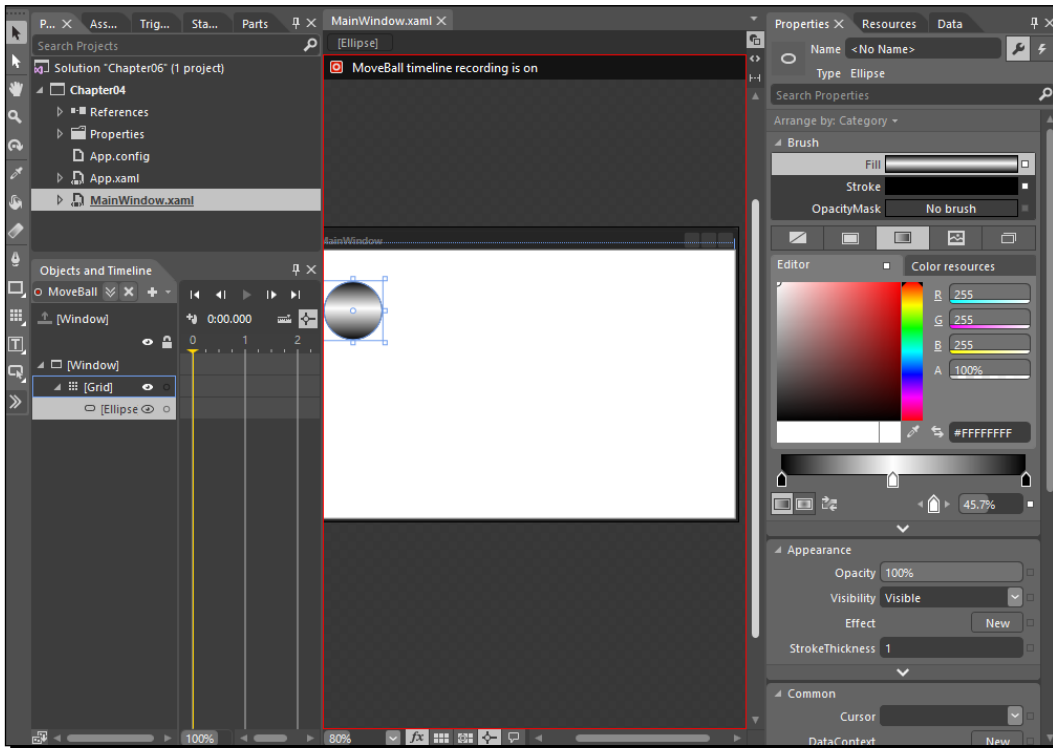


2. Once we click on the + button to add a storyboard, we will see a popup that asks us to either specify the name of the storyboard or apply the storyboard to all instances of the element. As we do not want to apply this storyboard to all ellipses, so we will change the name of the storyboard to `MoveBall`. This is the name that will be used to reference the storyboard in XAML as well as code. The storyboards are resources as evident in the title of the **Storyboard Resource** popup. The next screenshot shows this. So, just like any resources, storyboards could be reused and are available in the resources section of the Blend IDE. However, animations are very tightly coupled with the elements, so, generally, it is not useful to store the animation at application level unless we are using the same element at multiple places in the application for animations.

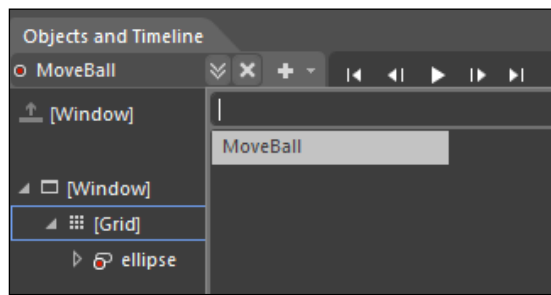


What just happened?


When we click on **OK**, we see a screen similar to the one shown here. This is also known as **animation workspace** and is the place where we could design our animation.



In the **Objects and Timeline** panel, Blend contains a storyboard picker control, from which we can search and select a storyboard in our project. This picker shows us the list of the available storyboards that we can open. This is demonstrated in the following screenshot:

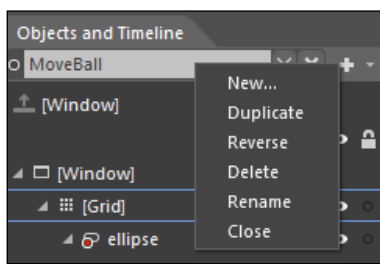


We can test the storyboard at the design time itself without having to run the application. When we select a storyboard, we see the playback controls on the **Objects and Timeline** panel. Let's go through them one by one from left to right:

- ◆ **First frame:** This moves the playhead  to the first frame of the animation
- ◆ **Previous frame:** This moves the playhead to the previous frame
- ◆ **Play:** This plays the animation from the current point in time, that is, from the current position of the playhead
- ◆ **Next frame:** This moves the playhead to the next frame
- ◆ **Last frame:** This moves the playhead to the last frame

After selecting the storyboard, if we right-click on the animation name, we see multiple options in a popup. Each option in the popup is described in the following list:


- ◆ **New:** This creates a new storyboard and asks for a name for it
- ◆ **Duplicate:** This creates a new storyboard that is the same as the existing one
- ◆ **Reverse:** This reverses the sequence and keyframes of the existing storyboard
- ◆ **Delete:** This deletes the selected storyboard
- ◆ **Rename:** This provides us with the option to rename the storyboard
- ◆ **Close:** This closes the animation view and storyboard

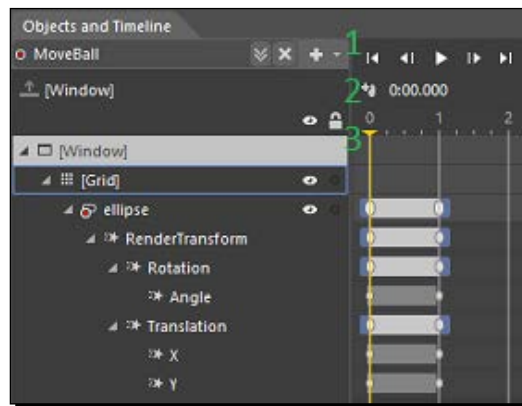


We can control storyboards using triggers (discussed later in the chapter).


Timelines

Animations in Blend are composed of timelines recorded on keyframes that represent the timing of property changes. Timelines provide a structure to the animation sequence in our application. We can think of the timeline as a layer on which the property changes of the objects are applied. The three sections of a timeline are depicted in the following screenshot:

- ◆ We can see the timeline inside the **Objects and Timeline** panel. At the top of the **Timeline** panel, we can see the various controls to play and seek the animation we are designing. The play button is enabled once we have added at least one animation timeframe.
- ◆ In the second section, we see the current playhead  position, which was 0:00.000 when we created the storyboard. Also, the other two options next to this playhead position are snapping options. The format of the time displayed is MM:SS:xxx (minutes, seconds, milliseconds) of the currently selected point.
- ◆ The third section that we see shows the visual representation of the playhead position, and we can drag it to the position we want.



Timeline recording

The center section displays the position of the objects at the current playhead  position. The red border that we see alongside the art board shows that the timeline recording is on and any property changes that we make to the object will be recorded at the time marked by the playhead. Once we are done editing the storyboard, we can turn off the timeline recording by clicking on the top-left portion of the recording area. Also, just below the **Objects and Timeline** label, we can see that the storyboard we are working with is selected and has a red icon next to it that shows we are in the recording mode.

Properties

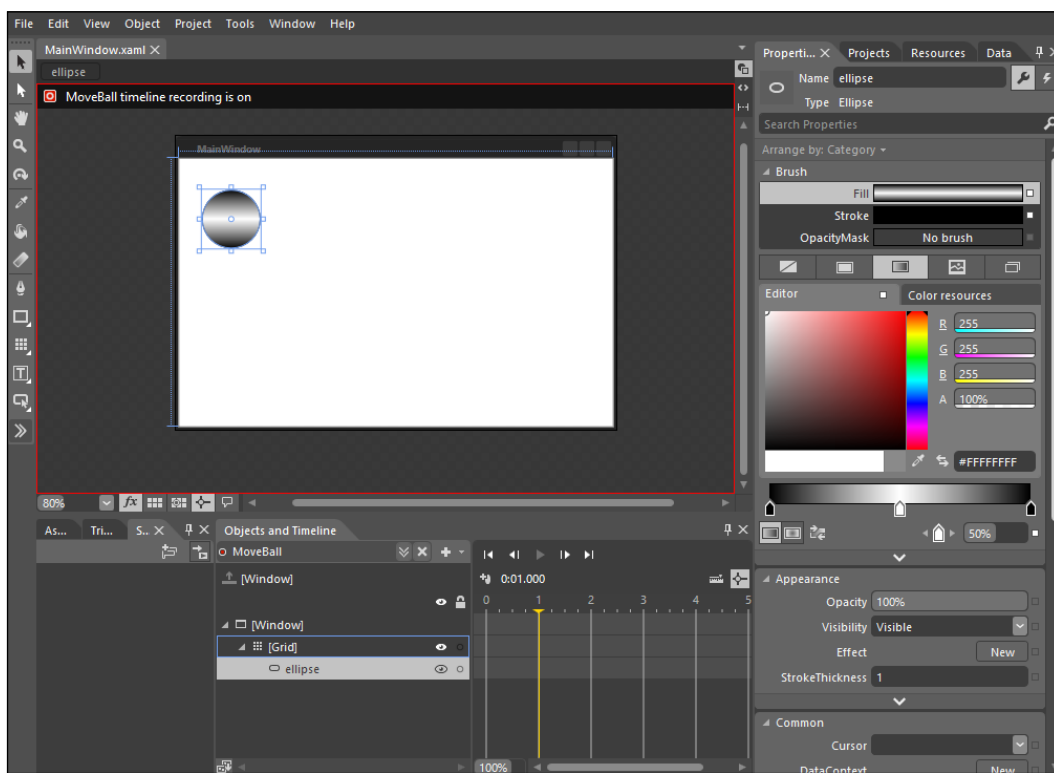
The right section is the **Properties** panel, which we use to specify the properties of the objects, but the difference here is that the properties that we change will be recorded as keyframes.

Animation workspace

The animation workspace allows more space to work with the animation by positioning the various panels.

Time for action – switching workspaces

When we press *Ctrl + F11*, Blend toggles between the design workspace and the animation workspace. In the animation workspace, we have more space to work with animation. This is shown in the following screenshot:




What just happened?

We just switched workspaces, from the design workspace to the animation workspace.

When we press *Ctrl + F11* again, we will move back to the design workspace.

Keyframe

A keyframe  in an animation sets a specific state of an object. A sequence of keyframes defines the movement that we see, and the position of a keyframe defines the movement timing. The first keyframe defines the starting of the animation and the next one determines how it's going to proceed. By changing the surrounding keyframes, we can change the starting or ending point of the transition, depending on whether the keyframe is before or after the transition.

For example, we can set a keyframe at the 0-second mark, record the position of the ball in the top-left corner of the art board, and then set a keyframe at the 1-second mark to record the position of the same ball in the bottom-right corner of the art board. Now, the animation will move the ball from the top-left corner to the bottom-right corner in 1 second.

When we run any storyboard animation, the WPF framework interpolates the property changes over the designated period of time and then displays the results in our application. We can use these keyframes and the storyboard to change any property of the object, and these properties can be visible or invisible.

There are four types of keyframes in Blend:


- ◆ **Object-level keyframes:** These keyframes apply to the whole object (such as an ellipse or rectangle) or to the object that contains multiple objects (such as a grid or canvas). In the latter case, when we expand the object node, we can see the individual elements on which we have recorded the keyframe. As we can see in the image, we have set the keyframe on the ellipse. To record such a keyframe, we can click on an object, such as an ellipse or rectangle, and click on the **Record Keyframe** button.
- ◆ **Compound keyframes:** These keyframes imply that the property has child properties being animated. To record such a keyframe, while still in the recording mode, change any of the compound properties of the object and link the translation and the keyframe will be automatically created.
- ◆ **Simple keyframes:** These keyframes represent the property change of a single property. As illustrated in the screenshot, the **X** and **Y** properties are simple keyframes. To record such a keyframe, while still in the recording mode, change any of the simple properties of the object, and the keyframe will be automatically created.

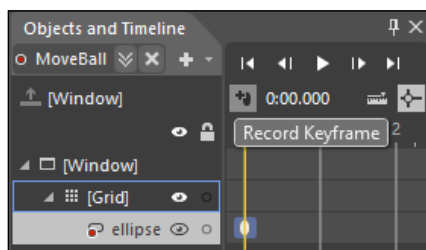
- ◆ **Implicit keyframes:** These keyframes are present when we move from one animation to another and the second animation does not have a keyframe at the 0-second mark. Blend animates the change from the last known value of the property to the first keyframe of the second animation. This last known value is known as the implicit keyframe even when this is the value between two keyframes in the first animation.

Knowledge about keyframes is useful while we are developing animations. There are times when we do not need to see the details of all the properties being animated and work better and faster with object-level keyframes and compound keyframes to modify a large group of properties at the same time with a single selection.

Time for action – using keyframes

Let us add a keyframe:

1. Select the ellipse and click on the **Record Keyframe** button, as shown in the next image. This will record the current state of the selected object. So, when we click on the **Record Keyframe** button, make sure that we select only relevant objects just as we selected the ellipse here.
2. A keyframe is responsible for changing the properties of an object. Once we click on the **Record Keyframe** button, we can see a white mark  in front of the **ellipse** object as it is the object that was selected when we clicked on **Record Keyframe**. So, this keyframe has the value of all the properties of **ellipse** at the 0 sec mark. This is shown by the following screenshot:



What just happened?



We just added a keyframe that will be in the initial state for the animation to start.

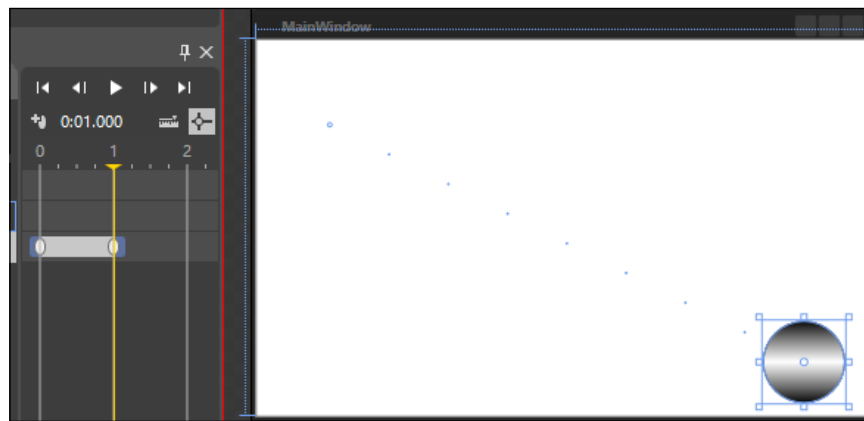
Translation and rotation animation

Let's now add some transformation to the ball so that our animation looks more realistic.

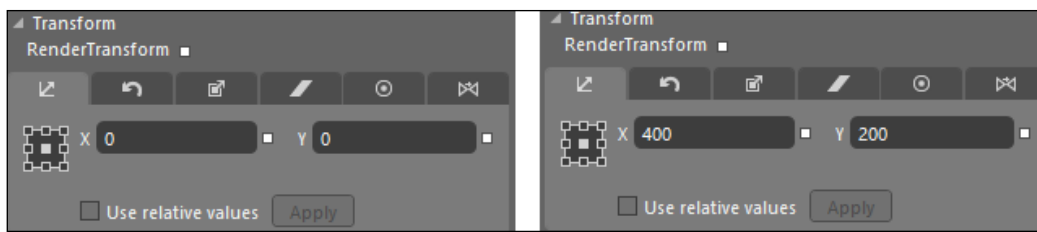
Time for action – using transforms

Perform the following steps to add some transformation to the ball:

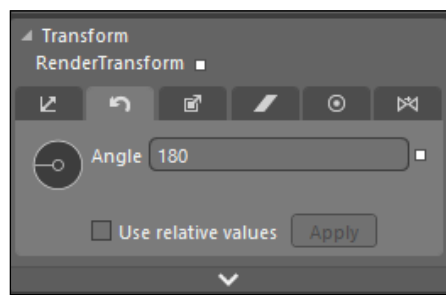
1. Now, move the playhead  in the **Timeline** panel from the current position, that is, 0, to 1. Click on and drag the ellipse to the bottom-right corner of the art board. When we drag the ellipse, we will notice a few changes:
 - The play button is now enabled. When we click on it, we can see the animation playing. And, we see a highlighted white mark  of the playhead in front of the ellipse. So, this keyframe saves all the properties of the ellipse at the 1 sec mark.
 - We also see a series of dots connecting the original position of the ellipse to the new position. This actually depicts the path that the ellipse will follow when the animation happens. This is shown in the following screenshot:



2. The animation that we have created moves the ball from the top-left corner to the bottom-right corner of our art board. When we go to the **Transform** section in the **Properties** panel with the ellipse, we see the multiple transforms available to apply to an element. The first one is the `Translate` transform, which specifies the position of the element using the `TranslateX` and `TranslateY` properties. In the following image, we can see the `TranslateX` and `TranslateY` properties at the 0 sec and 1 sec marks. This means that, in 1 sec, the ellipse will move 400 pixels to the right and 200 pixels toward the bottom.



3. The next transform that we have is `Rotate`. We can also add rotation animation to the ball. To do that, move the playhead to the 1 sec mark if it is not already there. Then, move the **Transform** section in the **Properties** panel if it is not already there. Select rotate transform and change the rotation angle from 0 to 180. By doing this, we have specified that, at the 1 sec mark, the ellipse should rotate by 180, but the animation engine of Blender will take care of rotating the ellipse by 180 degrees over the time period of 1 sec. Now, if we play the animation, we would see that the ball will translate as well as rotate. The following screenshot shows this:



What just happened?

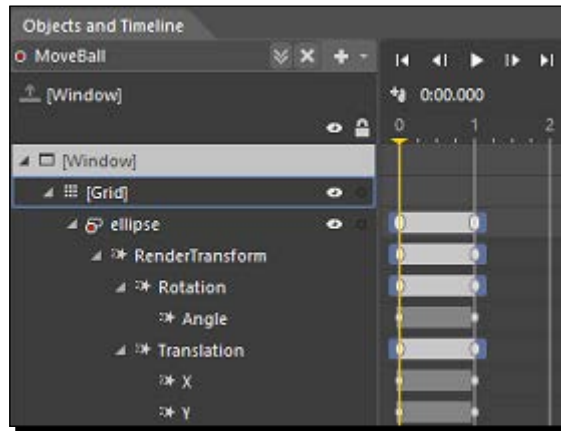
We just added translate as well as rotate transformations on the ball.

Have a go hero

Add multiple transformations to the ellipse and see their effects.

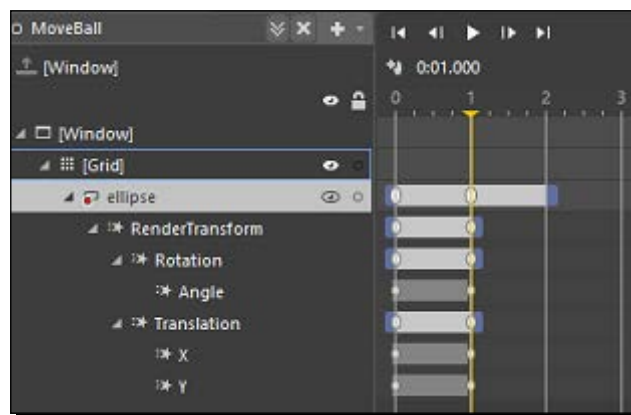
Animation recording symbol

We also notice that, while working with animation, we see a red circle on the object that we are animating, and that we can expand the ellipse to see details of the animations. This will show the various transforms we have applied and the times at which these transformations have been applied. The following screenshot shows this:



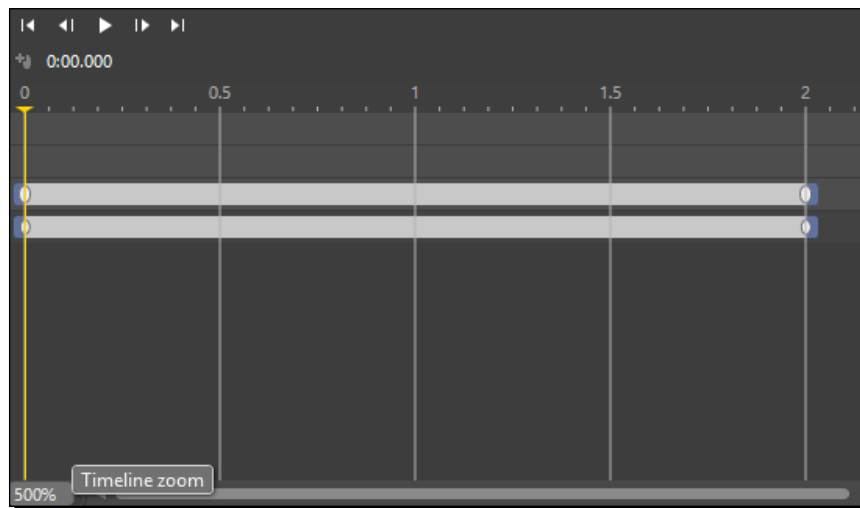
Keyframe editing

Another cool technique that we can use while working with animations is extending or shrinking the time for one or more keyframes. We can either select the keyframe/s we want to move to a different time or move the complete span by clicking on and dragging on the gray area (known as the timeframe). If we want to extend the time for a keyframe, we will need to click on and drag the blue area at the end of the timeframe (the gray area). This discussion is encapsulated by the following screenshot:



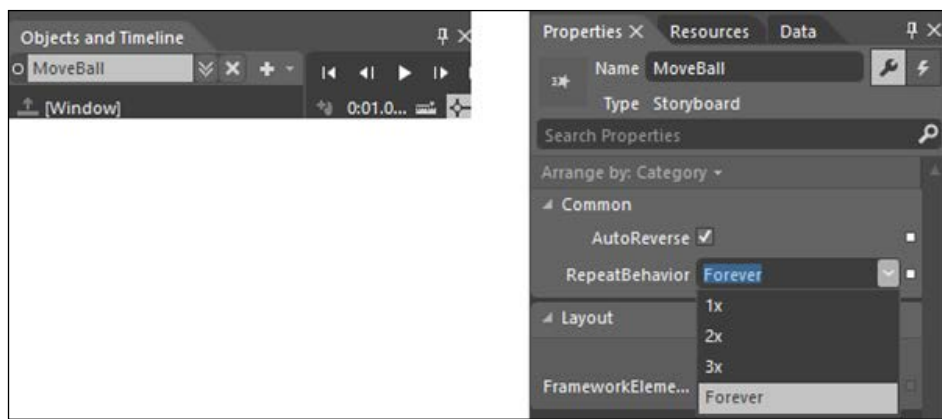
The Timeline zoom feature

Another feature that comes in handy at times is the **Timeline zoom** feature; this comes in handy when we are working with very short timespans. **Timeline zoom** is available at the bottom of the timeline. We could just click on it and enter the zoom value that we want for the **Timeline** panel. So, it will show even smaller time intervals to work with. The following screenshot depicts this:



Storyboard properties

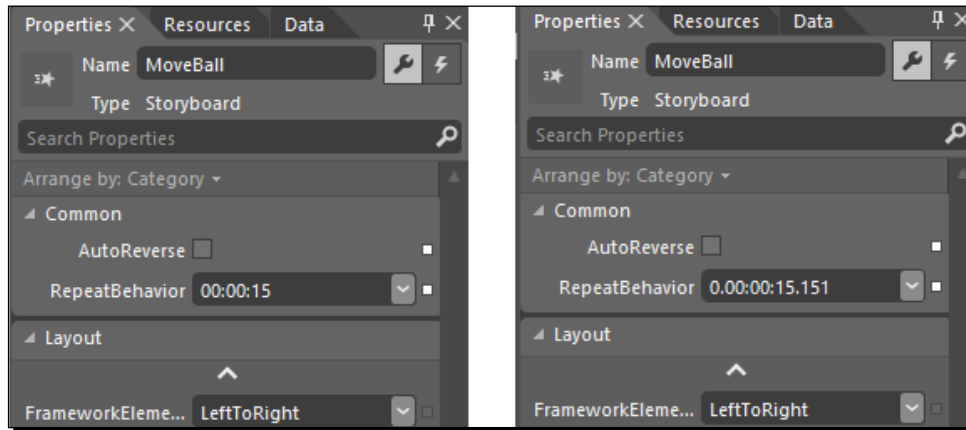
Now let's select the storyboard that we just created. When we do that, we can see a few properties in the **Properties** panel of the storyboard, as shown in the following screenshot:



The properties in the **Properties** panel of the storyboard are described here:

- ◆ **AutoReverse:** This will play the animation in the reverse direction once the animation finishes.
- ◆ **RepeatBehavior:** This will repeat the animation a specified number of times. The options that we see here are **1x**, **2x**, **3x**, and **Forever**. x represents the number of times an animation will run. We can also set RepeatBehavior for the number of times we want the behavior repeated by editing it manually. We can also specify the time for which we want to run the animation. The following are the two formats in which we can specify the amount of time for which we want the animation to run:
 - hours:minutes:seconds
 - days.hours:minutes:seconds.fractionalSeconds

The following screenshots capture the essence of the preceding paragraph:



Now, when we run the animation after selecting these behaviors, we will see these properties in action by ourselves.

XAML for the storyboard

In the previous section, we saw how we can create a simple animation, and we used the Blend designing interface to do that. Now, let's take look at the XAML code generated by Blend in the background and see how we can tweak that to make changes. The following is the XAML code generated:

```
<Window.Resources>
  <Storyboard x:Key="MoveBall" RepeatBehavior="00:00:15"
    AutoReverse="False">
```

```

<DoubleAnimationUsingKeyFrames Storyboard.TargetProperty="
  (UIElement.RenderTransform).(TransformGroup.Children)[3].
  (TranslateTransform.X)" Storyboard.TargetName="ellipse">
  <EasingDoubleKeyFrame KeyTime="0" Value="0"/>
  <EasingDoubleKeyFrame KeyTime="0:0:2" Value="400"/>
</DoubleAnimationUsingKeyFrames>
<DoubleAnimationUsingKeyFrames Storyboard.TargetProperty="
  (UIElement.RenderTransform).(TransformGroup.Children)[3].
  (TranslateTransform.Y)" Storyboard.TargetName="ellipse">
  <EasingDoubleKeyFrame KeyTime="0" Value="0"/>
  <EasingDoubleKeyFrame KeyTime="0:0:2" Value="200">
    <EasingDoubleKeyFrame.EasingFunction>
      <BounceEase EasingMode="EaseOut"/>
    </EasingDoubleKeyFrame.EasingFunction>
  </EasingDoubleKeyFrame>
</DoubleAnimationUsingKeyFrames>
<DoubleAnimationUsingKeyFrames Storyboard.TargetProperty="
  (UIElement.RenderTransform).(TransformGroup.Children)[2].
  (RotateTransform.Angle)" Storyboard.TargetName="ellipse">
  <EasingDoubleKeyFrame KeyTime="0" Value="0"/>
  <EasingDoubleKeyFrame KeyTime="0:0:2" Value="180"/>
</DoubleAnimationUsingKeyFrames>
</Storyboard>
</Window.Resources>
<Window.Triggers>
  <EventTrigger RoutedEvent="FrameworkElement.Loaded">
    <BeginStoryboard Storyboard="{StaticResource MoveBall}"/>
  </EventTrigger>
</Window.Triggers>

```

There are a few points to note:

- ◆ The storyboard is placed in the `Window.Resources` tag. This means that this storyboard is accessible to all the elements inside this window. The other places to place this storyboard are:
 - Inside the element itself that we are animating; this will make sure that this storyboard is accessible only to that element
 - `App.xaml` or resource dictionary is the other place, which will make the storyboard available to all the objects and elements in the application
- ◆ Then, we see that the storyboard has key (`MoveBall`), which we specified while creating our storyboard.

- ◆ Then, we see sections of `DoubleAnimationUsingKeyFrames`. These are sections that are used to perform the animation as these are places where all the information about the animation is stored. `DoubleAnimationUsingKeyFrames` is created when we deal with double values. The animation type that we are dealing with should match the storage type. Multiple types of animations are available in Blend, such as `ColorAnimation` to change colors, `ThicknessAnimation` for changing thickness, and so on. You can find more information on the types of animations at [http://msdn.microsoft.com/en-us/library/cc189038\(v=vs.95\).aspx#animation_types](http://msdn.microsoft.com/en-us/library/cc189038(v=vs.95).aspx#animation_types).
 - In the first instance of `DoubleAnimationUsingKeyFrames`, `TargetProperty` is `TranslateTransform.X`. `TranslateTransform` is used to move an element from one position to another. The `X` and `Y` properties are used to move an element toward the `x` and `y` axes. `TranslateTransform.X` specifies how much the ellipse will move in the horizontal direction. We can also see the time that will be taken to move, which is specified as the `KeyTime` property, and the amount that it will move is specified using the `Value` property of `EasingDoubleKeyFrame`.
 - In the second instance of `DoubleAnimationUsingKeyFrames`, `TargetProperty` is `TranslateTransform.Y`, and this specifies how much the ellipse will move in the vertical direction. We can also see the time that will be taken to move is specified as the `KeyTime` property, and the amount by which it will move is specified using the `Value` property of `EasingDoubleKeyFrame`. `EasingDoubleKeyFrame` associates an easing function with `DoubleAnimationUsingKeyFrame` as, in the preceding animation, we have added the `BounceEase` easing function.
 - The third instance of `DoubleAnimationUsingKeyFrames` specifies the rotation details, and we can see that the target property in this case is `RotateTransform.Angle`, and that its value will change to 180 in 1 second.
- ◆ Below the `Window.Resources` section, we can see the `Window.Trigger` section, which specifies at what event this animation will start. Currently, the storyboard is configured to start at the `Loaded` event. So, if we run this application, as soon as the load is complete, the animation will start.

So, we have figured out that storyboards are made up of structures called keyframes, and that these structures can be changed by specifying the time, property, and value.

Transition between keyframes

Using keyframe interpolation, we can define how property changes will animate between the two keyframes. We can modify the keyframe interpolation by doing the following:

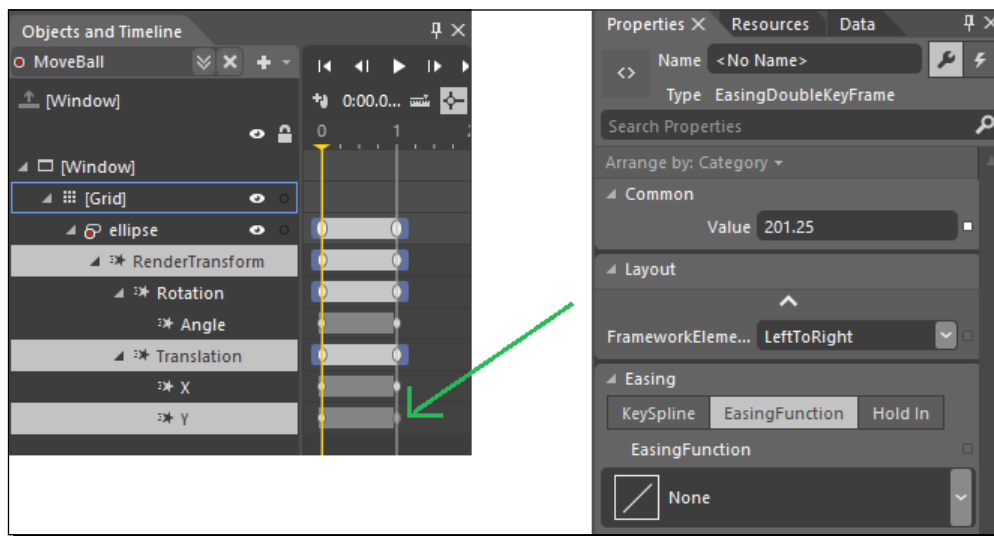
- ◆ Selecting the predefined **ease in** or **ease out** values
- ◆ Visually modifying the **KeySpline** graph
- ◆ Using a predefined **EasingFunction** (complex KeySpline graph)

Easing functions

Easing functions allow us to apply custom mathematical formulas to your animations. Easing effects in Blend can give effects like the ball slowing down or the ball moving faster (using easing functions such as circle, cubic, and so on), or, when a ball falls, we can add bounces (using easing functions such as bounce, elastic, and so on).

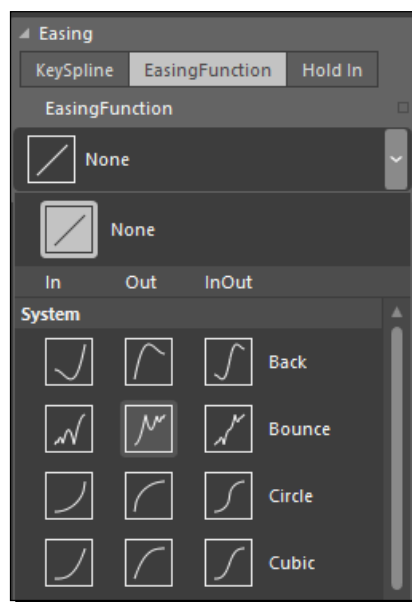
Time for action – using easing functions

Let us switch back to the design view. Expand the ellipse as shown in the image following image. Select the **Translation** section in it. When we do that in the **Properties** panel, we see something called `EasingFunction` selected as none. Go ahead and select the keyframe at the 1 sec mark in front of `Translation.Y`. We will see something like the following:



We can see and take note of a few things:

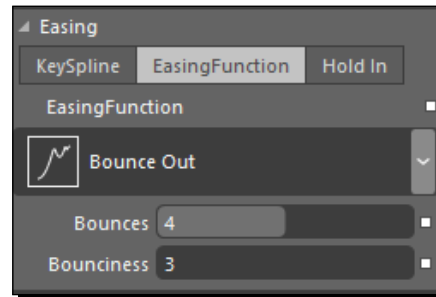
- ◆ We see the current value of `Translation.Y` as `201.25`, that is, the `Y` position of the rectangle at the `1 sec` mark.
- ◆ We see `EasingFunction` selected as `None`, which means that the translation will be linear.
- ◆ Easing functions specify the interpolation that the object will go through while moving from one keyframe to another. When we expand the `EasingFunction` dropdown, we will see a number of instances of `EasingFunction`:



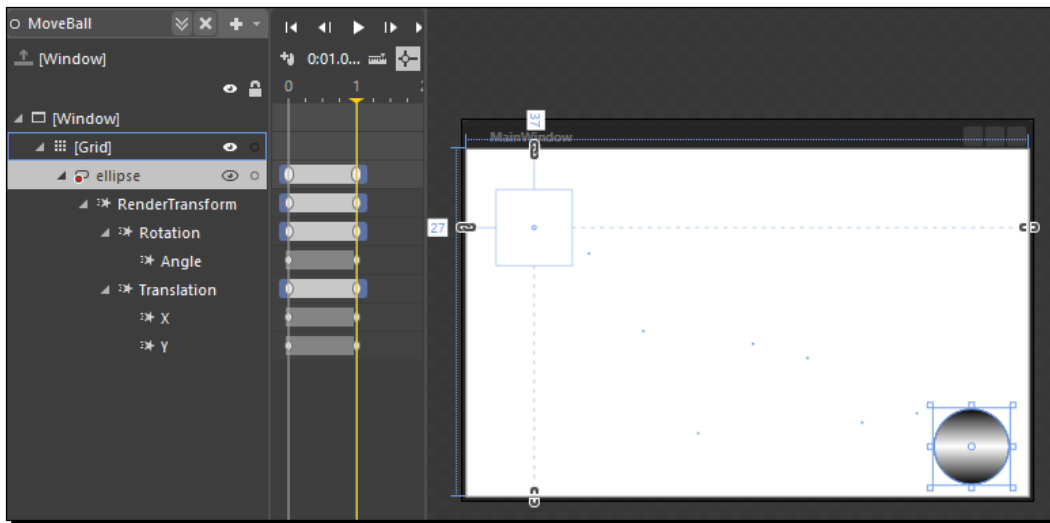
`EasingFunction` is divided into three types:

- ◆ **In:** This applies the easing effect to the beginning of the keyframe.
- ◆ **Out:** This inverts the formula applied to the interpolation. That means that the easing effect is not applied to the end, but starts from the end and moves toward the beginning.
- ◆ **InOut:** The easing effect starts in a straight manner, and half of the animation is inverted.

Select the **Bounce Out** transform. So, this will bounce the ellipse before it reaches its final value. After selecting this effect, we also see some additional properties that we could configure. They are the number of bounces and the bounciness of the object. Different easing effects have different configurable options. This is shown in the following screenshot:



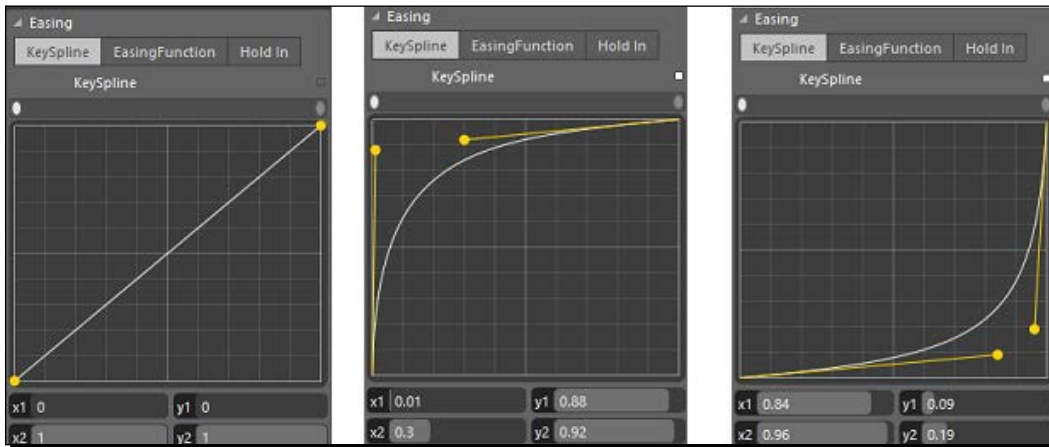
Also, if we go back and select the ellipse in the **Objects and Timeline** panel, we will see the new path of the ellipse shown by dots, as shown in the following screenshot:



If we play the animation or run the animation, we will see that the ball is bouncing rather than just moving in a straight line.

KeySpline

Next to the **EasingFunction** button, we can also see the **KeySpline** button, which could help us achieve a finer level of easing effects in our animation. So, we can either create our own KeySpline or use EasingFunction, which is a predefined form of KeySpline. If we click on the **KeySpline** button, we will see something like the following:



The horizontal axis depicts the length of animation that has been executed and the vertical axis represents the time passed. The straight line from the bottom-left corner to the top-right corner specifies that the animation will proceed linearly through time. However, we can change that by dragging the yellow circles around.

In the following image, the first setting makes the animation go faster initially and slow down later, and the second image shows the animation proceeding slowly initially and going fast later:

What just happened?

We just added an easing function and modified KeySpline for an animation.

Have a go hero – using different easing effects

Go ahead and try different easing effects in the animation and see how they behave.

Pop quiz

Q.1 How do WPF and Silverlight do the animation?

1. Adding and removing properties from the control
2. Adding properties to controls
3. Removing properties from control
4. Changing property values of controls

Q.2 How can we have the animation to run continuously repeatedly?

1. Set the RepeatBehavior of the animation to 1x.
2. Set the RepeatBehavior of the animation to Forever.
3. Set the RepeatBehavior of the animation to 2x.
4. Set the RepeatBehavior of the animation to 3x.

Summary

In this chapter, we had a look at how we can create storyboards and animation in Blend. In the next chapter, we will have a look at user controls and custom controls.

7

Understanding DataBinding

DataBinding is one of the most important services offered by WPF. DataBinding is used to connect the user interface to the information to be displayed (data model) and the user interface back to the data model. The data source can be properties of data from the database, XML, or other elements (including graphical elements). DataBinding is the core of the styling, animation, storyboard, and behaviors, and the basis of DataBinding are dependency properties.

In the previous chapter, we had a look at animations and storyboards. We had a look at different ways to create animations and use them in our application. This chapter will cover the following topics:

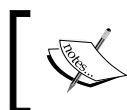
- ◆ Dependency properties
- ◆ DataBinding
- ◆ Using XML and local storage

Understanding dependency properties

A dependency property is like a normal CLR property, the value of which we can get and set. The only difference between the CLR property and dependency property is that the value of the dependency property is inherited down the visual tree and it implements the `INotifyPropertyChanged` interface for change notifications. All WPF entities have access to dependency properties as they are backed by the WPF property system. The advantage of using a dependency property is that the dependency property could dynamically derive its value from other inputs, such as animations, styles, behaviors, storyboards, and resources.

The ultimate goal of a dependency property is to manage the state, but dependency properties are registered with the dependency property framework, and the underlying property value is determined by the dependency property framework based on rules defined by the property registration.

The class that we use to create a dependency property inherits from the dependency object. `DependencyObject` is a dictionary that stores the local values of the dependency properties. The key of the dictionary item is the key defined with the dependency property. When we access a dependency property over its .NET property wrapper, it internally calls `GetValue(DependencyProperty)` to access the value. This method resolves the value using a value resolution strategy. If a local value is available, it reads it directly from the dictionary. If no value is set, it goes up the logical tree and searches for an inherited value. If no value is found, it takes the default value defined in the property metadata.



More details about the dependency object can be found at [http://msdn.microsoft.com/en-us/library/system.windows.dependencyobject\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/system.windows.dependencyobject(v=vs.110).aspx).

Understanding the attached property

An attached property is used as a type of global property that can be set to any object. In WPF and Silverlight, an attached property is defined as a specialized form of the dependency property that does not have the conventional `wrapper` property as we created in the previous section.

The purpose of an attached property is to allow the child elements to set the value of the property that actually belongs to their parent. For example, we have set the `Canvas.Top` or `Canvas.Left` properties in child controls even when these properties actually belong to `Canvas`:

```
<Canvas>
  <Button Canvas.Top="100" Canvas.Left="100">Click Me</Button>
</Canvas>
```

An introduction to DataBinding

Typically, `DataBinding` establishes the link between two objects (generally between the UI and business logic of an application) so that one object is updated when the other object is changed. Generally, the changes in the source object are reflected onto the target object. However, `DataBinding` also provides the capability to update the source from the target or both the source and target to update each other.

The data provides the notifications so that when the data changes its value, the elements that are bound to the data reflect changes automatically. `DataBinding` can also work in the opposite direction, that is, if the value of a UI element changes, the data bound to that element also automatically updates to reflect the change. For example, if the user edits the value in a `TextBlock` element, the underlying text data automatically updates and reflects the changes. This helps us in creating user interfaces that can be populated with minimal code. This also helps us in separating the UI for the code logic.

`DependencyProperty` has all the plumbing for `DataBinding` built-in. If you bind something to `DependencyProperty`, it will notify when it changes. So, for `DataBinding` to exist, we need `DependencyProperty`. The target property of a `DataBinding` should be `DependencyProperty`.

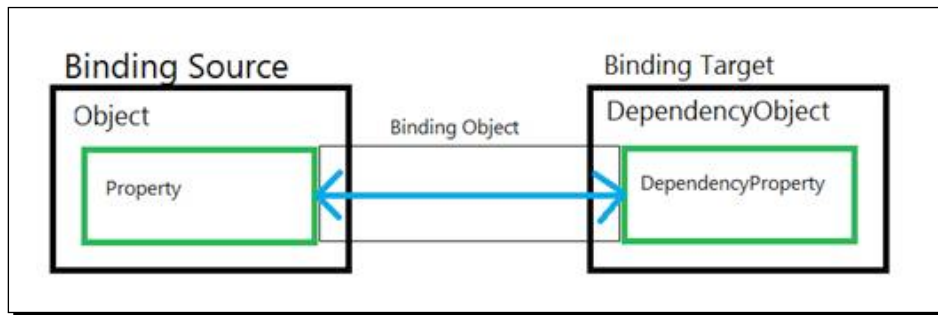
DataBinding modes

There are multiple modes available for `DataBinding`, and these modes are based on the direction of the flow of data. There are five modes in which we can set the `DataBinding` flow:

- ◆ **OneWay**: This `DataBinding` mode binds the data from the source to the target. This mode is also referred to as read-only as only the source can update the target automatically.
- ◆ **TwoWay**: This `DataBinding` mode binds the data from the source to the target as well as vice versa. This mode is also referred to as read-write as the source can update the target automatically and vice versa.
- ◆ **OneWayToSource**: This is the reverse of the `OneWay` mode and binds the data from target to the source. This mode is also referred to as write-only as only the target can update the source automatically.
- ◆ **OneTime**: This is the type of `DataBinding` from the source towards the target, but the property value is set only at the time of initialization and not updated after that.
- ◆ **Default**: This mode sets the default `DataBinding` mode, which is generally one-way or two-way depending upon the type of element.

The DataBinding model

Each and every kind of DataBinding follows the same model. DataBinding has a target, and the target has to be an element in the user interface. As illustrated in the following image, binding acts as the glue between the binding source and binding target:



Each binding has four components as shown in the preceding image. These components are the binding source, binding target, and binding target property and a path to the binding source value so that it can be used.

There are a few restrictions for DataBinding to work. They are as follows:

- ◆ To keep the source and target properties updated and synchronized, we need to implement the `INotifyPropertyChanged` interface, which has a single `PropertyChanged` event. More details on `INotifyPropertyChanged` can be found at [http://msdn.microsoft.com/en-us/library/system.componentmodel.inotifypropertychanged\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/system.componentmodel.inotifypropertychanged(v=vs.110).aspx).
- ◆ The target property has to be `DependencyProperty` or a CLR object that has implemented `INotifyPropertyChanged`. Mostly the UI element properties are dependency properties, and they support DataBinding by default as they implement the `INotifyPropertyChanged` interface.



Once DataBinding is set up, all the synchronization work is done by DataBinding. When DataBinding fails, it does so silently (a debug message is written, and, with tracing, a detailed explanation of the failure to bind is provided) and the default value is sent.

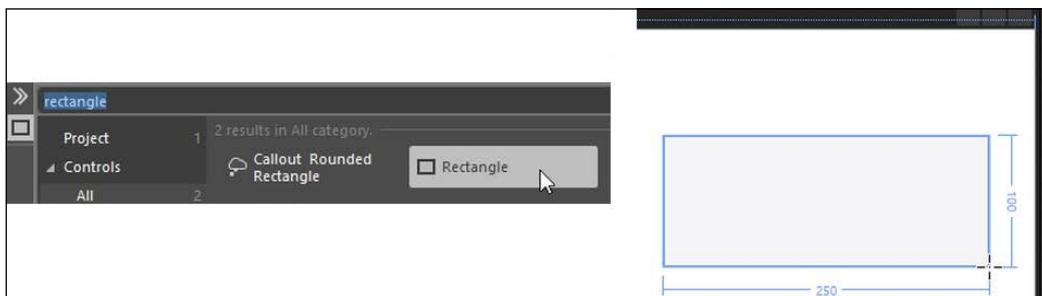
DataBinding properties to control

We will bind one property of the control to another property of the same control.

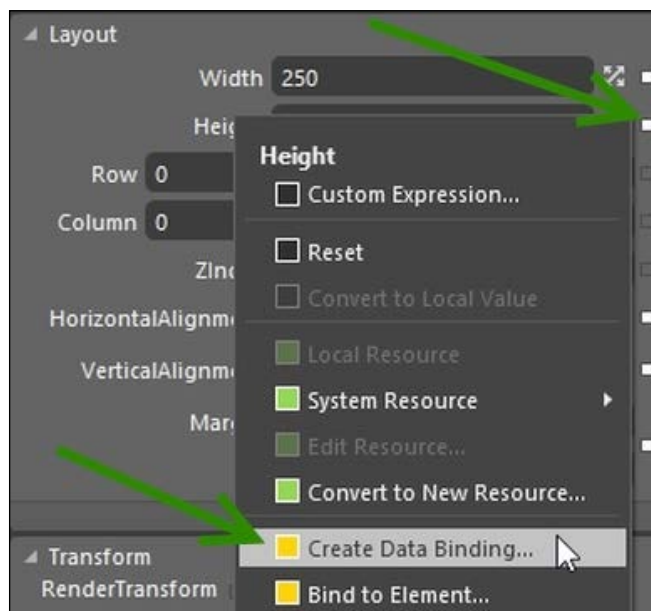
Time for action - DataBinding to one's own property

In this section, we will DataBind one property of the **Rectangle** shape to another one as follows:

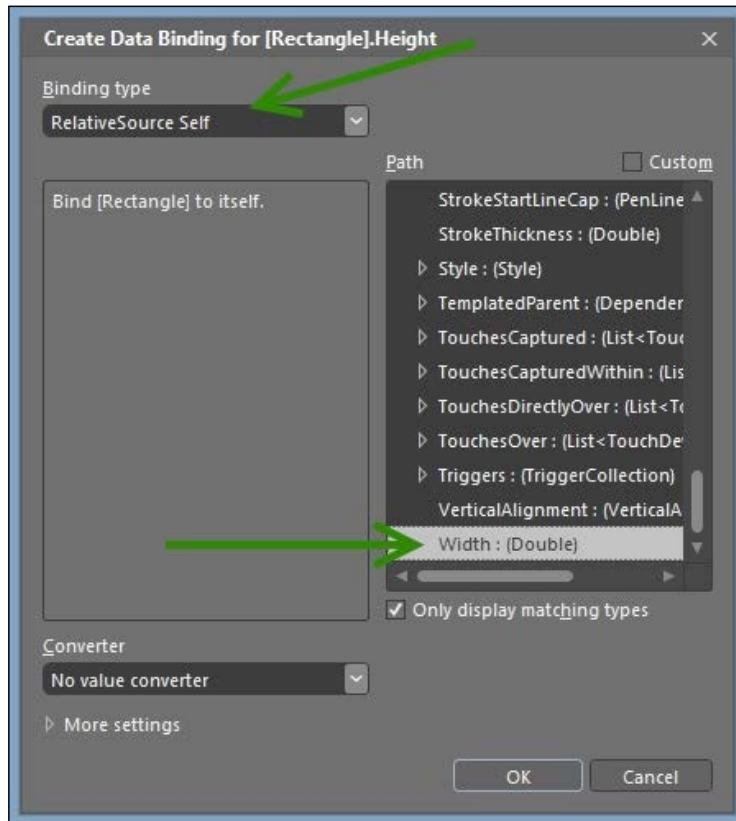
1. Create a new project in Blend and name it `Chapter07`. Drag and drop a **Rectangle** shape onto the art board. This is shown in the following screenshot:



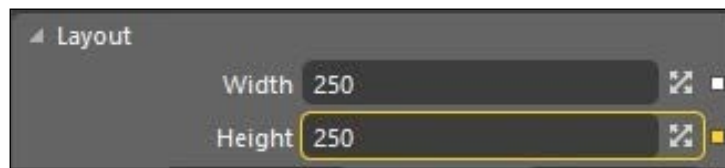
2. Now, we will go ahead and DataBind the **Height** property of the rectangle to itself. The way to assign the source property in DataBinding is to assign the source of the property in DataBinding is to assign the source of the property path. To do that, move to the **Properties** panel, left-click on the small white rectangle next to the **Height** property, and then select **Create Data Binding...** This is shown in the following screenshot:



- Now, select the binding type as **RelativeSource Self** as we want to bind the rectangle with a property of its own. We will see a list of properties that the **Height** property can be bound to. Select **Width: (Double)** and click on **OK**. The following screenshot encapsulates this discussion:



- Once we do that, we will see that the **Height** property is set to 250 as its value now comes from the **Width** property, which is set to 250. We also see a yellow rectangle across the **Height** selection option, and it implies that we cannot set this value directly and that this value will come from DataBinding. This is depicted in the following screenshot:





The `RelativeSource` property could also be used to set the source to an ancestor of a specific type. The source could be set to the first ancestor of the type or the n^{th} ancestor of the type.

Using `RelativeSource`, we could also set the source to `TemplatedParent`. We talked about `TemplatedParent` in *Chapter 4, Styles and Templates*.

The source could also be set to the previous data item in the data-bound collection.

5. Switch to XAML, and we will see the syntax for `DataBinding`. We have specified the value for the **Height** property, which is within the curly braces. We can see that the `Height` target property is set to a binding instance with `PropertyPath` as the `Width` property and `RelativeSource` as the `self`, which means that we are binding it to the same element. The `RelativeSource` property allows us to specify the source element from its relationship with the target rather than using the source element's name. The following code exemplifies this:

```
<Grid x:Name="LayoutRoot">
  <Rectangle Fill="#FFF4F4F5"
    Height="{Binding Width, RelativeSource={RelativeSource
    Self}}">
    Width="250" Margin="347,81,0,0" Stroke="Black"
    VerticalAlignment="Top"
    HorizontalAlignment="Left"/>
</Grid>
```

What just happened?

We did a `DataBinding` of the height of the rectangle to the width of the same element itself.



`DataBinding` makes it easy to display data on the user interface. If we need to display multiple items in the list, instead of adding an item one by one in code or in XAML, we could simply bind the element with a list of items.

Have a go hero

Try different combinations of DataBinding and see different types of properties work with Height.

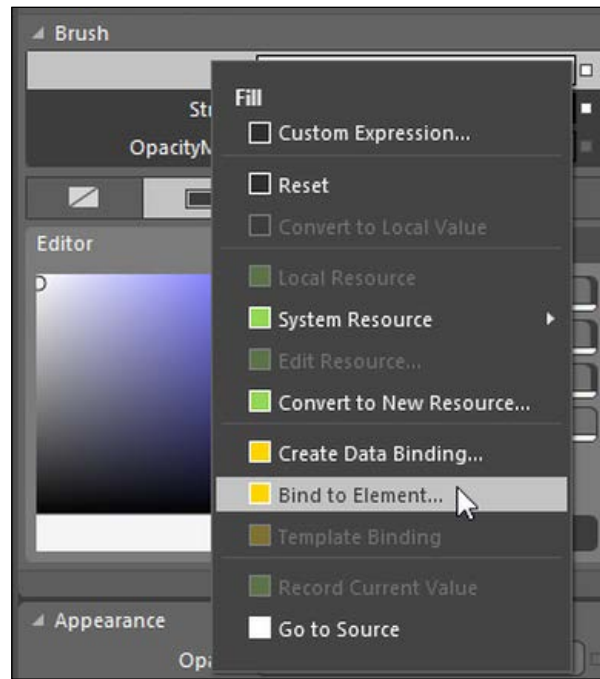
DataBinding control to control

We will now see how we can bind the properties of one control to another control. The DataBinding infrastructure is quite versatile; it not only gives us the ability to bind the controls to properties but directly with the properties of other controls as well.

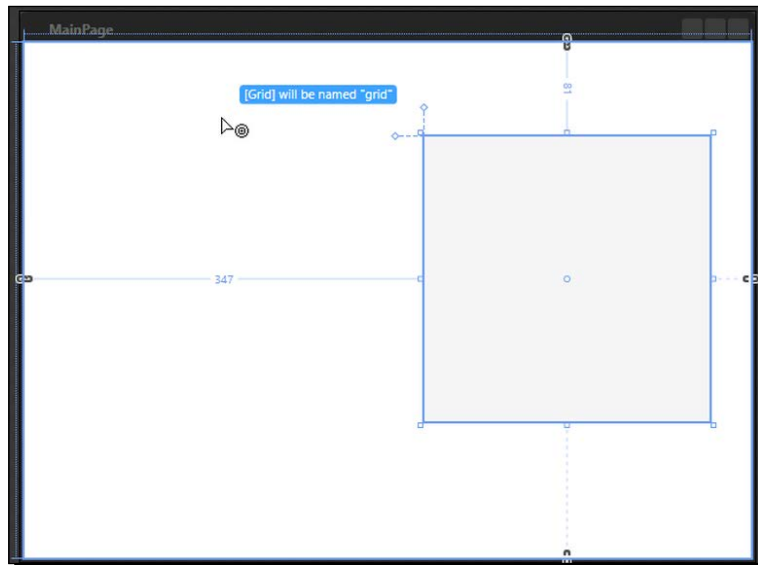
Time for action – DataBinding to properties of a different control

Perform the following steps to bind data to properties of a different control:

1. Drag and drop a rectangle from the art board onto the `MainPage.xaml` and move to the **Properties** panel.
2. Left click on the small, white rectangle next to the **RectangleBackground** property, and then select **Bind To Element...**. The following screenshot shows this:

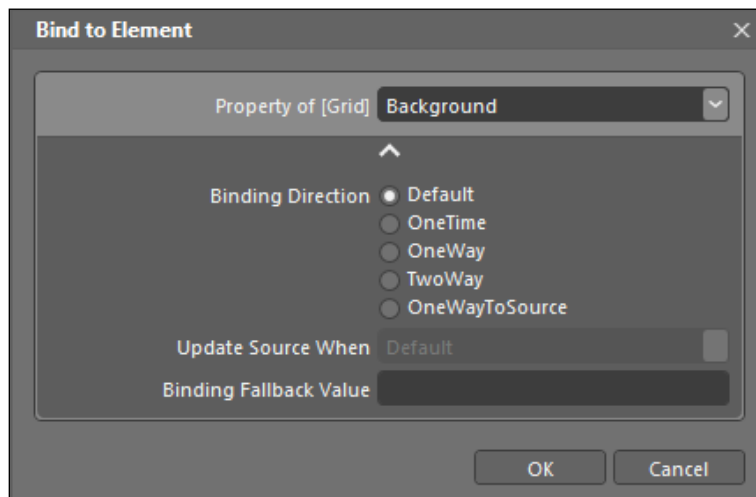


3. Now, go ahead and select the grid, as shown in the following screenshot:




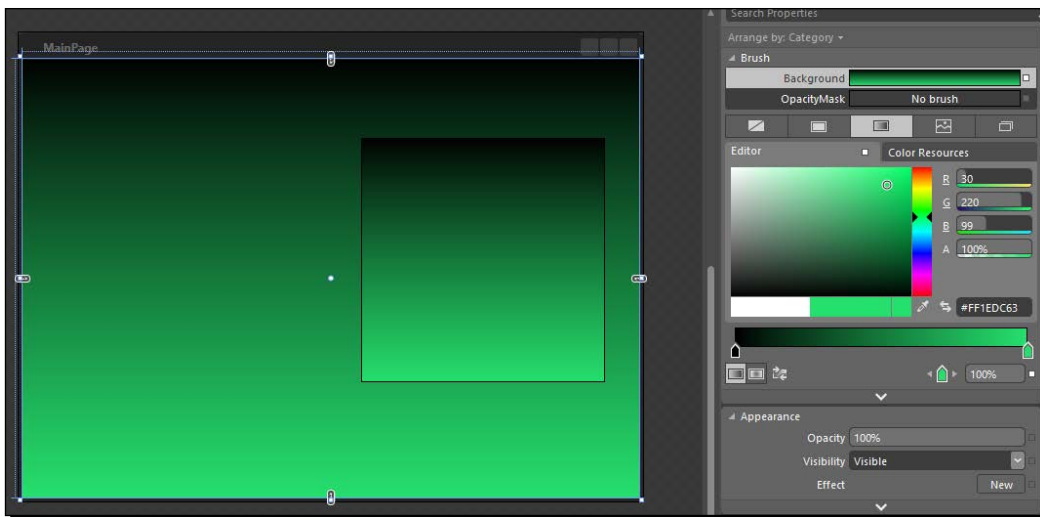
We will now see the set of options using which we can select the property, binding mode, source updated mode, and default value of the binding:

1. Let's bind the `Background` property of the grid to `RectangleProperty`, and let's then select **Binding Direction** as **OneWay**, as shown in the following screenshot:



- Let's change the `Background` property of the grid, and it will also change the background of the user control as we selected **Binding Direction** as **OneWay**. This is shown in the following screenshot:

 **TwoWay** binding is generally used in places where we want to pass the user input from the target element and present data back from the source.



- The XAML code for `RectangleBackground` would look somewhat like the following; we have specified the name of the element that we are using as the source of `DataBinding`:

```
Rectangle Fill="{Binding Background, ElementName=grid}"
```

What just happened?

We just performed `DataBinding` of the grid and `Rectangle`. We have bound the `Fill` property of the `Rectangle`, to the `Background` property of the grid, and whenever we change the `Background` property of the grid, `RectangleBackground` will also change.

Using DataSource

You have learned how to DataBind with one property at a time. The framework also provides us with the capability of DataBinding to DataSource like a collection as well. This type of DataBinding is applicable to ItemControls, such as ListBox, ComboBox, ListView, and so on. We need to set ItemSource and ItemsTemplate of these controls, and we can see the values of the collection populated in these controls.

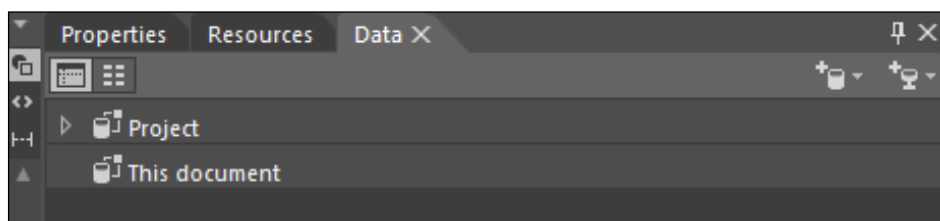


ListBox, ComboBox, and ListView are ItemControls present in WPF that are useful when showing a collection of items. More details on ItemControls can be found at [https://msdn.microsoft.com/en-us/library/system.windows.controls.itemcontrol\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.windows.controls.itemcontrol(v=vs.110).aspx).

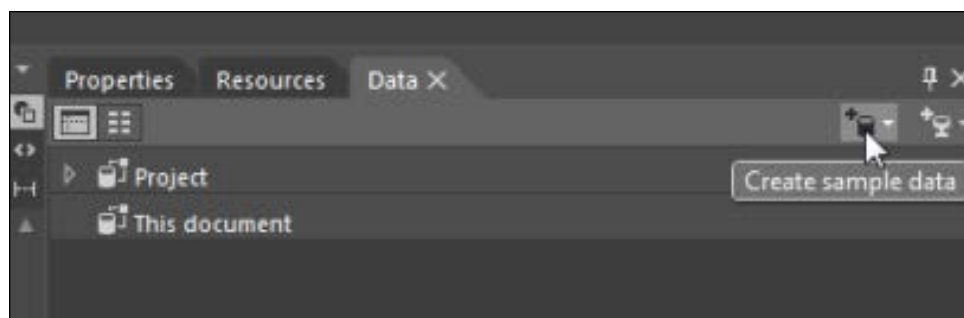
Time for action – DataBinding to DataSource as a collection

Blend provides us with the capability to create DataSource for our application:

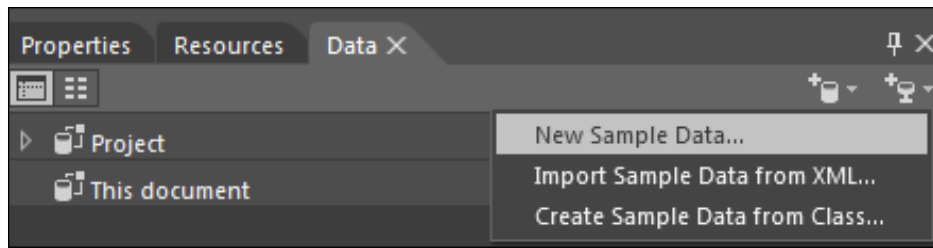
1. To create DataSource, let's move to the **Data** panel, which is alongside the **Properties** and **Resources** panel, as shown in the following screenshot:



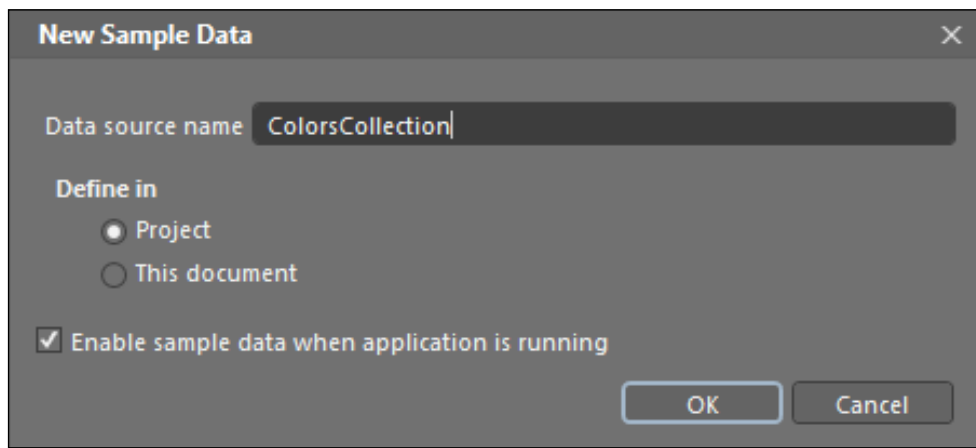
2. Click on the **Create sample data** icon, as shown here:



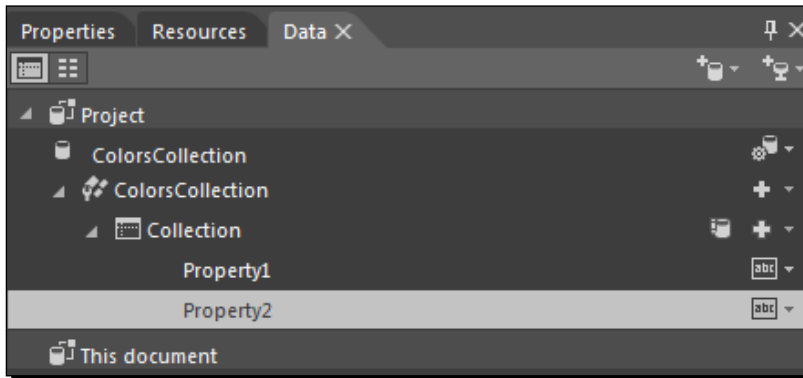
3. Once we do that, we will see the option to create **New Sample Data...**, **Import Sample Data from XML...**, and **Create Sample Data from Class...**. To create **Create Sample Data from Class...**, all we need to do is select the class that we want to import data from, and the same is the case with **Import Sample Data from XML...**. All we need to do select the XML file from which we need to import the data. However, we will have a look at **Importing Sample Data from XML...** in the next section. For now, just go ahead and select **New Sample Data...**, as shown in the following screenshot:



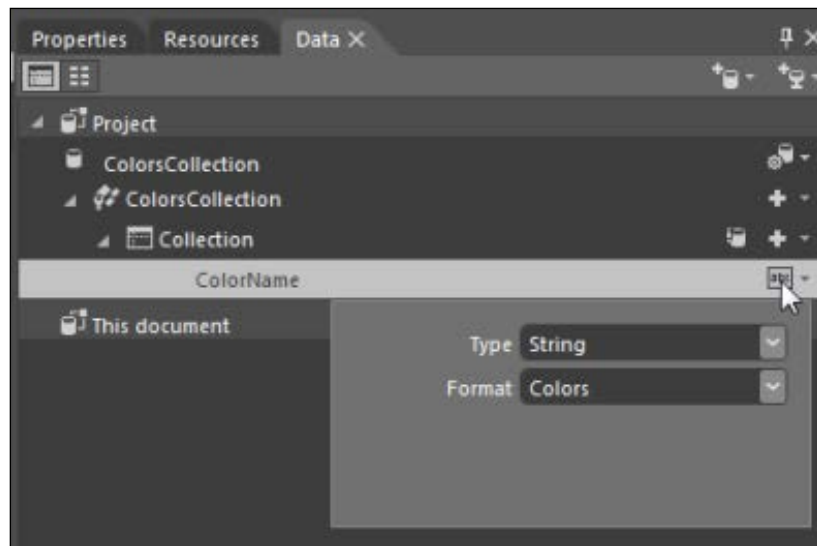
4. We now see a popup in which we can name DataSource and also select the scope of DataSource, which can be a whole project or just the document. Apart from this, we see the **Enable sample data when the application is running** option. We will select this option so that we can see the data even when the application is running. The following screenshot depicts this:



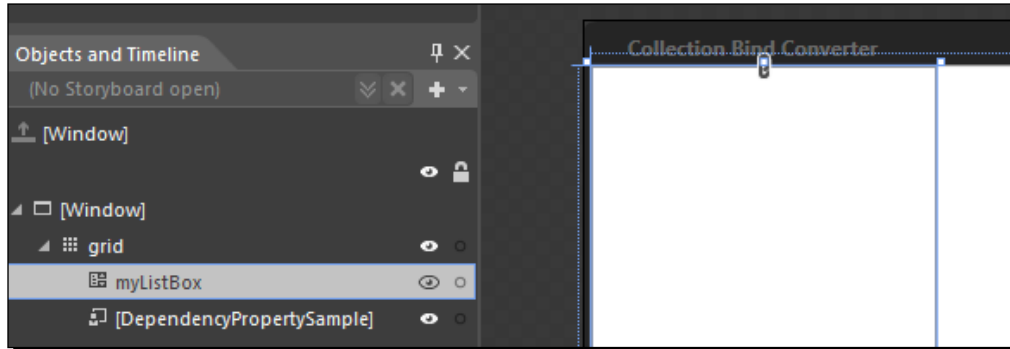
- When we click on **OK**, we see something like the following in the **Data** panel. Right click on **Property2**, delete it, and rename **Property1** to **ColorName**, as shown here:



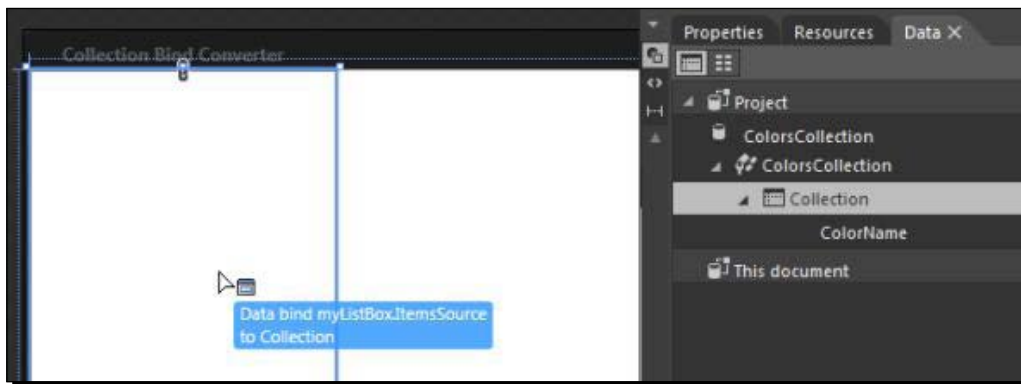
- Now, click on the **abc** icon next to the property name, and you will find multiple options to set those values. We will select **Type** of the property as **String** and **Format** of the property as **Colors**. This will create a sample collection of colors in the background that we will use. This is depicted here:



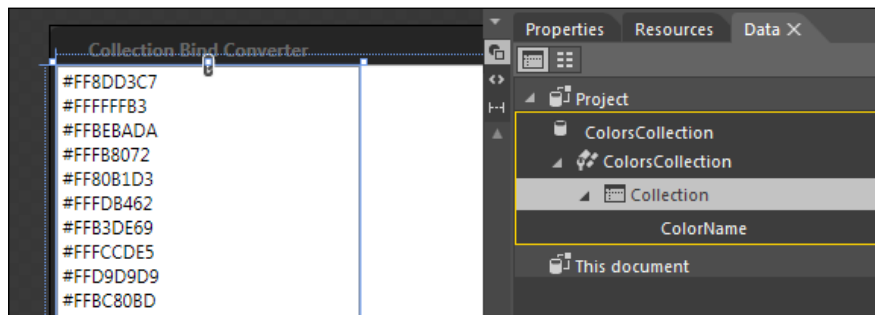
7. Now, go to the **Assets** panel, add **ListBox** on the art board, and rename it `myListBox`. This is encapsulated in the following screenshot:



8. Now, go back to the **Data** panel, select **Collection**, and drag it onto **ListBox** that we just added, as shown here:



9. When we do that, we see a couple of things. **ListBox** is a list of color codes, and we see an orange rectangle across **ColorsCollection** as it is bound to data now. The following screenshot shows this:



What just happened?

We just bound the data of the `DataSource` collection to `ListBox`.

To better understand what is going on in the background, let's have a look at the XAML code. There are a few interesting things to note that Blend has done for us in the background:

- ◆ In `Windows.Resources`, it added `DataTemplate` for `ListBox`. `DataTemplate` is useful to show the same kind of data over and over or show the same data at multiple instances. In this case, `DataTemplate` simply contains `TextBlock` inside `StackPanel` to show the code of the color being displayed.
- ◆ `DataTemplate` is assigned a key, which is used in `ListBox` to assign it as `ItemTemplate` for `ListBox`. This is how items of `ListBox` will be displayed.
- ◆ We also see that Blend has specified `ItemsSource` of `ListBox` as a collection, which is the same collection we bind to the `ListBox`.



There is also a target property named as `items`, but it is not a dependency property, so we have used `ItemsSource` instead. If we use the `Items` property, we would not be able to use features of dependency properties like dynamic updates.

- ◆ The last but actually the most important thing to note here is `DataContext`, which is set on the grid. `DataContext` is set as a whole `DataSource ColorsCollection`. With the use of data context, we can hook up multiple controls to share the same source for `DataBinding`. The data context is a property of the grid. If the source property is not set in `DataBinding`, then control looks for `DataContext`. But there is no `DataContext` for `ListBox`, but for the grid. `DataContext` is an inherited property, and it flows down to the descendants, so `TextBlocks` inherits the data context from the grid. This `DataContext` becomes the implicit source of these bindings, and the bindings just specify the path. The following code demonstrates the usage of the concepts discussed here:

```
<Window.Resources>
  <local:ColorConverter x:Key="converter" />
  <DataTemplate x:Key="ItemTemplate">
    <StackPanel>
      <TextBlock Text="{Binding ColorName}"/>
    </StackPanel>
  </DataTemplate>
</Window.Resources>
<Grid x:Name="grid"
  DataContext="{Binding Source={StaticResource
  ColorsCollection}}">
```

```
<ListBox Name="myListBox" HorizontalAlignment="Left"
SelectionMode="Extended" Width="200"
ItemTemplate="{DynamicResource
ItemTemplate}" ItemsSource="{Binding Collection}" />
<Rectangle HorizontalAlignment="Left" Margin="491,57,0,0"
VerticalAlignment="Top" Fill="{Binding Background,
ElementName=grid,
Mode=TwoWay}" />
</Rectangle>
</Grid>
```

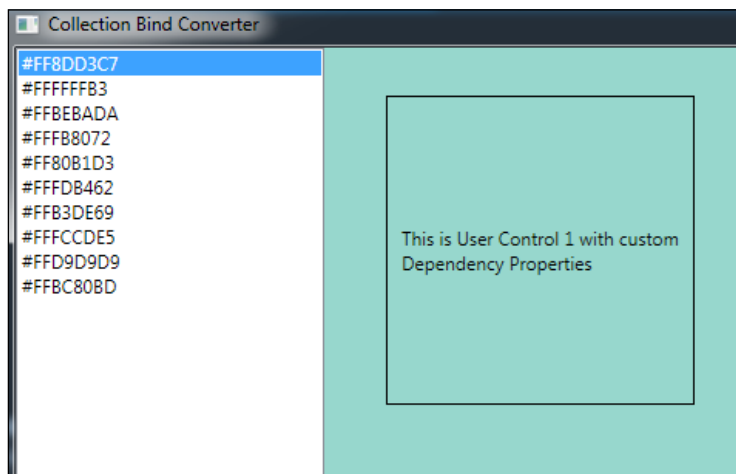
Time for action – DataBinding the background with SelectedValue

Let's go ahead and DataBind the Background property of the grid to a selected value of ListBox so that whenever we change the selection, the background of the grid will change:

1. To do that, we will need to add the following code to the grid control. Here, we are specifying that we are binding to the ColorName property of SelectedValue, and the element we are binding to is myListBox. Also, we need to specify the value converter, which will convert a string value to color. Here's how we can do this:

```
Fill="{Binding SelectedValue.ColorName, Converter={StaticResource
converter}, ElementName=myListBox}"
```

2. Now, when we run the application, we will see that as we change the selected value in ListBox, the background color of the grid changes as well. This is shown in the following screenshot:



What just happened?

We DataBound the background color property of the grid to the selected value property of ListBox.

Pop quiz

Q1. How many modes are there for DataBinding?

1. 3.
2. 4.
3. 5.
4. 6.

Q2. Is it possible to use a property of a control as a DataSource?

1. Yes, as long as it is a dependency property.
2. Yes, as long as it is a CLR property which implements `INotifyPropertyChanged`.
3. Yes, as long as it is a dependency property or a CLR property which implements `INotifyPropertyChanged`.
4. Yes we can use any property.

Summary

In this chapter, we looked in detail at what DataBinding is and how it works. We had a look at the ways in which we can do DataBinding in our application. We also had a look at various kinds of DataSource that we can use for DataBinding.

In the next chapter, we will have a look at vector graphics and try to understand how they work.

8

Vector Graphics

The applications that we design and develop end up running on diverse devices with different screen sizes, resolutions, and pixel densities. While developing our applications, we need to carefully consider the target devices for our application. Vector graphics allows us to create graphics that work on screens of multiple resolutions without loss of quality.

In the previous chapter, we had a look at DataBinding. In this chapter, we will have a look at the following topics:

- ◆ Raster and vector graphics
- ◆ Shapes
- ◆ Pen, Pencil, and Path


An introduction to vector graphics

When we are designing our applications, we generally use two types of graphics assets: raster graphics (pixel-based graphics) and vector graphics (mathematical function-based graphics).

Raster graphics

By raster graphics, we mean images composed of pixels. A pixel is a dot that is the smallest controllable element onscreen and is the basic unit of display for a raster graphic. The issue with raster graphics is that, when scaling happens (the number of pixels in the source image and the number of pixels used to display it), the graphic does not look good. The following is a Windows logo, the original resolution of which is 62 x 62 pixels, but when we scale it up, we see that the image becomes pixelated.



 We could use vector graphics for all the graphics in the application, but, if you still need to use raster graphics, it's recommended that you include images of the highest-supported resolution so that the scaling up of images does not cause pixelation.

Vector graphics

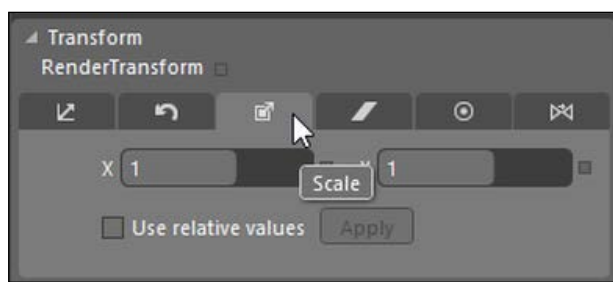
The resolution of computer monitors is improving, and the applications that we develop still have to look great on devices with different resolutions. Vector graphics allows us to do this because vector graphics uses geometrical functions to display these illustrations as opposed to pixels used to display bitmaps. The output device does not need to make any effort when the scaling happens because the geometrical functions are used to do the scaling. We could use points, lines, curves, paths, and shapes to create vector illustrations.

Time for action – zooming in to a WPF control

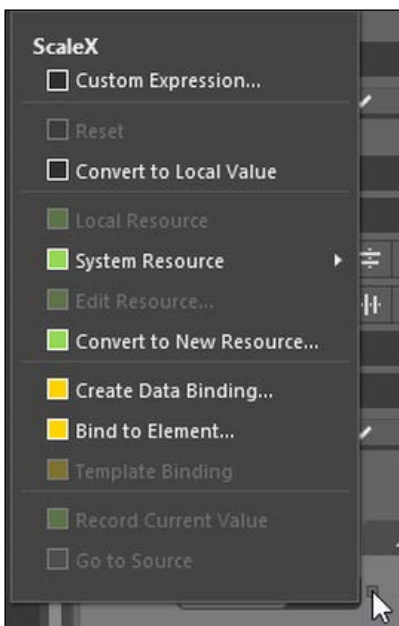
1. Create a new WPF application and name it `Chapter08`. Now, drag and drop a button onto the art board and give it some style, as shown here:



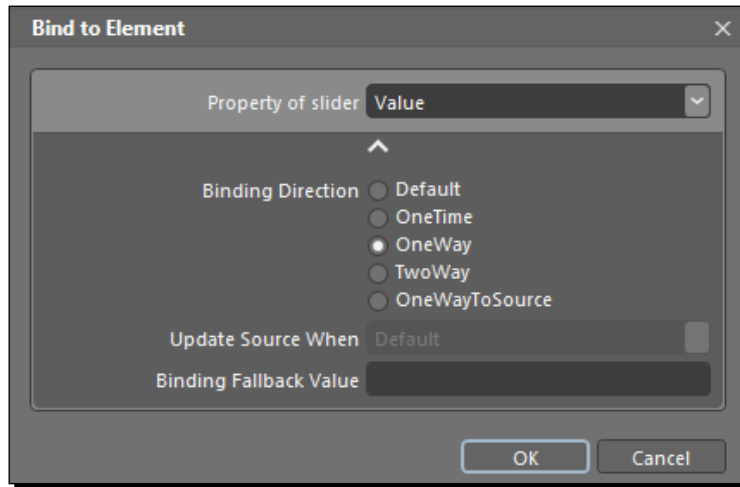
2. Now, go to the **Projects** tab, rename `MainWindow.xaml` to `ButtonZoom.xaml`, and press `ctrl + shift + s` to save all the documents that we modified. Now drag and drop a slider control onto the bottom of the grid. With the slider selected, move to the **Properties** panel and change the minimum and maximum values of the slider to 1 and 5 respectively.
3. We will now data-bind these values to the **ScaleX** and **ScaleY** properties of the button. The **ScaleX** property sets how much the object is stretched or shrunk on the x axis and the **ScaleY** property sets how much the object is stretched or shrunk on the y axis. Let's go ahead and select the button, move to the transform section in the properties section, and select scale, as shown in the following screenshot:



4. Click on the small square next to the value of x and select **Bind to Element...**. This is shown in the following screenshot:

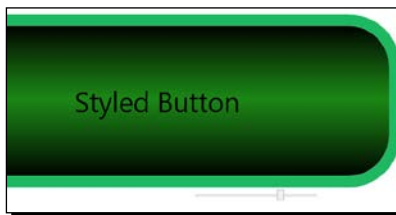


5. Select the slider control, and we will see a popup that asks us to select the property of the slider with which we want to bind the scale property of the button. We will see multiple options there. For now, we will just use the **OneWay** option. This is shown in the following screenshot. Do the same thing for **ScaleY** as well.



What just happened?

Both the ScaleX and ScaleY properties of the button are now bound to the value of the slider, and now when we run the application and move the slider, we will see that the button scales up and down, but there is no difference in the quality of the button graphics. We have seen that WPF controls use vector graphics to render themselves.



One of the good practices to follow when designing applications targeted at multiple screens is to have a minimum resolution to start with. It helps in defining the baseline for the content structuring of the application. Generally, 1024 x 768 is the resolution that is used as the minimum baseline for many applications. If we do not have a minimum resolution requirement for our application, then the application layout could truncate or even break in some scenarios.

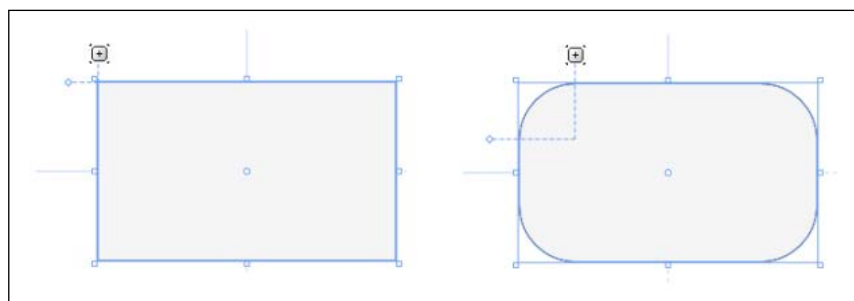
Shapes

When creating the design for our application, we might need to create elements with different shapes, sizes, and colors. We could draw these shapes on the art board, and any shape that we draw on the art board becomes an object. We can draw a shape by selecting either a rectangle or an ellipse from the toolbox, moving to the art board, and dragging the mouse from one point to the other point. These shapes are vector objects.

Time for action – adding a shape

Move to the **Projects** panel, right click on the project, and select **add new item**. Select **UserControl** and name it `ShapesDemo.xaml`. Go to the **Assets** panel and click on the rectangle. Now, move to the art board, press the mouse's left button, and while the mouse's left button is still pressed, drag the mouse across the art board to draw a rectangle.

We now have a shape that we just drew, and we can resize it by grabbing one of the eight tiny squares present in the four corners of the rectangle and at the center of each of the sides of the rectangle. Also, we can hover the mouse over one of the squares present in the top-left corner of the rectangle and the mouse pointer will change to a **+** sign. Then, we can use this to change the corner radius of the rectangle. This is shown in the following diagram:



If we press the *Shift* key and then move the radius handle, it would modify the individual radii and not both *x* and *y*.

The XAML code for the rectangle shape would look similar to the following code:

```
<Rectangle Height="173" Width="308" RadiusY="32.5" RadiusX="32.5"/>
```

We can modify `RadiusX` and `RadiusY` in XAML as well.

The various other shapes that are available in Blend include Ellipse, Line, Path, Polygon, and Polyline. These objects help us create different shapes. More details on these shapes can be found at [https://msdn.microsoft.com/en-us/library/vstudio/System.Windows.Shapes\(v=vs.100\).aspx](https://msdn.microsoft.com/en-us/library/vstudio/System.Windows.Shapes(v=vs.100).aspx).

What just happened?

We just created a shape, resized it, and then changed the corner radius of that shape.

Importing graphics

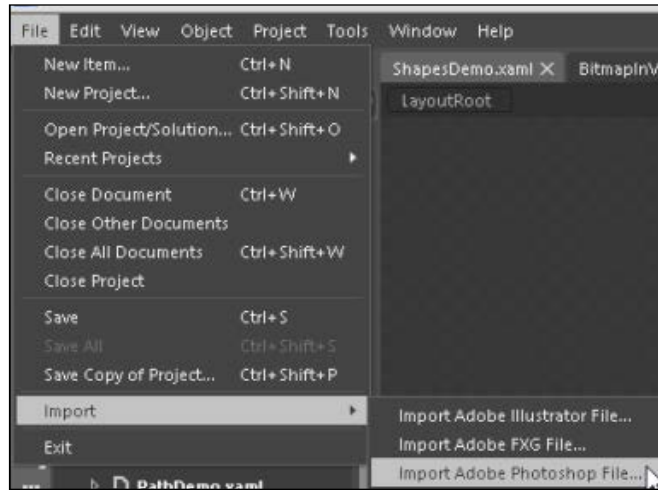
We can import Adobe Illustrator files (.ai), Adobe Flash XML Graphics files (.fxg), or Adobe Photoshop files (.psd). These files are imported into the currently open document. While importing Adobe files, we have the option to view and select the layers we want to import.

The features of these files are imported in the following formats:

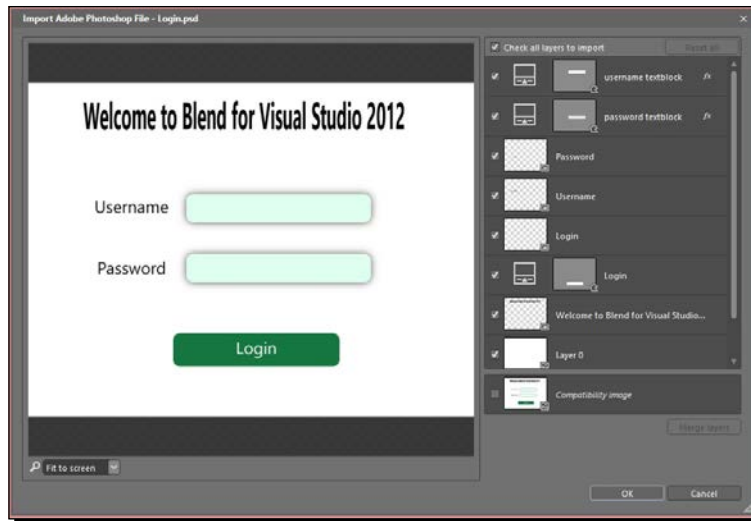
- ◆ **Layers:** Blend imports each layer as an individual object acting as a layout container. Blend keeps the name of the layer the same as in the Adobe file, but we have the option to change it.
- ◆ **Text:** Text could be imported as bitmap images or editable objects.
- ◆ **Vectors:** Vector objects can be imported as Path object or images.
- ◆ **Gradient:** Linear and radial gradients are imported as editable, and any other fill is rasterized. Color stops and opacity stops are imported as gradient stops in the fill and OpacityMask properties respectively.
- ◆ **Patterns:** Any patterns are imported as image brushes.

Time for action – importing graphics

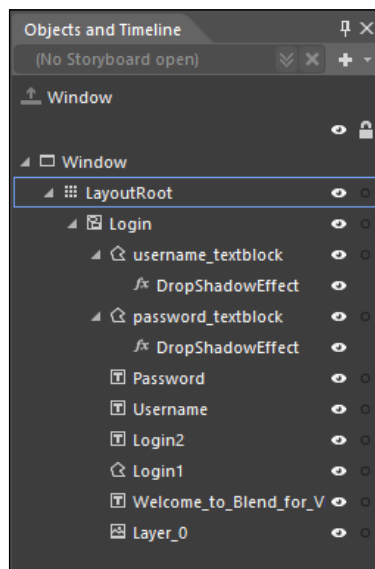
We use Photoshop for the import. To import a .psd file, navigate to **File | Import | Import Adobe Photoshop File...** and select the .psd file provided in the sample code or any of your own .psd files. This is shown in the following screenshot:



When we do that, we will see the option to import the selected layers or one compatible, merged image of the entire design. Make sure **Check all layers to import** is selected and that all the layers are selected as well. This is encapsulated in the following screenshot:



Once the file is imported, we will see a Canvas as the root element, the name of which is the same as the imported file. The graphics on the art board is not made up of XAML elements, which we can also see in the **Objects and Timeline** panel. This is depicted in the following screenshot:



What just happened?

We have imported graphics, the elements of which are now converted into XAML elements, so we can use and modify these objects according to our requirements.

The Line, Pen, and Pencil tools

We can draw paths using the line, pen, and pencil objects.

Line

The Line tool is the simplest tool available to create a path. Using the Line tool, we can draw a path between two points.

Pen

With the line object, we only have the option to draw a straight line. The maximum that we can do is apply different kinds of transforms on the line object, such as rotate, skew, scale, and so on. If we want to draw a shape in one flow, a more useful tool will be Pen. The two most used properties of Pen are brush and thickness. We can also use the end cap style when we are using the line as a directional arrow.

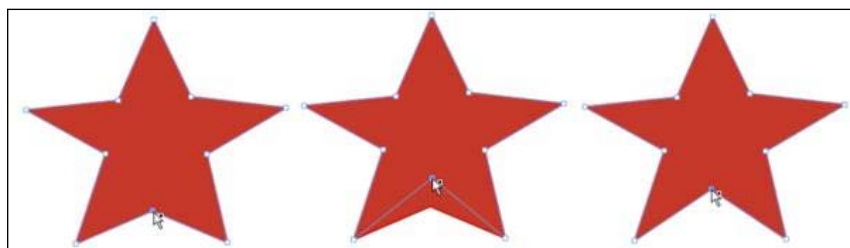
Time for action – creating a shape using Pen

- 1.** Select the pen object from the toolbox. Now, click anywhere on the art board, and then click on another point; that will draw a line. The first point that we click on the art board with the Pen tool selected is the starting point of the drawing. Thereafter, any point that we click on creates a line between the last point and the new point. If we click and, without releasing the mouse button, drag the point, then it would create a curve, and the more we drag, the steeper will be the angle of the curve. This is shown in the following image:



- 2.** Now, wherever we click on the art board, a new line will be formed with the last clicked point as the start point and current clicked point as the end point. So, let's start clicking at points so that we start forming a path shaped as a star. This is shown in the preceding image.

3. Let's continue adding points onto the art board so that we create a path in the form of a star. We can finish this path formed using the Pen tool either by clicking on the first point or by selecting from the toolbar.
4. Don't worry if we are not able to create the shape at first go. We can select any of the points that we created and move them around so as to get the desired shape.



5. The XAML code for the star shape we created would look somewhat as shown here. It produces a path with the points. The `Data` property of a `Path` object defines the shape's geometry. The following is the aforementioned XAML code:

```
<Path Data="M264,98 L225,186 124.33333,196.33333 211,244
178.50028,342.00043 262.50018,286.00033 344.5001,340.50043
322.00012,244.50026 400.50004,194.00017 304.66667,182 z"
Fill="#FFBF2E20" HorizontalAlignment="Left" Height="246"
Margin="172.833,96,0,0" Stretch="Fill" Stroke="#FFC70E0E"
StrokeThickness="2" VerticalAlignment="Top" Width="278.167"/>
```

6. We can use either `StreamGeometry` or `PathGeometry` when we describe `Path`. These are the two mini-languages that are used to define geometric paths.
7. In the preceding code, we have used `StreamGeometry`. `StreamGeometry` is a lightweight alternative to `PathGeometry` as it does not support `DataBinding`, animation, or modification.
8. `PathGeometry` represents a complex curve that can be composed of arcs, curves, and shapes. We could represent the above path in `PathGeometry` as follows.

```
<Path Fill="#FFBF2E20" HorizontalAlignment="Left" Height="246"
Margin="172.833,96,0,0" Stretch="Fill" Stroke="#FFC70E0E"
StrokeThickness="2" VerticalAlignment="Top" Width="278.167">
  <Path.Data>
    <PathGeometry Data="M264,98 L225,186 124.33333,196.33333
211,244 178.50028,342.00043 262.50018,286.00033
344.5001,340.50043 322.00012,244.50026 400.50004,194.00017
304.66667,182 z">
  <Path.Data>
</Path>
```



We should use StreamGeometry when we don't need to modify the path after creating it and use PathGeometry when we need to modify the path. More information on this can be found at [https://msdn.microsoft.com/en-us/library/ms751808\(v=vs.100\).aspx](https://msdn.microsoft.com/en-us/library/ms751808(v=vs.100).aspx).

What just happened?

We created a star-shaped path using the Pen tool. Also, we modified the path to reach the end goal of our shape.

Pencil

So, we just created shapes that were composed of lines; however, if we want to create a freeform shape or drawing, then the Pencil tool is our friend.

We can draw any freeform path using the Pencil tool just like we would draw on any drawing board. When we draw a shape using a Pencil, a Path is produced.

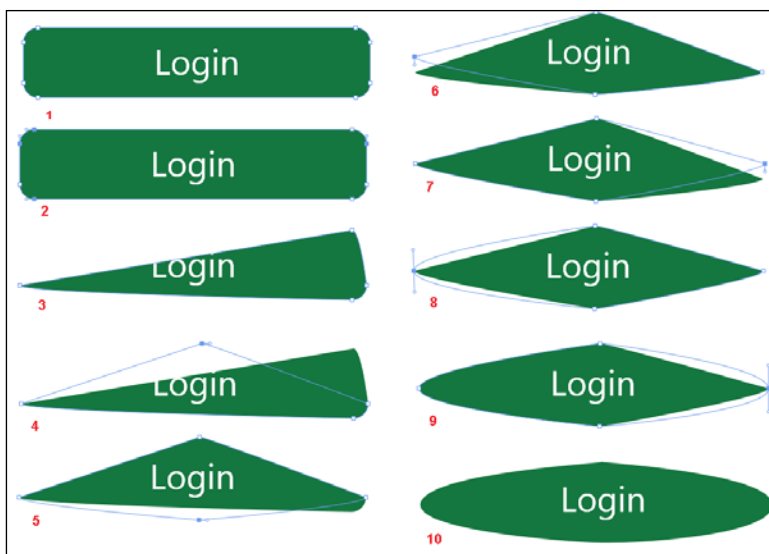
Paths

Paths are vector objects made up of a series of curves and lines. Paths are the most versatile vector objects available in WPF and Silverlight as using them, we can create any shape, drawing, or graphic using paths.

Working with shapes and shape operations might not provide the level of control or granularity that we might want in our design, but using Paths, we have much more control over the design that we are making.

Time for action – modifying a Path

1. Click on the **Login** button on the **Login** screen that we imported from the .psd file. Follow the steps specified in the following image:



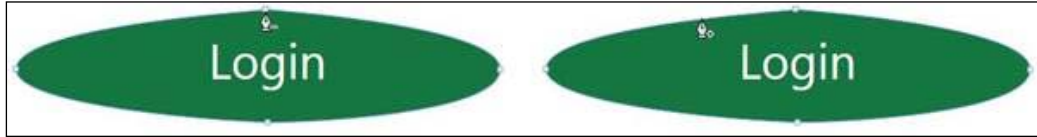
2. Select the **Login1** path with the **Direct Selection** tool already selected.
3. Select four of the eight points by pressing the *Ctrl* key and clicking the mouse button.
4. Press *Delete* to delete the four points.
5. Drag the point in the top-right corner to the center.
6. Drag the bottom-right point to the center.
7. Drag the left-hand side point to the center.
8. Drag the right-hand side point to the center.



In this case, we wanted the segment to adjust (a segment is a line or arc between two points) in length, so we made the point move around, but if we wanted the segment to stay the same in size and shape and just change its placement, then we could even move only the segment.

9. Press the *Alt* key, press and hold the mouse's left-hand side button on the left-hand side point, and drag it a little upward. When we do this, we see that the Bezier handle associated with that point is displayed.
10. Press the *Alt* key, press and hold the mouse's left button on the right-hand side point, and drag it a little upward. Once the Bezier handles are visible, we can select and move them too. When we move one Bezier handle, the opposite one also moves along with it so that the curve stays smooth. If we want to move only one Bezier handle, we could do that by pressing the *Alt* key and then moving the individual Bezier handle.

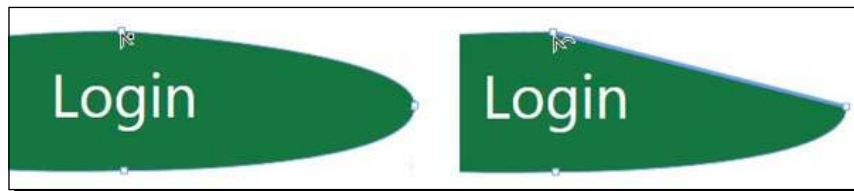
- 11.** Click outside the Login path.
- 12.** With the Pen tool selected, we can remove a point by simply clicking on it and also add a new point by clicking anywhere on the existing path. In the following image, you can see that when we do a mouseover on an existing point, we see a small - icon and the + icon when we hover the mouse anywhere else on the path. This is shown in the following image:



- 13.** When we see the cross sign, it means that no changes will be made to any existing path but a new point will be created. This is shown in the following image:



- 14.** To remove the Bezier curve from a point, we need to press the *Alt* key and then click on the point. As we can see in the following image, when we do a mouseover on a point while pressing the *Alt* key, we see that the mouse cursor changes. This is shown in the following image:



- 15.** We can also bend a segment by pressing *Alt*, clicking on the segment, and then moving it. Straighten the segment using *Alt + Click*. The following is the output of this step:



- 16.** The XAML code for this button would be a path that would look similar to the following; we have a path with multiple data points:

```
<Path x:Name="Login1" Data="F1 M125.00075,0.02538874
C125.00075,0.02538874 242.11804,5.4829142 242.55902,29.754374
243,54.025834 143.86706,55.983359 126.05902,55.754374
126.05902,55.754374 -0.49752824,61.024619 0.0014717614,29.525056
0.50047176,-1.974507 125.00075,0.02538874 125.00075,0.02538874
z" Fill="#FF0F6C36" Height="56.086" Canvas.Left="207.5" Canvas.
Top="357.974" Width="242.56"/>
```

What just happened?

We modified the path, which was in the shape of a rectangle, into an ellipse. We also saw how we can add and remove points and arcs.

BitmapScalingMode

In WPF 4.0, Microsoft has changed the default quality of the images from high quality to low quality to improve the performance of the applications. So, if we want the image to be displayed in high quality, then we would set the `BitmapScalingMode` property on the image in XAML as shown here:

```
<Image RenderOptions.BitmapScalingMode="HighQuality" />
```

The various values that could be set as `BitmapScalingMode` and their details can be found at [http://msdn.microsoft.com/en-us/library/system.windows.media.bitmapscalingmode\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/system.windows.media.bitmapscalingmode(v=vs.110).aspx).

DPI awareness

Writing a DPI-aware application is the key to making a UI look consistent across a wide variety of DPI display settings. An application that is not DPI aware but is running on a high-DPI display, or across monitors of different DPIs, will be scaled by the system to the appropriate size so that it is still usable but can suffer from visual artifacts, including incorrect scaling of UI elements, clipped text, and blurriness. By making our applications DPI aware, we can present our application's UI in a predictable manner.

There are three categories that our applications would fall in depending on the DPI awareness of the application:

- ◆ **Not DPI-aware applications:** These applications always render at the lowest desktop DPI, that is, 96 DPI. This class of applications is unaware of different system DPIs. **Desktop Window Manager (DWM)** virtualizes and scales these applications to account for high DPI.
- ◆ **System-DPI-aware applications:** These applications are considered DPI aware and render at the system DPI to avoid being scaled. These applications do so on a system-DPI level rather than the per-monitor-DPI level because they cannot respond to dynamic changes in DPI during a single session. System-DPI aware applications render optimally on the primary display, and DWM does not scale and virtualize them. However, if the user moves the application to a display with a higher or lower DPI than that of the primary screen, DWM scales it up or down.
- ◆ **Per-monitor-DPI-aware Applications:** These applications dynamically scale up or down when a user changes the DPI or moves the application between monitors that have different DPIs. These applications always render crisply and at the correct size for a given display. DWM does not scale and virtualizes this class of applications.

The recommended set of DPI settings and minimum resolutions to consider when testing DPI awareness levels is shown in the following table:

DPI setting	Minimum resolution
96 (100%)	1024 x 720
120 (125%)	1280 x 960
144 (150%)	1536 x 1080
192 (200%)	2048 x 1440

More details on developing DPI-aware applications can be found at <http://msdn.microsoft.com/en-us/library/windows/desktop/dn469266%28v=vs.85%29.aspx>.

Pop quiz

Q1. What are the different shapes available in WPF?

1. Ellipse & Rectangle.
2. Line & Path.
3. Polygon & Polyline.
4. All.

Q2. Which tool can I use to draw a path?

1. Pen.
2. Pencil.
3. Both.

Summary

In this chapter, we looked into the details of the graphics tools and elements in WPF and Silverlight. We used various vector tools available in Blend and created graphics.

In the next chapter, we will take a look at user controls and custom controls, how they work, and how we can use them well in our applications.

9

User Controls and Custom Controls

In Windows Presentation Foundation, controls are based on the concept of composition, meaning we can add almost any control as the content of other controls. User controls allow us to create a reusable group of controls and partition the application into smaller, reusable blocks.

Custom controls are extensions of existing controls with additional properties or behaviors. The custom control gets its look from the control template defined. The custom control can also have a custom behavior.

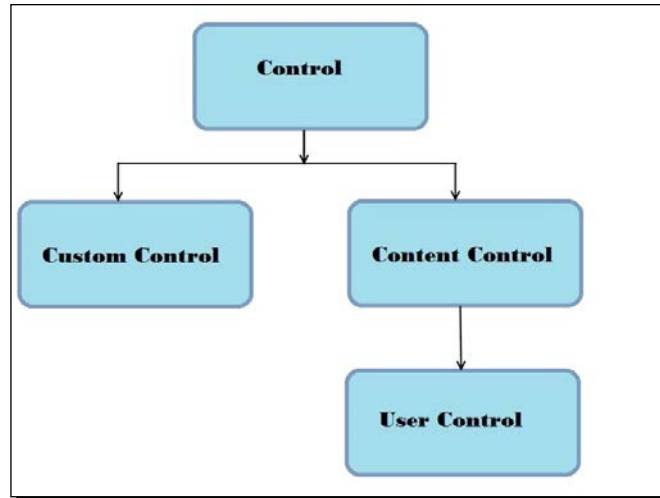
In the previous chapter, we had a look at vector graphics, paths, geometry, and shapes.

In this chapter, we will have a look at the following topics:

- ◆ User controls
- ◆ Custom controls

User controls and custom controls can encapsulate a portion of the UI and behavior and allow us to create maintainable and reusable controls. We can use multiple instances of a user or custom control inside the same window. The major usability of these controls lies in reuse; however, these could also come in handy when multiple people work on the same project.

Custom Controls derive from the `Control` class or any of its subclasses. `UserControls` derive from the `UserControl` class, which derives from `ContentControl`, which in turn derives from `Control`. So, technically, all user controls are custom controls.



User control or custom control – which to use and when

Understanding user controls and custom controls and how to create them is good, but we also need to know when and where to use them. There is no rule of thumb, but there are certain guidelines that we could follow to determine which control to use. Before we decide to create a control, we should ask the following questions to ourselves:

- ◆ Is the same capability provided by an existing control? – This might save you the effort of developing a new control altogether
- ◆ Why do I need this control? – This will help you in determining the functionality of the control
- ◆ Can I achieve the same functionality by changing the style of the control? – This might trim down the amount of work you need to do in developing the control
- ◆ Who are the end users of the control? – This will help you in streamlining the requirements of the control
- ◆ Is there a control that provides a part of the functionality? – This will help us in determining the base class of the control and also give a starting point to develop the control

- ◆ Can this be achieved by combining existing controls? – This will help you determine which user control or custom control you need to build

User control	Custom control
This is used where a control that is composed of a collection of controls is needed.	This is used when an already existing control needs to be customized and extended
This cannot be styled/templated, but the controls that it contains can be styled/templated	This can be styled/templated.
This might not provide seamless, look-less behavior as it is a collection of controls and they might not look perfect together.	This could provide seamless, look-less behavior.
Easier to design as user control is derived from <code>ContentControl</code> , and have <code>ContentTemplate</code> , but doesn't have <code>ControlTemplate</code> .	This is comparatively complex to design. These controls derive from the <code>Control</code> class, have <code>ControlTemplate</code> and don't have <code>ContentTemplate</code> .
Example: Background color selector	Example: Switch control

The differences mentioned in the preceding table will help you choose between user controls and custom controls. We will use the preceding table and questions when developing user controls and custom controls.

Understanding and creating a user control

A user control can contain a layout control, which, in turn, can contain any number of controls, resources, and animation timelines. A user control has its own code-behind file. A user control is a whole unit in itself and is reusable. We generally create user controls when we need a reusable control composed of several other controls.

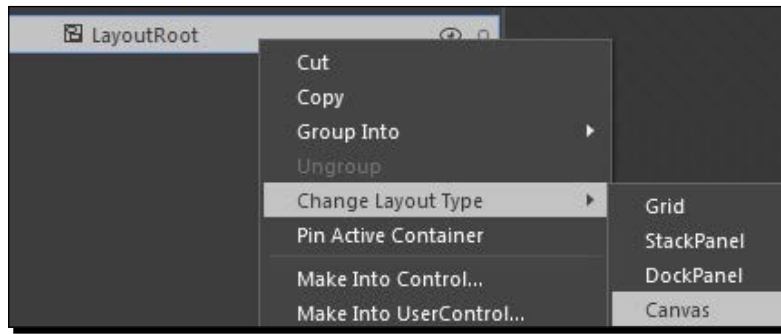
We want a control that allows the user to select the background of the application. So, we will ask ourselves a few questions:

- ◆ Is the same capability provided by an existing control? – No
- ◆ Why do I need this control? – To allow the user to choose a color at runtime
- ◆ Can I achieve the same functionality by changing the style of the control? – No
- ◆ Who are the end users of the control? – All application users
- ◆ Is there a control that provides a part of the functionality? – No
- ◆ Can this be achieved by combining existing controls? – Yes

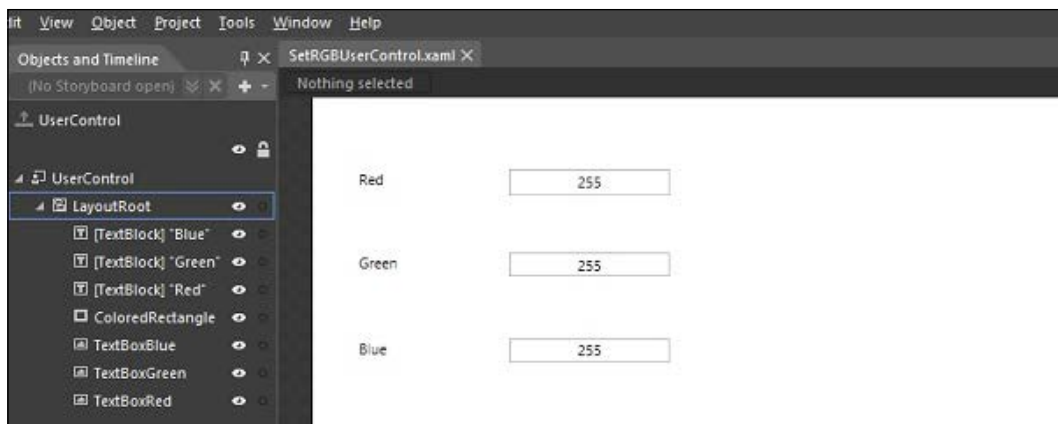
Time for action – creating a user control that selects the background color

To create a user control that selects the background color, perform the following steps:

1. Open Blend, create a new WPF project, and name it `Chapter09`. In the **Projects and Timeline** panel, right-click on the project and select **Add New Item**. Select **UserControl** and name it `SetRGBUserControl.xaml`. We append **UserControl** to the name of user controls as a convention followed in WPF applications.
2. Right-click on **LayoutRoot** and select **Change Layout Type | Canvas**, as shown in the following screenshot:



3. We will design a user control to set the RGB (red, green, and blue) values of the color we want. Drag and drop three **TextBlock** objects and three **TextBox** objects onto **Canvas**. Resize and reposition them as shown in the following screenshot:



4. Change the text of TextBlock objects to **Red**, **Blue**, and **Green**. Change the text of TextBox objects to **255**. Rename the TextBox objects to **TextBoxRed**, **TextBoxGreen**, and **TextBoxBlue**. Here's the XAML code to help you:

```
<Canvas x:Name="LayoutRoot" Background="White">
  <TextBox x:Name="TextBoxRed" Height="22" Canvas.Left="60.5"
    TextWrapping="Wrap" Text="255" Canvas.Top="9" Width="132"
    HorizontalAlignment="Center" VerticalAlignment="Center"
    HorizontalContentAlignment="Center"
    VerticalContentAlignment="Center" TextAlignment="Center"
    TextChanged="TextBoxRed_TextChanged"/>

  <TextBox x:Name="TextBoxGreen" Height="19" Canvas.Left="60.5"
    TextWrapping="Wrap" Text="255" Canvas.Top="78" Width="132"
    HorizontalAlignment="Center" VerticalAlignment="Center"
    HorizontalContentAlignment="Center"
    VerticalContentAlignment="Center"
    TextChanged="TextBoxGreen_TextChanged"/>

  <TextBox x:Name="TextBoxBlue" Height="20" Canvas.Left="60.5"
    TextWrapping="Wrap" Text="255" Canvas.Top="147.5" Width="132"
    HorizontalAlignment="Center" VerticalAlignment="Center"
    HorizontalContentAlignment="Center"
    VerticalContentAlignment="Center" TextAlignment="Center"
    TextChanged="TextBoxBlue_TextChanged"/>

  <TextBlock Height="22" Canvas.Left="7" TextWrapping="Wrap"
    Text="Red" Canvas.Top="9" Width="97"
    HorizontalAlignment="Center"
    VerticalAlignment="Center"/>

  <TextBlock Height="19" Canvas.Left="7" TextWrapping="Wrap"
    Text="Green" Canvas.Top="78" Width="97"
    HorizontalAlignment="Center"
    VerticalAlignment="Center"/>

  <TextBlock Height="20" Canvas.Left="7" TextWrapping="Wrap"
    Text="Blue" Canvas.Top="147.5" Width="97"
    HorizontalAlignment="Center" VerticalAlignment="Center"/>
</Canvas>
```

You might have noticed that the names that we added to the controls are represented as `x:Name` attribute. This uniquely identifies object elements for access to the instantiated object from code-behind or general code. We need to add a name to the control when we want to access the control in the code-behind.

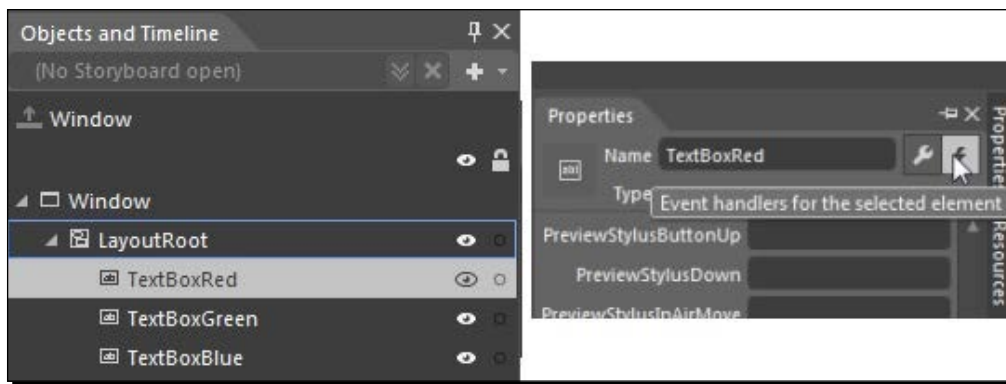
```
x:Name="TextBoxRed"
```

What just happened?

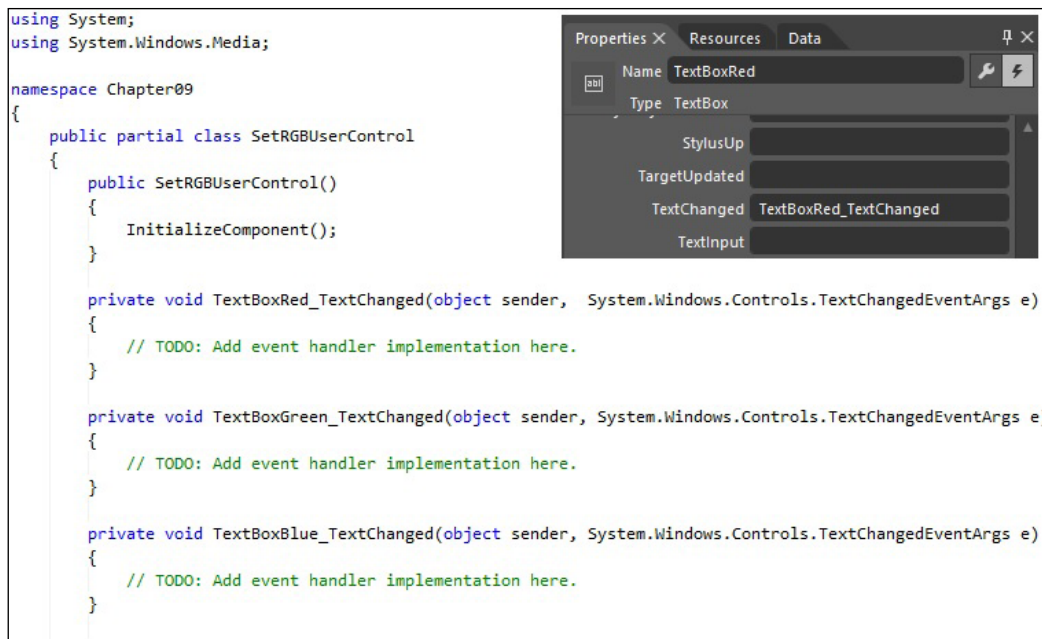
We have created the user interface to assign the RGB values. In a while, we will see this user interface in action.

Time for action – adding event handlers

The user interface that we created in the previous section does not have any behavior. We need to do some work to make it interactive. Select **TextBoxRed** and, in the **Properties** panel, click on the **Event handlers** section. This opens up the list of events available for the **TextBox** object. This is shown in the following screenshot:



Move down to the **TextChanged** event and double-click in the empty **TextBox** next to it. We will be taken to the code-behind page of the XAML. We will see that an event handler has been added to the **TextChanged** event of **TextBoxRed**. Repeat the steps to add event handlers to the other two **TextBox** objects **TextChanged** events as well. The following screenshot depicts this:



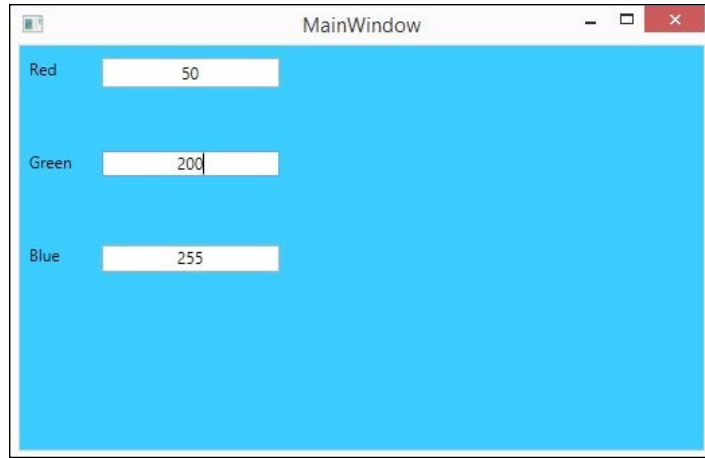
Now, add the code in these event handlers to change the fill of the rectangle as a combination of the red, green, and blue values mentioned in the `TextBox` objects. To that, we will add a method to change the fill of the rectangle and call that method from **TextChanged** event handlers. This is shown in the following code:

```
private void TextBoxRed_TextChanged(object sender,
    TextChangedEventArgs e)
{ SetBackground(); }
private void TextBoxGreen_TextChanged(object sender,
    TextChangedEventArgs e)
{ SetBackground (); }
private void TextBoxBlue_TextChanged(object sender,
    System.Windows.Controls.TextChangedEventArgs e)
{ SetBackground (); }
private void SetBackground ()
{
    byte redColor, greenColor, blueColor;
    if (Byte.TryParse(TextBoxRed.Text, out redColor)
        && Byte.TryParse(TextBoxGreen.Text, out greenColor)
        && Byte.TryParse(TextBoxBlue.Text, out blueColor))
    {
        LayoutRoot.Background = new
            SolidColorBrush(Color.FromArgb(255,
```



```
        redColor, greenColor, blueColor));  
    }  
}
```

Let's run the application and, as we change the values of **Red**, **Green**, or **Blue**, we will see that the fill of the rectangle also varies. This is shown in the following screenshot:

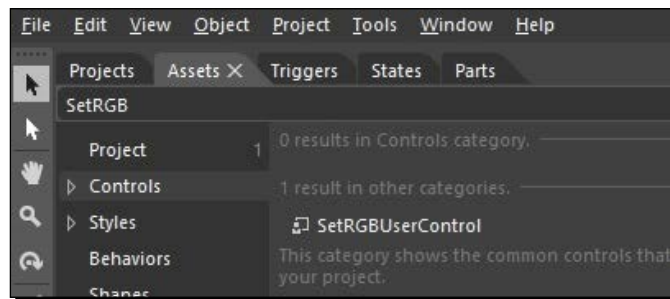


What just happened?

We have added event handlers to catch the **TextChanged** event on these **TextBox** objects, and, whenever the text changes, we recalculate the fill color of the background.

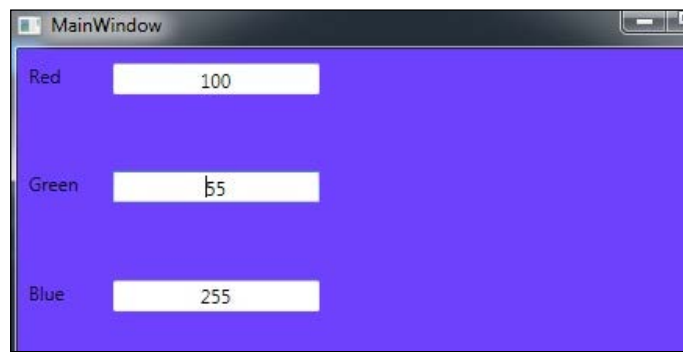
Time for action – adding a user control in a window

We could add user control in a window, page, or another user control. To add **SetRGBUserControl** to **MainWindow**, simply open **MainWindow** from the **Projects** panel. Then, Open **Assets** and **Search** for **SetRGBUserControl**. This is shown in the following screenshot:



Drag and drop the user control onto the art board. This will add a copy of **SetRGBUserControl** to the grid layout of `MainWindow.xaml`. We can also add the user control by dragging and dropping the user control onto the **Objects and Timeline** panel as well.

Run the application and we can see the user control as part of **MainWindow**, and when we change the values in the textboxes, the color of the rectangle changes. This is shown in the following screenshot:



When we look at the XAML code for `MainWindow.xaml`, we can see a namespace declaration added. The namespace refers the current `Chapter09` and has an alias `local`. **SetRGBUserControl** is available in this namespace, and we need to reference it in the XAML code to create an instance of **SetRGBUserControl**. This is encapsulated in the following screenshot:

```
SetRGBUserControl.xaml  MainWindow.xaml X  App.xaml
1 <Window
2   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6   xmlns:local="clr-namespace:Chapter09" mc:Ignorable="d"
7   x:Class="Chapter09.MainWindow"
8   x:Name="Window"
9   Title="MainWindow"
10  Width="640" Height="480">
11
12  <Grid x:Name="LayoutRoot">
13    <local:SetRGBUserControl HorizontalAlignment="Left"
14      Margin="37,10,0,0" VerticalAlignment="Top"/>
15  </Grid>
16 </Window>
```

What just happened?

We have added a user control to **MainWindow**.

Understanding and creating custom controls

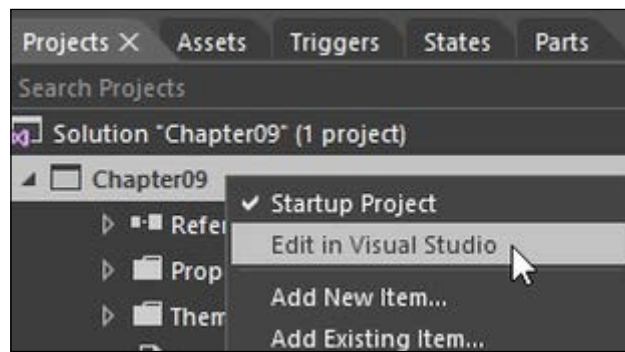
We would generally create a custom control when we need functionality that is not provided completely by any existing control, and, by extending the current control, we could achieve that functionality. So, we will ask ourselves a few questions.

- ◆ Is the same capability provided by an existing control? – No
- ◆ Why do I need this control? – To visually show the user the current state of the control
- ◆ Can I achieve the same functionality by changing the style of the control and not the behavior? – No
- ◆ Is there a control that provides a part of the functionality? – Yes. `ToggleButton`
- ◆ Can this be achieved by combining existing controls? – No

Time for action – creating a custom control

Perform the following steps to create a custom control

1. Right-click on the project name, `Chapter09`, in the **Projects** panel and select **Edit** in Visual Studio. It is easier to create a custom control in Visual Studio as it provides a built-in template to create custom controls. This is shown in the following screenshot:



2. In Visual Studio, right-click on the project name, `Chapter09`, in **Solution Explorer** and select **Add New Item....**

3. Select **CustomControl (WPF)** in the **Add New Item** window, and name it `SwitchCustomControl.cs`, as shown in the following screenshot:



4. When we click on **Add**, a new class appears that inherits from `Control`. Along with this, the following screenshot also shows the initial steps to define the custom control and use it.

```

SwitchCustomControl.cs
Chapter07.SwitchCustomControl
using System.Windows;
using System.Windows.Controls;

namespace Chapter0
{
    /// <summary>
    /// Follow steps 1a or 1b and then 2 to use this custom control in a XAML file.
    /// Step 1a) Using this custom control in a XAML file that exists in the current project.
    /// Add this XmlNamespace attribute to the root element of the markup file where it is
    /// to be used:
    ///     xmlns:MyNamespace="clr-namespace:Chapter0 "
    /// Step 1b) Using this custom control in a XAML file that exists in a different project.
    /// Add this XmlNamespace attribute to the root element of the markup file where it is
    /// to be used:
    ///     xmlns:MyNamespace="clr-namespace:Chapter0 ;assembly=Chapter0 "
    /// You will also need to add a project reference from the project where the XAML file lives
    /// to this project and Rebuild to avoid compilation errors:
    ///     Right click on the target project in the Solution Explorer and
    ///     "Add Reference"->"Projects"->[Browse to and select this project]
    /// Step 2)
    /// Go ahead and use your control in the XAML file.
    ///     <MyNamespace:SwitchCustomControl/>
    /// </summary>
    public class SwitchCustomControl : Control
    {
        static SwitchCustomControl()
        {
            DefaultStyleKeyProperty.OverrideMetadata(typeof (SwitchCustomControl),
                new FrameworkPropertyMetadata(typeof (SwitchCustomControl)));
        }
    }
}

```



We can accomplish the same in Blend by adding a class and adding the code to it.

5. For each control, we should specify the rules of behavior of the control. We had a look at templates in *Chapter 4, Styles and Templates*. Here, we want to change the appearance of `ToggleButton` on checking and unchecking. We can do this by calling `GetTemplateChild` to apply this change. This change can be used to show any user-defined template. This is shown in the following code:

```
public class SwitchCustomControl : Control
{
    static ToggleButton SwitchButton;

    public override void OnApplyTemplate()
    {
        base.OnApplyTemplate();

        if (Template == null) return;

        SwitchButton = GetTemplateChild("SwitchTemplate") as ToggleButton;

        if (SwitchButton == null) return;

        // Detach event handlers on Checked and Unchecked event
        SwitchButton.Checked -= SwitchClick_Checked;
        SwitchButton.Unchecked -= SwitchButton_Unchecked;

        // Attach event handlers on Checked and Unchecked event
        SwitchButton.Checked += SwitchClick_Checked;
        SwitchButton.Unchecked += SwitchButton_Unchecked;
    }
    static void SwitchButton_Unchecked(object sender, RoutedEventArgs e)
    {
        SwitchButton.Background = new SolidColorBrush(Colors.Red);
        SwitchButton.Content = "Off";
    }
    void SwitchClick_Checked(object sender, RoutedEventArgs e)
    {
        SwitchButton.Background = new
        SolidColorBrush(Colors.Green);
        SwitchButton.Content = "On";
    }
}
```

6. We have overridden the `OnApplyTemplate` method of the `FrameworkElement` class. This is the method that is called before the element is displayed. In this method, we have applied `SwitchTemplate` and also added event handlers for checked and unchecked events. In these event handlers, we have set the background and text of `ToggleButton`.
7. We will add a path so that it looks like a push button. We discussed creating `Path` and `ControlTemplate` in the previous chapters. The XAML code will look somewhat similar to the following code:

```
<ToggleButton x:Name="SwitchTemplate" Background="Green"
Content="On" IsChecked="True" RenderTransformOrigin="0.5,0.5">
  <ToggleButton.Template>
    <ControlTemplate>
      <Viewbox Stretch="Uniform">
        <Canvas Height="75" Width="100">
          <Rectangle Canvas.Left="0" Width="40" Height="40"
Fill="{TemplateBinding Background}"
Stroke="{TemplateBinding BorderBrush}"
StrokeThickness="0.5"
RadiusX="10" RadiusY="10"/>
          <TextBlock Height="15" Width="30" Canvas.Left="10"
Canvas.Top="10" Text="{Binding Path=Content,
RelativeSource={RelativeSource TemplatedParent}}">
            </TextBlock>
          </Canvas>
        </Viewbox>
      </ControlTemplate>
    </ToggleButton.Template>
  </ToggleButton>
```

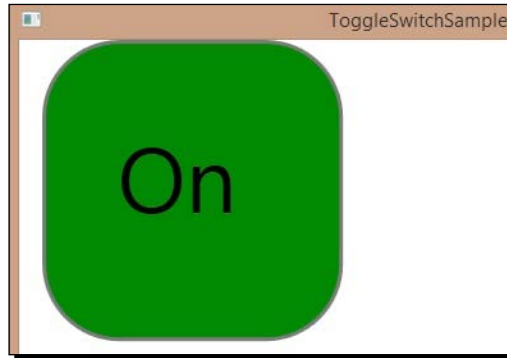
8. We created a control with the look and feel that we want. We will now use this XAML as the template for our custom control. `ControlTemplate` will make sure that the control looks the way we want it to look. The following code encapsulates this discussion:

```
<local:SwitchCustomControl x:Name="myControl">
  <local:SwitchCustomControl.Template >
    <ControlTemplate>
      <ToggleButton x:Name="SwitchTemplate">

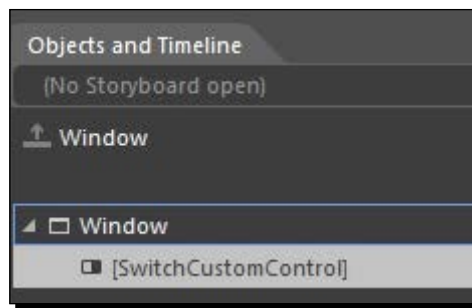
                                </ToggleButton>

      </ControlTemplate>
    </local:SwitchCustomControl.Template>
  </local:SwitchCustomControl>
```

9. Right-click on this custom control in the **Projects** panel and select startup:



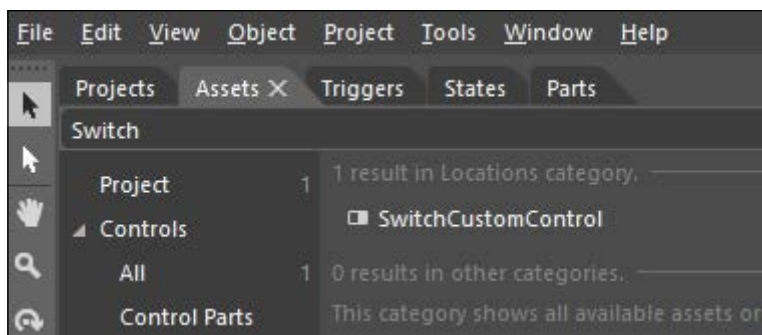
10. If we check the **Objects and Timeline** panel, we will see that **SwitchCustomControl** is added to ViewBox, as shown in the following screenshot:



11. When we run the application and click on the button, we see the background and content changing, as shown in the following screenshot:



- 12.** Also, now if we go to the **Assets** panel and search for **SwitchCustomControl**, we will be able to see it and simply use it by dragging and dropping it onto the art board. This is shown in the following screenshot:



What just happened?

We extended **ToggleButton** to create our custom control and also added behavior that will be visible when we click **ToggleButton**. We modified **ToggleButton** to look different and explicitly display the checked and unchecked states as **On** and **Off** respectively.

Pop quiz

Q1. What is a user control?

1. UserControl is a built-in Panel Control.
2. UserControl is a Shape Control.
3. UserControl is the same as Custom control.
4. UserControl wraps existing controls into a single reusable group.

Summary

In this chapter, we took a look at user controls and custom controls. In the next chapter, we will take a look at creating Windows Phone applications.

10

Creating Windows Phone Apps

One of the more popular forms of coding in recent times is developing apps (applications) that run on mobile devices, such as phones and tablets. Everyone using a smart phone is probably using one or more popular apps on a daily basis. In this chapter, we will create apps that will run on Windows Phone.

In this chapter, we will create a Windows Phone application from start to finish and submit it to the Windows Phone Store. This chapter will cover the following topics:

- ◆ Installing Windows Phone SDK
- ◆ Windows Phone introduction
- ◆ Creating and running a Windows Phone app
- ◆ Understanding Windows Phone Emulator
- ◆ Testing the application for store submission
- ◆ Submitting the app to the store

Installing Windows Phone SDK

To develop Windows Phone 8 applications, we need to install the Windows Phone 8 SDK. The following are the system requirements to run Windows Phone 8 SDK:

Operating System	Windows 8 64-bit (x64) client versions
Hardware	8 GB of free disk space 4 GB of RAM 64-bit (x64) motherboard
Windows Phone Emulator	Windows 8 Pro or higher (for Hyper-V) and Second Level Address Translation (SLAT)

The SDK is freely available at <http://dev.windowsphone.com/en-us/downloadsdk>. Download SDK 8.0 because, using that, we can develop an application for both Windows Phone 7.5 and Windows Phone 8 devices. If you already have a paid version of Visual Studio 2012, then the SDK will download and install the tools required for Windows Phone development. If you do not have Visual Studio 2012 installed, then the SDK will install Visual Studio 2012 Express for Window Phone.

An introduction to Windows Phone

The Windows Phone platform is different from the Windows platform. The first and foremost thing to keep in mind is that the Windows Phone platform has a smaller screen size, limited processing speed, and limited memory as compared to a computer, so our applications need to be designed to work efficiently within these limitations. The following is a list of capabilities and features that are available in Windows Phone applications:

- ◆ **Tiles:** A tile is a representation of an app on the screen. When you tap on a tile, the corresponding app is launched. The tile can have static content, such as an image or a graphic, or could have regularly changing content. This changing content could also be real-time information from the app.
- ◆ **Toast notifications:** Toast notifications allow the app to send messages to the user that are shown as pop-up notifications at the top of the screen. These toast notifications happen when the user is not in the app or is in another app.
- ◆ **Lock screen:** This screen is displayed when the device is locked. The app can still show the tile (when pinned), but it has to be a text-only version.
- ◆ **Location:** The location feature available on the phone allows the developer to query the current location of the phone.

- ◆ **Maps and Navigation:** The map allows the user to see the maps of locations and also see navigation directions to locations.
- ◆ **Speech:** Using the speech feature, we could command Windows Phone to perform various tasks, such as phone calls, texts, web search, and so on. More information on the speech feature can be found at <http://www.windowsphone.com/en-us/how-to/wp8/apps/use-speech-on-my-phone>.
- ◆ **Wallet:** The Windows Phone wallet can digitally save user credit cards, memberships, coupons, and so on, which could be used for purchases. More information on the wallet feature can be found at <http://www.windowsphone.com/en-US/How-to/wp8/apps/wallet>.
- ◆ **Camera and Photos:** The camera allows the user to capture photos and videos and can be accessed using the Photos and Videos application.
- ◆ **Permanent back button:** Windows Phone has a permanent back button that takes the user back to the immediate previous state. For example, if you are in an application, the back button might take you to the previous screen if you are on a second or further screen or out of the application if the user is on the first screen. This can be a nice addition to bring up a "pause menu" in games.
- ◆ **Accelerometer:** The accelerometer determines the direction of movement of the device. The accelerometer's sensor detects the force of gravity and other forces applied to the device that result in the movement of the device. This information is available via the Motion API ([https://msdn.microsoft.com/en-us/library/windows/apps/microsoft.devices.sensors.motion\(v=vs.105\).aspx](https://msdn.microsoft.com/en-us/library/windows/apps/microsoft.devices.sensors.motion(v=vs.105).aspx)). The API removes the gravity component from the device's acceleration, so the user is able to determine the current acceleration vector of the device. More information on getting data from the compass can be found at [https://msdn.microsoft.com/en-us/library/windows/apps/ff431810\(v=vs.105\).aspx](https://msdn.microsoft.com/en-us/library/windows/apps/ff431810(v=vs.105).aspx).
- ◆ **Compass:** The compass determines the angle of rotation of the device as compared to the earth's magnetic north pole. This sensor might not be available on all Windows Phone devices, so we should check for the sensor's availability before trying to get value from it. More information on getting data from the compass can be found at [https://msdn.microsoft.com/en-us/library/windows/apps/hh202974\(v=vs.105\).aspx](https://msdn.microsoft.com/en-us/library/windows/apps/hh202974(v=vs.105).aspx).
- ◆ **Gyroscope:** The gyroscope determines the velocity of the device across each axis. The values from the gyroscope can be used to determine the orientation of the device in space. The gyroscope, however, does not determine the rotational angle of the device. This sensor might not be available on all Windows Phone devices, so we should check for the sensor's availability before trying to get value from it. More information on getting data from the gyroscope can be found at [https://msdn.microsoft.com/en-us/library/windows/apps/hh202943\(v=vs.105\).aspx](https://msdn.microsoft.com/en-us/library/windows/apps/hh202943(v=vs.105).aspx).

Guidelines for Windows Phone applications

Microsoft has a complete set of design guidelines to design and develop Windows Phone applications. These provide guidelines from designing to conceptualizing the application. These design guidelines can be found at [https://msdn.microsoft.com/en-us/library/windows/apps/hh202915\(v=vs.105\).aspx](https://msdn.microsoft.com/en-us/library/windows/apps/hh202915(v=vs.105).aspx).

Understanding Windows Phone Emulator

The SDK also installs Windows Phone Emulator, which is a virtual machine running in a Hyper-V environment. This virtual machine runs the Windows Phone operating system in a phone-like window to run and test applications. The emulator provides us with a virtualized environment to run, debug, and test our applications. The important thing to note about the emulator is that it runs the applications with performance comparable to a real phone (mostly lower-end models). Microsoft, however, recommends the testing of applications on real devices before we submit those applications to the store. It's because emulators are imperfect (it doesn't have all the features available in a phone), and it is not uncommon to see differences between emulators (virtual machines) and real devices.

We can connect to the network from the emulator. The emulator behaves just like a physical device that is on the network (including the IP address). This networking feature is available by default. We can test almost all the features in the emulator apart from the following features:

- ◆ Compass
- ◆ Gyroscope
- ◆ Vibration controller
- ◆ Brightness

Time for action – Windows Phone Emulator

Now, let's see a few major capabilities of the emulator. We have three buttons available at the bottom of the emulator just as we would have on a phone. We do not have the side buttons of the phone available, but we can access them with keyboard function keys. The mapping is available at [http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff754352\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff754352(v=vs.105).aspx). We can toggle between the onscreen keyboard and the actual hardware keyboard by pressing *Pause/Break*.

The first button that we see is the back button with a backward-facing arrow that always performs the back action. It does not matter whether we are inside or outside an application. To check the major capabilities of the emulator, perform the following steps:

1. Let's click on the second button, which is the Windows button on the emulator.



That will take us to the home page of the emulator. Here, we will see the live tiles as we would on Windows Phone. We have the calendar, pictures, music, games, and so on. If we are developing applications using these components, we can test them within the emulator as well. At the start page of Windows Phone where our pinned applications are visible.

2. Now, if we swipe right by pressing the mouse's left button and dragging it to the left-hand side, then we will see the next screen has all the apps that are installed on the phone.
3. When we scroll down, we will find settings, which is the place where we can change the settings of this emulator. We can change the theme, assign ringtones, and turn the location on and off. Basically, we can make the most of the settings that we can on an actual device.
4. The third button that we see is the search button, which turns us to the Bing search page, which is more like a pivot application. If you swipe right by pressing the mouse's left button and dragging it to the left-hand side you can see the next screen.
5. Along with the emulator, we also see a fly-out menu. From this fly-out menu, we can perform the following actions on the emulator, as shown in the screenshot that follows these points:
 - ❑ Close
 - ❑ Minimize
 - ❑ Rotate counterclockwise and clockwise
 - ❑ Change size and zoom



What just happened?

We had a look at Windows Phone Emulator and the various features available in the emulator that allow us to run and test applications.



It is a good idea to keep the emulator running while working on the app; this will save you time by not restarting the emulator again and again, and Blend will automatically connect to the running instance of the emulator.

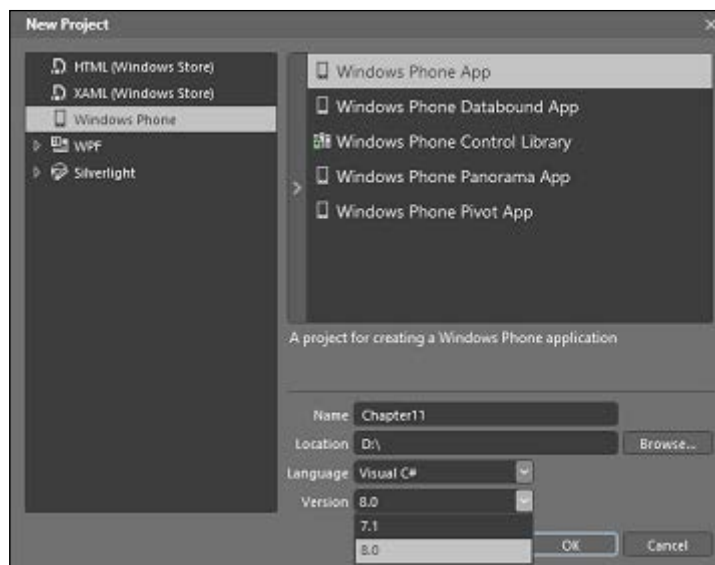
Creating a Windows Phone application

We will now create a Windows Phone application from the available templates. We will create a fun SoundBoard application. We will start off by adding one sound file and one button that plays the sound on being tapped.

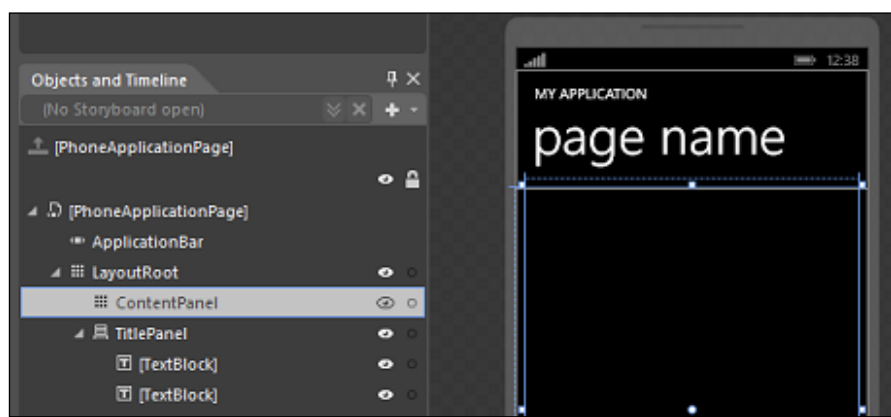
Time for action – creating a Windows Phone application

Perform the following steps to create a Windows Phone application:

- 1.** Start Blend and select **New Project** and **Windows Phone** in the project type in the left-hand side panel. Once we do that, we will see multiple types of available templates to create the **Windows Phone App**.
- 2.** Select **Windows Phone App**. We can choose between OS versions here, which are **7.1** or **8.0**. We will go ahead with **8.0** this time as that is the highest version supported by Visual Studio 2012. This is shown in the following screenshot:



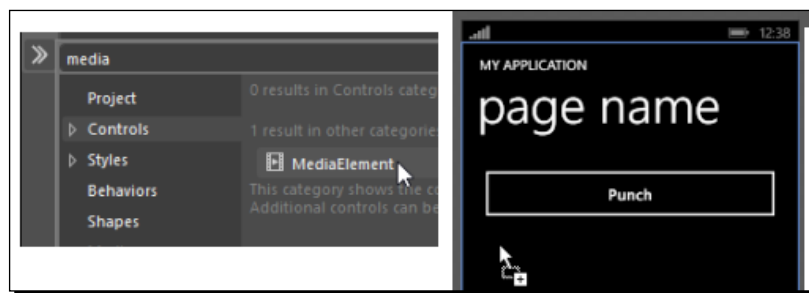
3. Once we click on **OK**, a new project is created, and, in the center, we see a phone-shaped art board that helps us imagine how the application will look on an actual device. Also, in the **Assets** panel, we see the controls available for the Windows Phone application. You can also notice that the screens for these apps are known as pages as evident from the name `MainPage.xaml`, which we can see in the **Projects** panel.
4. We can also see in the **Objects and Timeline** panel that we have **TitlePanel** (`StackPanel`) and **ContentPanel** (`grid`) added by default. In **TitlePanel**, we have two `TextBlock` objects, which show the application name and the page name. **ContentPanel** is where we put the main content for the page. This is depicted in the following screenshot:



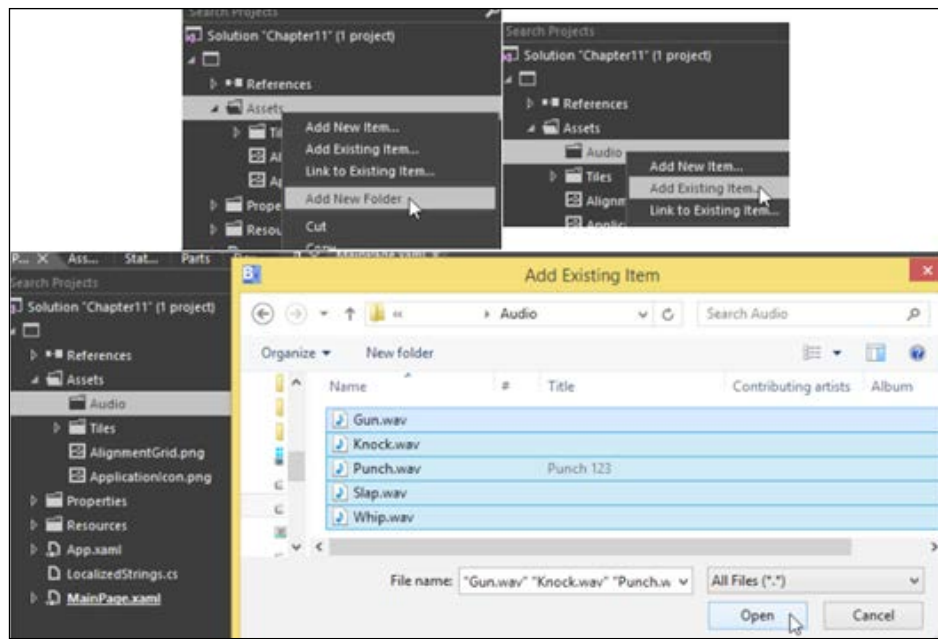
5. Add **Button** in the **ContentPanel** grid, and then change the content of the button to **Punch** as we will make this button play the punch sound on being clicked. The following screenshot illustrates this:



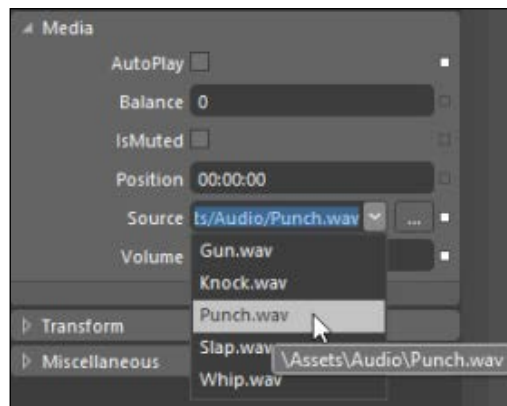
6. From the **Assets** panel, add **MediaElement** to the **ContentPanel** grid. This discussion is encapsulated in the following screenshot:



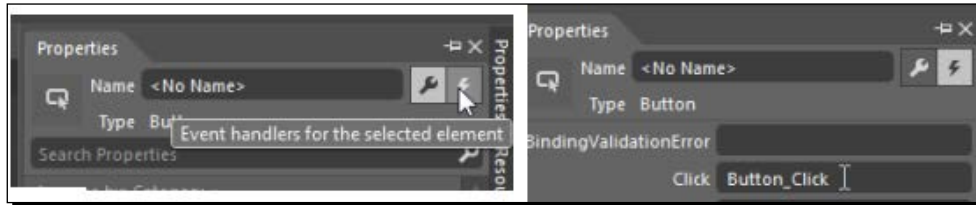
7. Create new folder in the **Assets** folder and rename it to **Audio**. Then, include the audio files (.wav) available along with the code of this chapter. The following screenshot shows this:



8. Select **MediaElement** from the **Objects and Timeline** panel, and then move to the **Properties** panel. Add name **MyMediaElement** to **MediaElement**, uncheck **AutoPlay** (we will play the sound on the button being clicked), select **Source** as **Punch.wav**, and change **Volume** to 1. The following screenshot illustrates this point aptly:



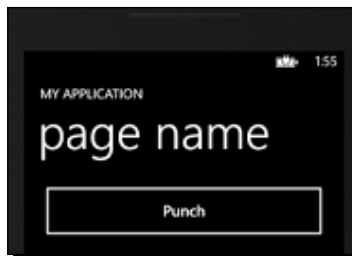
9. Also, move to event handlers for the button and double-click in front of the click event to generate an event handler:



10. In this event handler, we will add the code to play the audio file. Add the following code in the `Button_Click` handler:

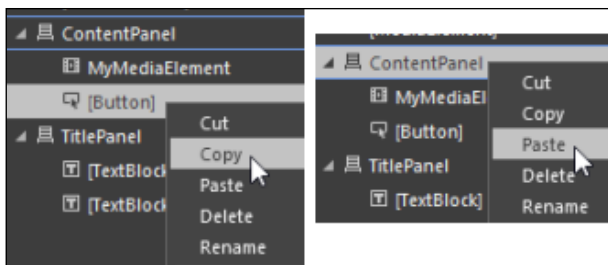
```
private void Button_Click(object sender, System.Windows.RoutedEventArgs e)
{
    this.MyMediaElement.Play();
}
```

11. Once we press `F5`, we will see that the application is running in a phone-like container. This is called Windows Phone Emulator. It is a virtual machine running a full Windows Phone 8 operating system. When we click on the **Punch** button, it will simulate the touch action on the emulator, and we will hear the punch audio playing. This is shown in the following screenshot:



12. In the **Objects and Timeline** panel, right-click on **ContentPanel** and select **Change Layout Type | StackPanel**. This will help us add multiple buttons easily.

- 13.** Right-click on **Button** in **ContentPanel**, select **Copy**, right-click on **ContentPanel**, and then select **Paste**. Repeat this four more times so that we have five buttons in total in **ContentPanel**. This is aptly illustrated in the following screenshot:



- 14.** Rename the buttons' text to Gun, Knock, Slap, and Whip, respectively. Add an event handler for the click event of each of the buttons as we did for the **Punch** button.
- 15.** Add the following code to `MainPage.xaml.cs`. This code will accept the URI of the audio file, load the audio file into `MyMediaElement`, and, once the audio file is loaded, `MyMediaElement` will play the file. This is encapsulated in the following code:

```
public MainPage()
{
    InitializeComponent();
    MyMediaElement.MediaOpened += MyMediaElementMediaOpened;
}

// Set the audio file as the source of MediaElement
// Add Event Handler for MediaOpened event
// This event is fired when file is opened after validation
private void SetAndPlay(string soundFileUrl)
{
    MyMediaElement.Source = new Uri(soundFileUrl,
        UriKind.Relative);
}

// Play the audio once it validated and opened
private void MyMediaElementMediaOpened(object sender,
    RoutedEventArgs e)
{
    MyMediaElement.Play();
}
```

16. Now on each the button clicks we will pass the address of the audio files respectively:

```
// Call method to play Gun sound
private void ButtonClick(object sender, RoutedEventArgs e)
{
    SetAndPlay("/Assets/Audio/Gun.wav");
}

// Call method to play Knock sound
private void ButtonClick1(object sender, RoutedEventArgs e)
{
    SetAndPlay("/Assets/Audio/Knock.wav");
}

// Call method to play Punch sound
private void ButtonClick2(object sender, RoutedEventArgs e)
{
    SetAndPlay("/Assets/Audio/Punch.wav");
}

// Call method to play Slap sound
private void ButtonClick3(object sender, RoutedEventArgs e)
{
    SetAndPlay("/Assets/Audio/Slap.wav");
}

// Call method to play Whip sound
private void ButtonClick4(object sender, RoutedEventArgs e)
{
    SetAndPlay("/Assets/Audio/Whip.wav");
}
```

- 17.** Now, when we run the application, we will hear different audios depending on the button we clicked. Take a look at the following screenshot that depicts this:

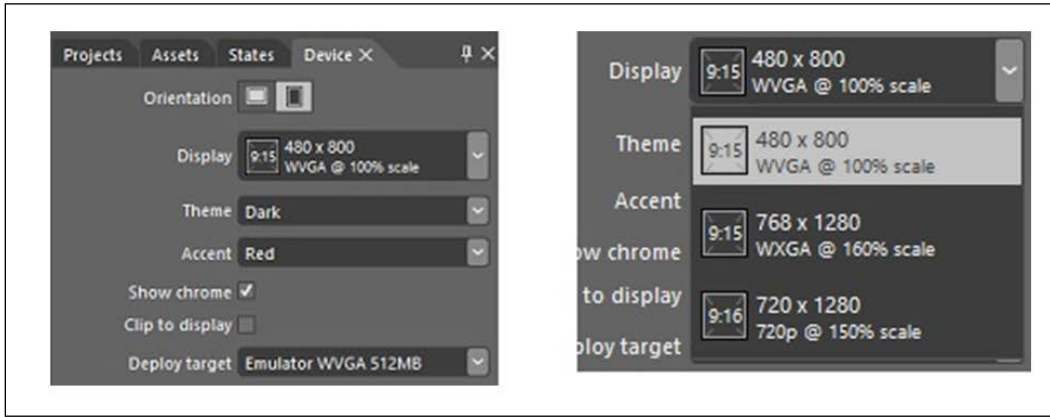


What just happened?

We created our first Windows Phone application and ran it on the emulator. You will notice that the emulator looks a lot like an actual phone. We will have a look at the emulator in detail in the next section.

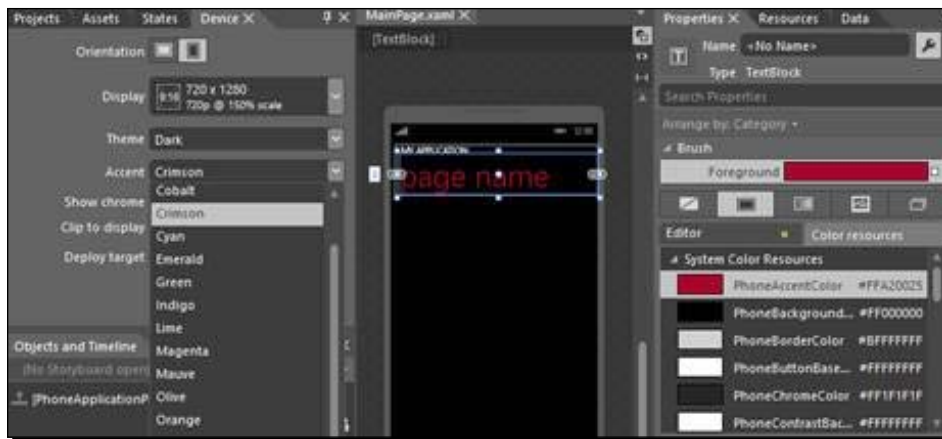
Exploring the Device panel

The **Device** panel is exclusive to Windows Phone applications. There are multiple settings available to preview the changes we are making to the application. When we move to the **Device** tab, which is available along with the **Projects** tab at the top, we see something similar to the following screenshots:

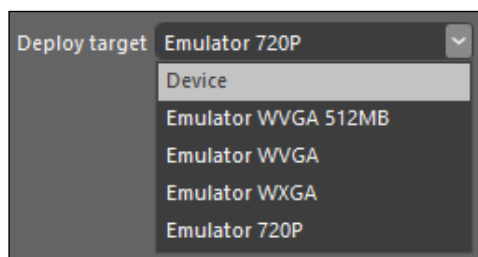


The options in the preceding screenshot are explained in the following points:

- ◆ **Orientation:** We can set the orientation to design the application in either the landscape orientation or the portrait orientation.
- ◆ **Display:** We can select the different resolutions and different aspect ratios and check how the application looks in that.
- ◆ **Theme:** We can choose from the dark and light themes as available on the phone and verify how the color scheme looks.
- ◆ **Accent:** We can choose the accent to be one of the various colors available. When the accent color is changed by the user of the application, the changed color is reflected at the places where we have used Windows Phone's built-in styles. As illustrated in the following screenshots, we have assigned the foreground color of **page name** to **PhoneAccentColor**. Now, when we change **Accent** color to **Crimson**, the foreground color of page name changes as well.



- ◆ **Show chrome:** This setting enables or disables the chrome UI of the phone.
- ◆ **Clip to display:** This resizes the data content to fit the app's viewing area.
- ◆ **Deploy target:** This is the only runtime setting available. This allows us to deploy our application to a real device or different versions of the emulator. The second option in the list, **Emulator WVGA 512 MB**, runs our application in a memory-constrained environment. We can use this option to verify our apps for lower-configuration devices. The WVGA (800 x 480), WXGA (1280 x 768), and 720p (1280 x 720) emulators allow us to test our applications on the various possible resolutions of Windows Phone devices. You should test your application on all the resolutions to make sure that the app looks good on all the phones. This point is illustrated in the following screenshot:



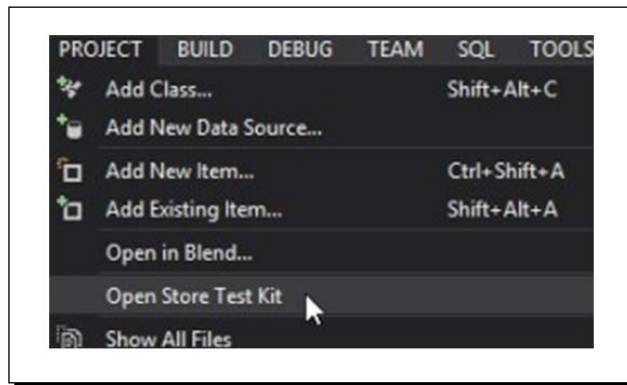
Testing the application before submitting to the store

Once we have created our application, we will need to test the app using Store Test Kit before submitting it to the store.

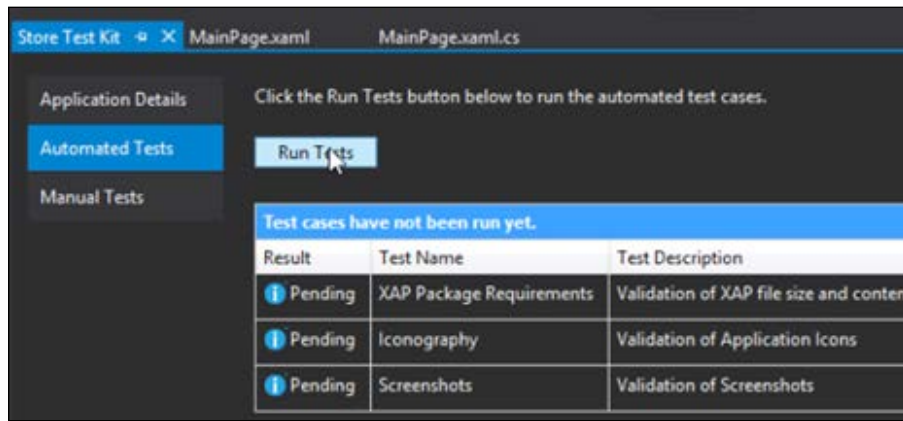
Time for action – testing our application

Perform the following steps to test your application:

1. Before we try to submit the app to the store, we need to do the compliance testing of the application. Store test kit can help us with that. Open the project in Visual Studio, and then, from the menu, select **Project | Open Store Test Kit**. The following screenshot shows this:



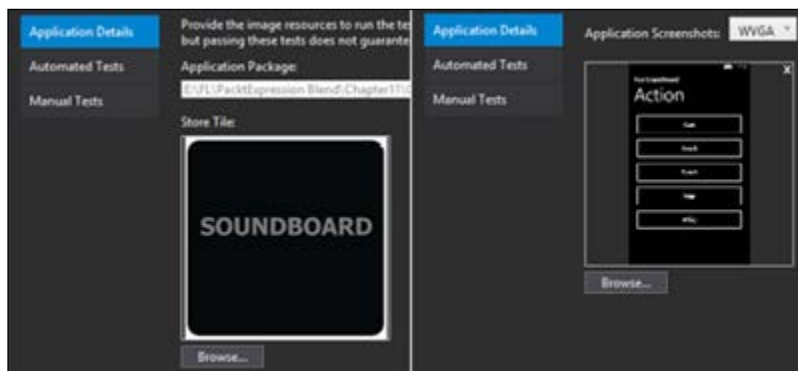
2. This will open **Store Test Kit**. You will find three options on the left-hand side—**Application Details**, **Automated Tests**, and **Manual Tests**. Select **Automated Tests** and click on **Run Tests**. This is illustrated in the following screenshot:



3. You will see an error stating that the tests will expect a release version of the XAP package.
4. To fix this error, in the toolbar, select the drop-down list next to the **Run** button and select **Release**. Then rebuild the project. This is shown in the following screenshot:

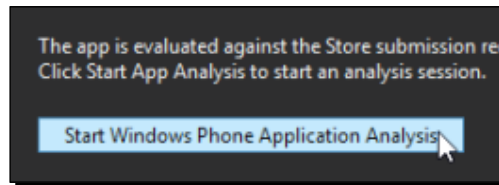
Passed: 1 Failed: 2	
Result	Test Name
✓ Passed	XAP Package Requirements
✗ Failed	Iconography
✗ Failed	Screenshots

5. Now, click on **Run Tests** again, and this time, we will see that the tests run, but two of them fail because we have not provided the appropriate icon and screenshots for the application.
6. You could either provide your icon and screenshot for the application or use the ones provided with this chapter. We need to add the icon and screenshot in **Application Details**. We need to provide a 300 x 300 application tile. We also need to provide at least one screenshot for each of the resolutions, that is: 480 x 800 (WVGA), 768 x 1280 (WXGA), and 720 x 1280 (720P). Take a look at the following screenshot that shows this:

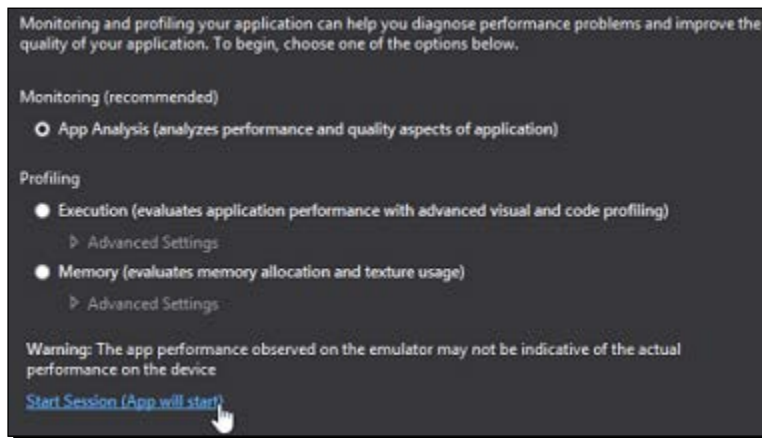


7. Now, when we rerun **Automated Tests**, we will see that all the tests pass.

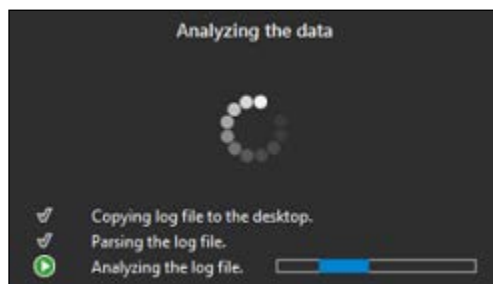
- Also, on the **Automated Tests** tab, we see the **Application Analysis** button. This execution will add a new `.sap` file, which is a blank performance log file that captures data about the performance of your app as it runs. The log file will then be pored over and reported on to let you know how well it runs in certain situations. Click on **Start Windows Phone Application Analysis**.



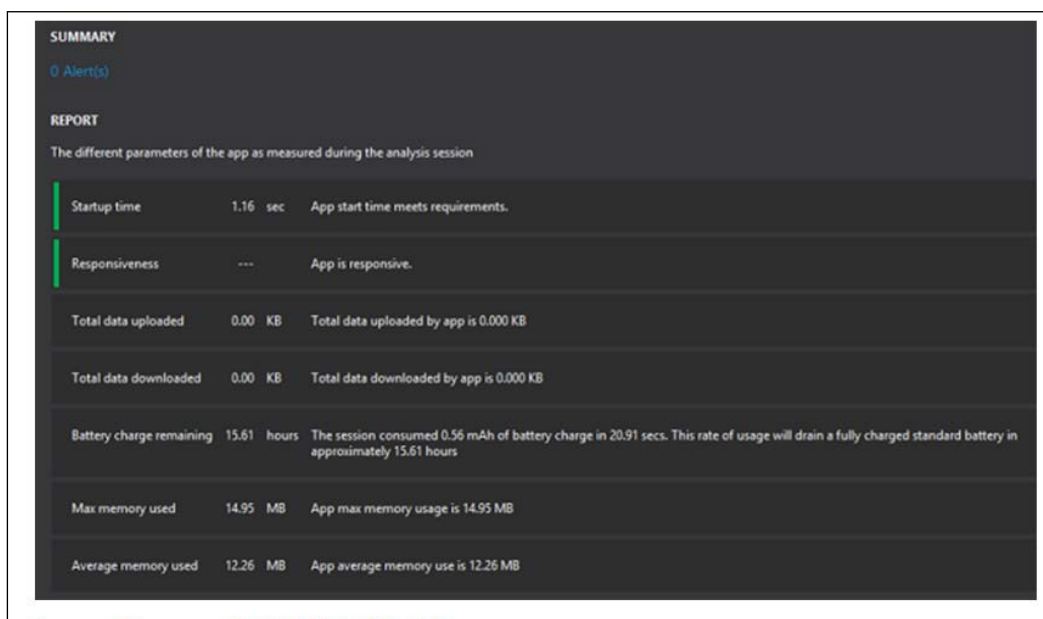
- You will see a new screen with multiple types of data that we could collect for reporting as illustrated in the following screenshot. You can find more details about this at [http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj215908\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj215908(v=vs.105).aspx).



- 10.** When we click on the **Start Session (App will start)** link, the application will load in the emulator, and we will see the monitoring message in Visual Studio. During this time, we can perform all the intended and expected scenarios in the application. This is depicted in the following screenshot:



- 11.** When we click on the **End Session** link, we see a report similar to the following screenshot. This report helps us in analyzing the readiness of the application for the store.



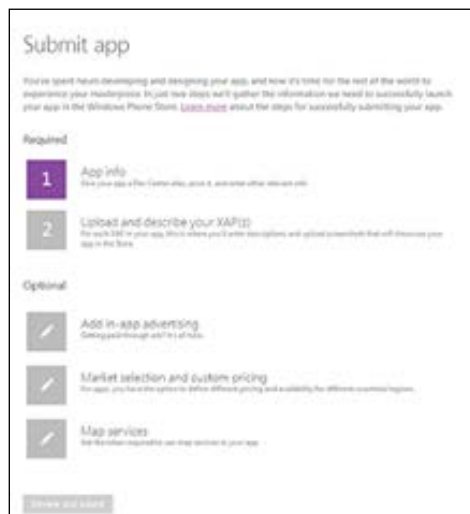
Submitting our application to the store

Once we have created and tested our application, the time comes to submit this application to the store to share it or to make money. This is a multiday process, and it might take multiple attempts to be successful.

Time for action – submitting the application

Perform the following steps to submit the application:

1. The first thing that we need to do to submit the app is create a developer account at <http://developer.windowsphone.com>.
2. Once that is done, go to <https://dev.windowsphone.com/join> to buy the developer subscription. There is a onetime fee of \$19 to register as an individual app developer.
3. Follow the steps mentioned at <https://dev.windowsphone.com/register> to make the payment and get a developer subscription.
4. To test your application on a Windows Phone device, your phone needs to be registered at the developer centre and unlocked using the developer unlock tool.
5. Once you have a valid subscription, submit the application, and then you will see a screen similar to the one shown here. Just follow the steps to submit the application. Some of the steps may be optional depending on the type of application and whether it's a free app or not.



6. Before you submit the app to the store, go through the instructions mentioned at [http://msdn.microsoft.com/en-US/library/windowsphone/develop/hh184843\(v=vs.105\).aspx](http://msdn.microsoft.com/en-US/library/windowsphone/develop/hh184843(v=vs.105).aspx). This will help you in reducing the app rejections.
7. You can also find more information about the submission process at [http://msdn.microsoft.com/en-US/library/windowsphone/help/jj206729\(v=vs.105\).aspx](http://msdn.microsoft.com/en-US/library/windowsphone/help/jj206729(v=vs.105).aspx).

What just happened?

We submitted our application to the store by following the guidelines laid out by Microsoft to submit applications.

Pop quiz

Q1. What are the various display modes supported by Windows Phone?

1. WXGA.
2. WVGA.
3. 720p.
4. All.

Q2. Can we have live tiles for the application in Window Phone?

1. No.
2. Yes.

Q3. Can we take screenshots from Windows Phone Emulator?

1. Yes.
2. No.

Summary

In this chapter, we created different types of Windows Phone applications, used an emulator, and submitted the app to the store.

In the next chapter, we will take a look at creating Windows Store applications.

11

Creating Windows 8 Store Apps

Apps are at the core of the Windows 8 experience. Windows 8 is installed on millions of devices, and the apps we create can reach hundreds of markets worldwide. Windows 8 apps can work on different devices, such as desktops, laptops, tablets, all-in-ones, and so on. We will see how to develop and submit a Windows app.

In the previous chapter, we had a look at creating, running, and submitting Windows Phone apps. In this chapter, we will cover the following topics:

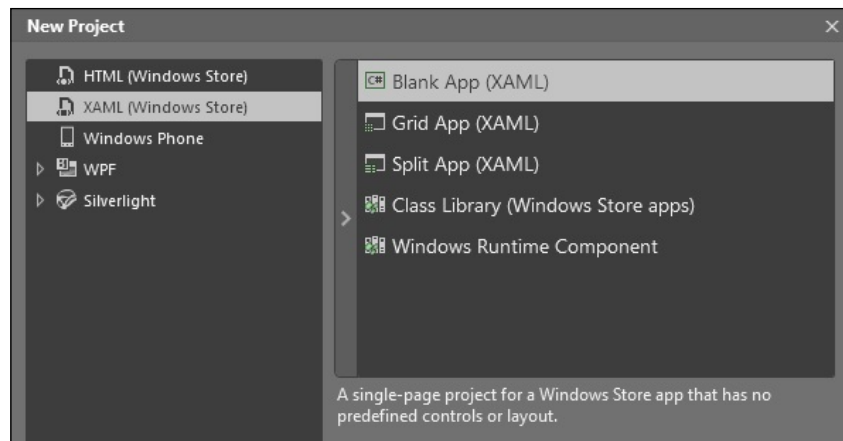
- ◆ Different templates available for Windows 8 Store apps
- ◆ Creating and running a Windows 8 Store app
- ◆ Submitting the Windows Store app

We can develop Windows 8 Store applications only on a Windows 8 (or higher) system. So, that being said, in order to run samples of these chapters and get hands-on, we will need a Windows 8 system.

Templates

We could either create an XAML or HTML Windows Store application. In this book, we will talk about XAML applications. When we create a new project in Blend for Visual Studio 2012 and select the XAML Windows Store application in the left-hand side panel, we will have the following five templates to select from:

- ◆ **Blank App (XAML):** This is a project with nothing in it.
- ◆ **Class Library (Windows Store apps):** A class library for Windows Store apps lets us create a managed class library that can be used and reused in Windows Store apps and Windows Runtime Components.
- ◆ **Grid App (XAML):** The Grid App template is a great way to start a Windows Store app that you can customize to enable users to browse through categories to find content in which they will want to fully immerse themselves. Examples include shopping apps, news apps, and photo or video apps.
- ◆ **Split App (XAML):** The Split App template is a great way to start creating a Windows Store app that you can customize to enable users to view a list of items and item details in a two-column view, where users might want to switch quickly between items and where the list might be updated dynamically. Examples include a newsreader, a sports scores app, or an e-mail app.
- ◆ **Windows Runtime Component:** We can use this template to create `dlls` (class libraries) that can be used by any Windows Store application regardless of the language in which the application is written in:



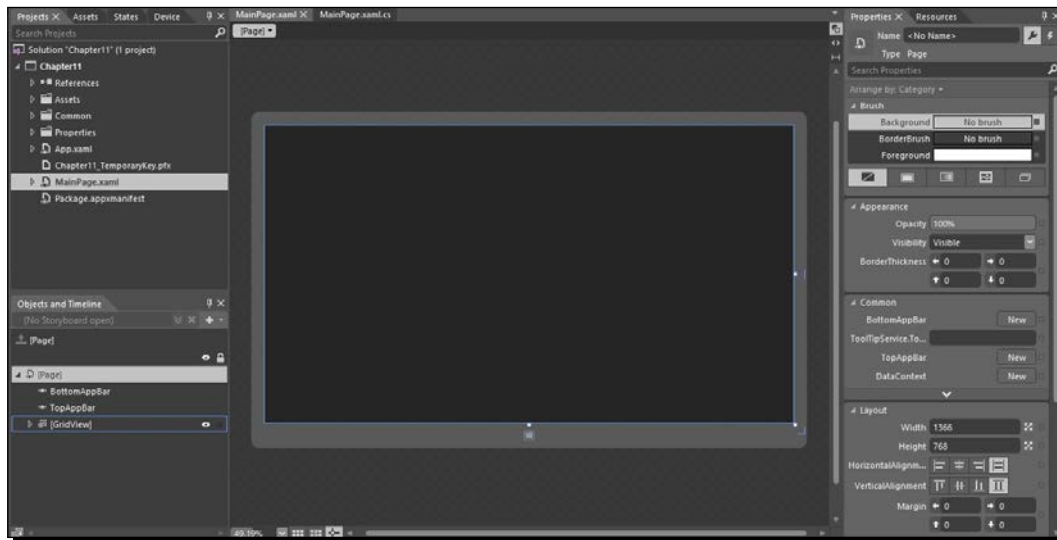
Creating Windows Store apps with XAML and C#

In this section, we will have a look at how we can create a Windows 8 Store application using XAML and C#. We will create an application similar to the Windows Phone application that we created in the previous chapter.

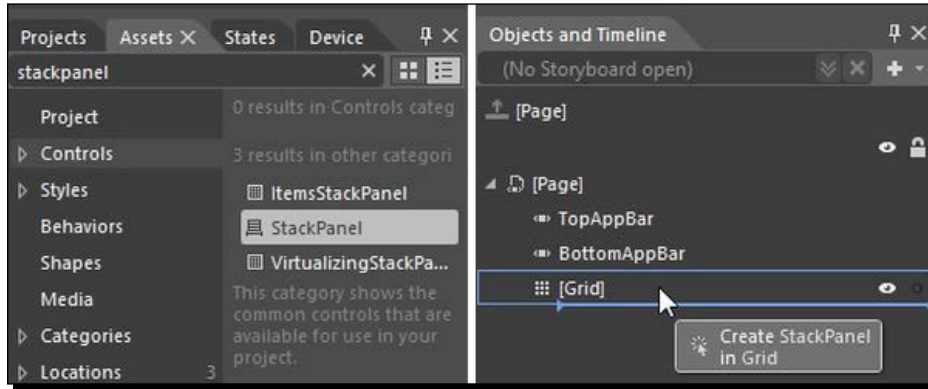
Time for action – creating a Windows 8 Store app

To create a Windows 8 Store app, perform the following steps:

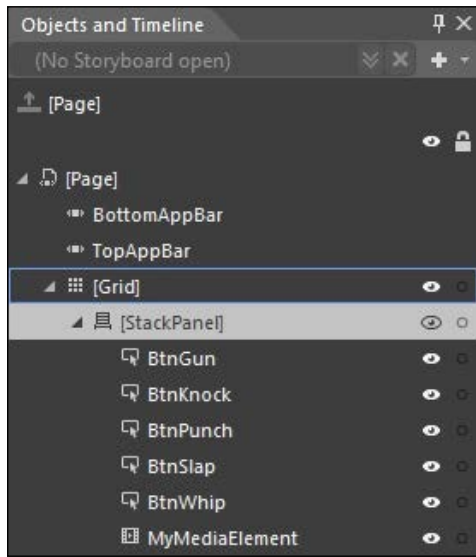
1. Create a new project, select **XAML (Windows Store)** in the left-hand side panel, and select **Blank App (XAML)** in the right-hand side panel.
2. Once we click on **OK**, we will see a screen similar to the following screenshot. Except for the center area, the rest of the screen looks quite similar to the screens that we have seen in the previous chapters of the book. This template provides the infrastructure to create XAML Windows 8 Store apps.



3. Add **StackPanel** in **Grid** by selecting **StackPanel** from the **Assets** panel and dropping it onto the grid. This is depicted in the following screenshot:

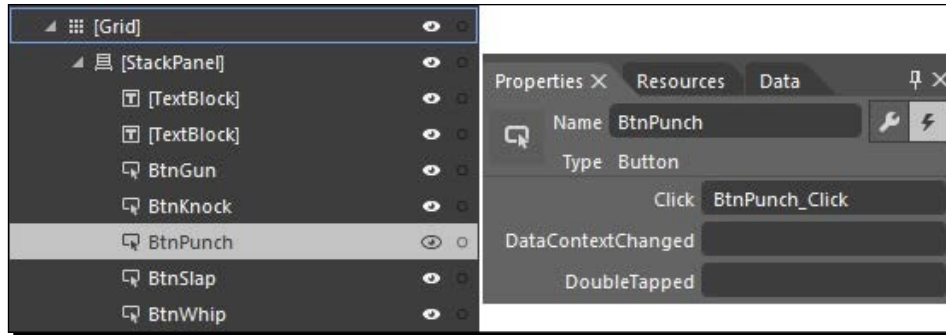


4. Add five buttons and one **MyMediaElement** in **StackPanel** and name them as shown in the following image:



5. Create a new folder in the **Assets** folder and rename it to **Audio**. Then, include the audio files (.wav) available along with the code of this chapter.

6. Select **BtnPunch**, move to the **Events** panel, and double-click in front of the **Click** event to add an event handler to the **Click** event. Do the same for the remaining buttons as well. This is shown in the following screenshot:



7. Add the following code to the code-behind file to handle click events on the buttons:

```
private void BtnGun_Click(object sender, RoutedEventArgs e)
{ SetAndPlay("/Assets/Audio/Gun.wav"); }

private void BtnKnock_Click(object sender, RoutedEventArgs e)
{ SetAndPlay("/Assets/Audio/Knock.wav"); }

private void BtnPunch_Click(object sender, RoutedEventArgs e)
{ SetAndPlay("/Assets/Audio/Punch.wav"); }

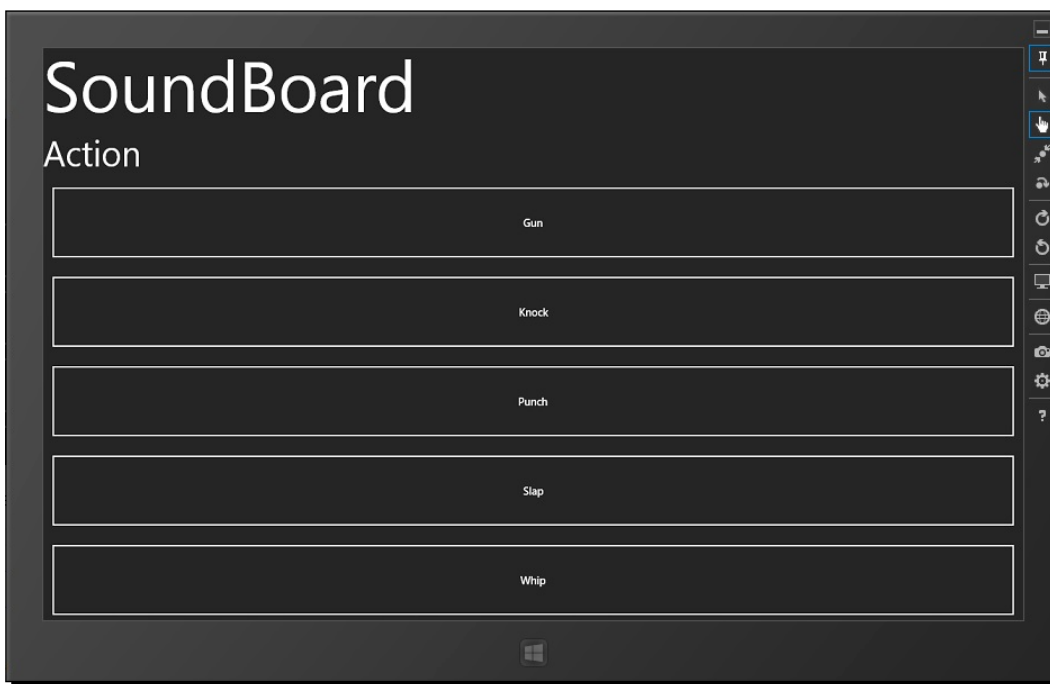
private void BtnSlap_Click(object sender, RoutedEventArgs e)
{ SetAndPlay("/Assets/Audio/Slap.wav"); }

private void BtnWhip_Click(object sender, RoutedEventArgs e)
{ SetAndPlay("/Assets/Audio/Whip.wav"); }

private void SetAndPlay(string soundFileUrl)
{ MyMediaElement.Source = new Uri(this.BaseUri, soundFileUrl); }

private void MyMediaElementMediaOpened(object sender,
RoutedEventArgs e)
{ MyMediaElement.Play(); }
```

8. Now, when we run the application, we see a fullscreen **SoundBoard** application, which is shown in the following screenshot:



Have a go hero

Because of the limited content that can be put into this book, we cannot cover all the types of templates that are available to create XAML Windows 8 Store applications. However, we recommend that you go ahead and try to create applications based on the various templates that are available.

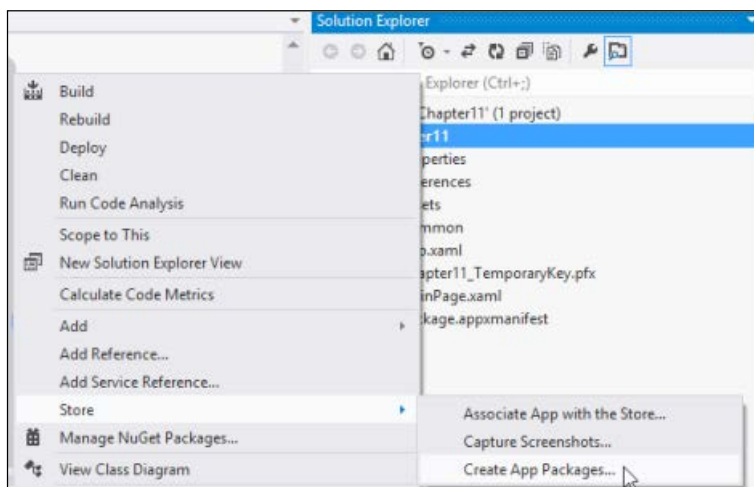
Submitting your app to Windows Store

Once we are done with our application, we need to submit the application to the store. The store is located at <https://dev.windows.com/en-us/dashboard>. Once you have signed up for a developer account, you will be able to submit the application. While submitting the application, we need to provide details about the application. There are two ways to submit the application.

Time for action – submitting the app to Windows Store

Before we submit the application to the store, we need to perform the following steps:

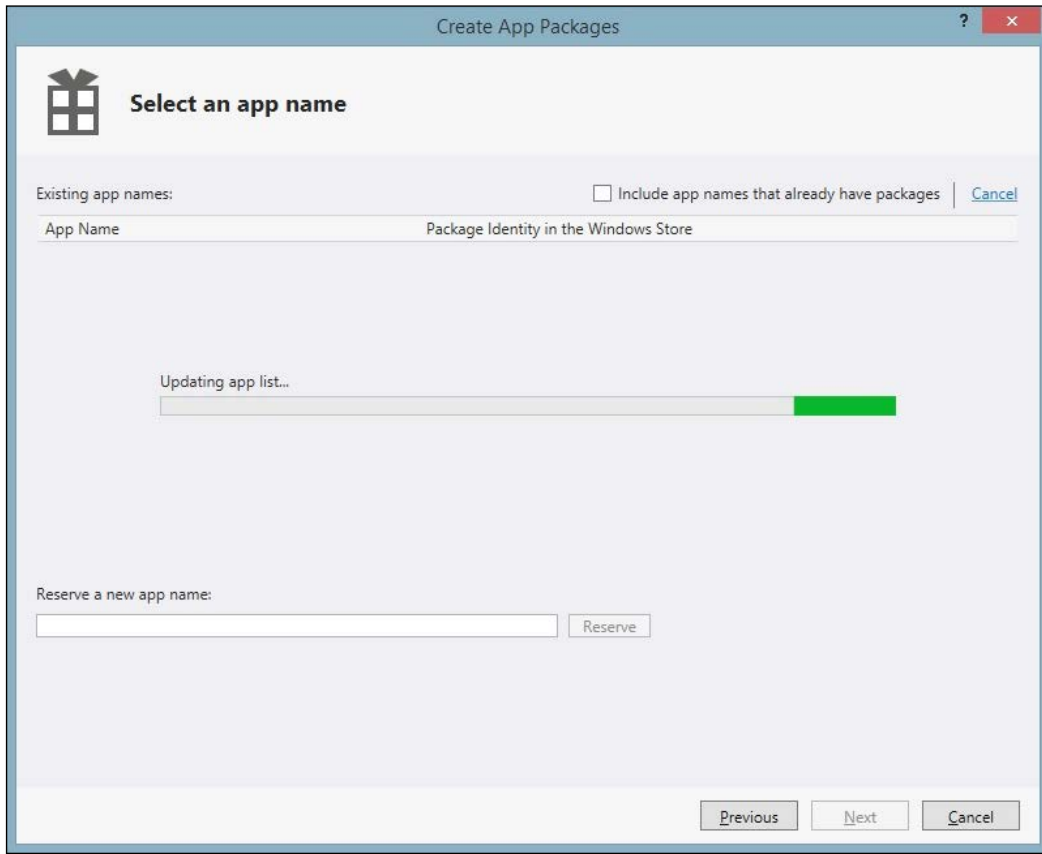
1. Capture the application's screenshots—we can do that by running the application on a simulator and capturing the screenshots.
2. Create the application package by right-clicking on the project name in Visual Studio and select **Create App Packages...**. The following screenshot depicts this step:



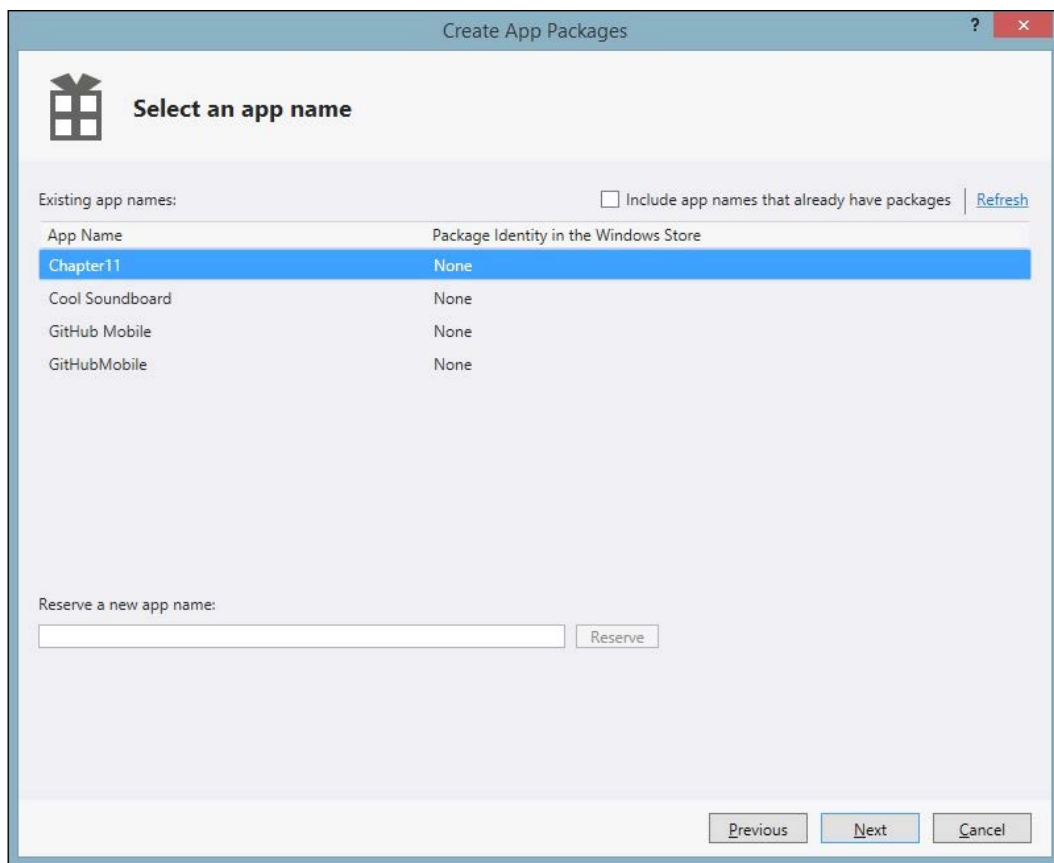
3. We will be asked whether we want to create packages to submit to Windows Store or not. Select **Yes** and click **Sign In**. Enter your Windows Store credentials and proceed. This step is shown in ample detail by the following screenshot:



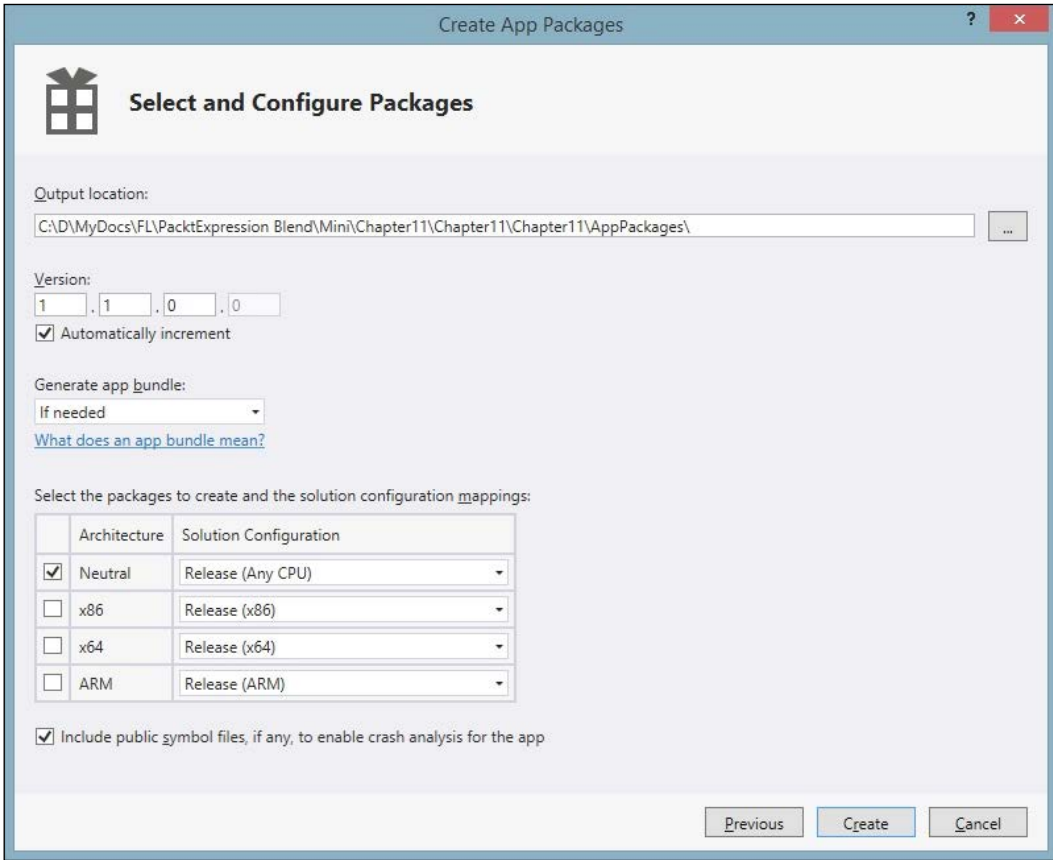
4. Go to Windows Store and reserve a name for the application if you haven't already done that. The following screenshot shows you how:



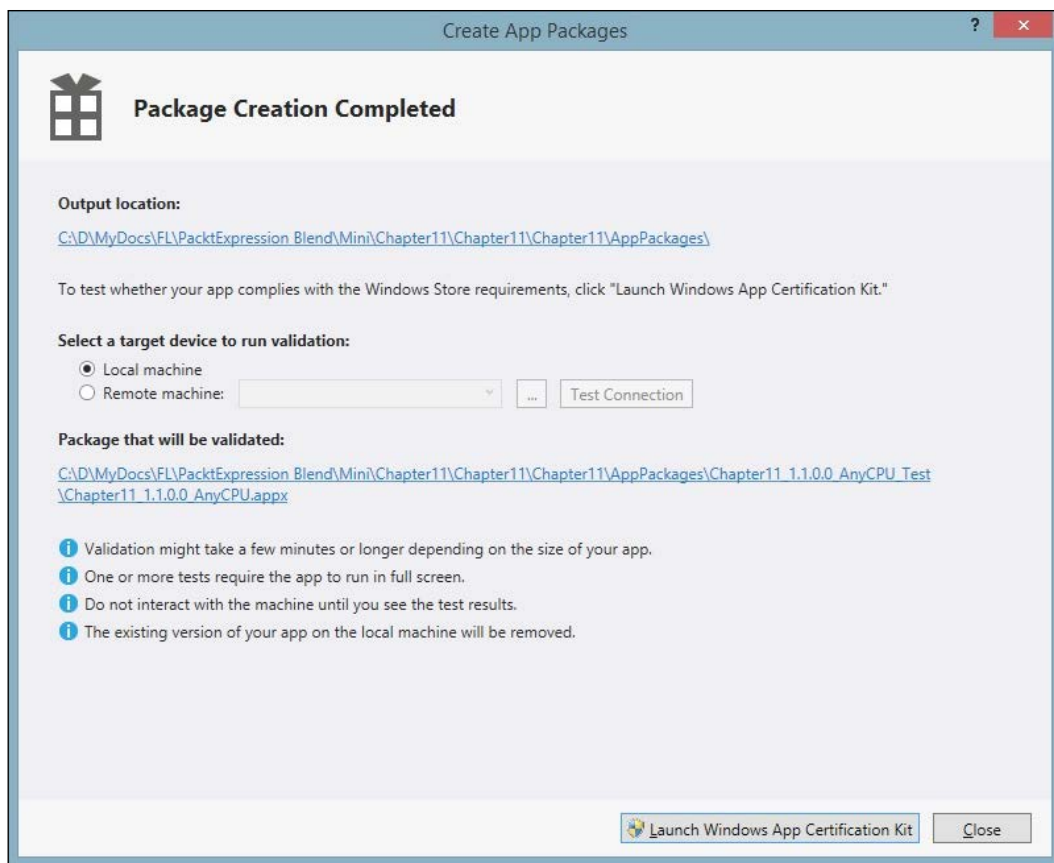
5. Once we have reserved a name for the app, it will autopopulate in the window. To move further, we need to select the app and click on **Next**, as shown in the following screenshot:



- Keep the default settings and click on **Create**, as depicted in the following screenshot:



7. Once the app packages are created, we will see the following window that allows us to verify the application for certification using **Launch Windows App Certification Kit**. This is shown in great detail in the following screenshot:



8. Make sure that the app certification kit passes all tests.
9. Before we upload the app packages we just created, we need to provide the information required for that.

What just happened?

We had a look at the steps to submit our app to Windows Store.

Stages of app submission







Once we have submitted the application, our application goes through various stages. Here's a quick look at what is happening behind the scenes during each of these stages:

1. **Preprocessing:** This is where the application will be checked to make sure that it has all of the appropriate details that are needed to publish your app. This includes checking the status of the developer account, and, if the app has a purchase price or any in-app offers, it is ensured that the Windows Store team has all of the paperwork in a file so that the developer can be paid.
2. **Security tests:** The application is checked for viruses and malware.
3. **Technical compliance:** Windows App Certification Kit is used to check that the app complies with the technical policies. These are exactly the same technical certification assessments that are included in the SDK and that can be run locally before the developer uploads their package.
4. **Content compliance:** The testers take a look at the app to check that the contents comply with the content policies. Since there are real people looking at the app, this process can take longer than the other steps.
5. **Release:** This stage goes by very quickly unless the developer has specified a publish date in the future.
6. **Signing and publishing:** In this final step, the packages you submitted with a trusted certificate must match the technical details of the developer account. This provides customers with the assurance that the app is certified by Windows Store and hasn't been tampered with. Then, app packages are published to the store, along with all of the other data that will be visible in the app listing page so that millions of Windows 8 users will be able to find, acquire, and enjoy the app.



We can check the current stage of the application from the dashboard at <https://appdev.microsoft.com/storeportals>.

Certification status
We are certifying Weather: release 1 for listing in the Windows Store. [Learn more](#)

 Complete	Pre-processing Usually done within 1 hour
 Complete	Security tests Usually done within 3 hours
 In progress	Technical compliance Usually done within 6 hours
 Pending	Content compliance Usually done within 5 days
 Pending	Release Waiting until the app passes certification
 Pending	Signing and publishing Usually done within 2 hours



Also, before you submit the application, do not forget to go through the application certification requirements at <http://msdn.microsoft.com/en-us/library/windows/apps/hh694083.aspx>.

Pop quiz

Q1. How can we distribute a Windows app?

1. We can distribute it through Windows 8 Store.
2. We can host it on a web server.
3. We can distribute it through CD/DVD.
4. We can put it on the network drive.

Q2. Can we take screenshots from a simulator?

1. Yes.
2. No.

Summary

In this chapter, we created different types of Windows 8 Store applications, used a simulator, and submitted the app to the store.

Pop Quiz Answers

Chapter 1, Getting Started with Blend

Pop quiz

Q1	2
----	---

Chapter 2, Layout Panels

Pop quiz

Q1	1
Q2	4

Chapter 3, Working with XAML

Pop quiz

Q1	1
Q2	1

Chapter 4, Styles and Templates

Pop quiz

Q1	1
----	---

Chapter 5, Behaviors and States in Blend

Pop quiz

Q1	1
Q2	2

Chapter 6, Understanding Animation and Storyboards

Pop quiz

Q1	4
Q2	2

Chapter 7, Understanding DataBinding

Pop quiz

Q1	3
Q2	3

Chapter 8, Vector Graphics

Pop quiz

Q1	4
Q2	3

Chapter 9, User Controls and Custom Controls

Pop quiz

Q1	4
----	---

Chapter 10, Creating Windows Phone Apps

Pop quiz

Q1	4
Q2	2
Q3	1

Chapter 11, Creating Windows 8 Store Apps

Pop quiz

Q1	1
Q2	1

Chapter 12, Prototyping Using SketchFlow

Pop quiz

Q1	1
Q2	1
Q3	1



Note that *Chapter 12, Prototyping Using SketchFlow*, is available online at https://www.packtpub.com/sites/default/files/downloads/38820T_Chapter12.pdf.

Index

A

animation

- frame-based animation 85
- recording symbol 96
- rotation animation 94
- service 86
- time-based animation 85

animation behaviors

- about 70
- ControlStoryboardAction 70-72
- FluidMoveBehavior 73
- FluidMoveSetTagBehavior 73
- storyboard, adding 70

animation workspace

- about 88-91
- switching 91

app

- submission, stages 190, 191
- submission, URL 191
- submitting, to Windows Store 184-189
- URL 190

application

- creating, in Blend 4
- executing 19
- integrating, into Visual Studio 20
- Silverlight application, creating 5, 6
- submitting, to store 176, 177
- testing 172-175

application skinning

- about 61
- resource dictionaries, creating 62, 63

Art board 6

Assets

- Behaviors 10
- Effects 10
- Styles 10
- UI components 10

Assets panel 7

attached property 108

B

behavior objects 69

BitmapScalingMode

- about 137
- URL 137

Blend

- about 1
- application, creating 4-6
- downloading 2, 3
- for Visual Studio 2012 1, 2
- installing 3

Blend IDE

- about 6
- Art board 6
- Assets panel 7
- Object and Timeline panel 7
- Open documents 6
- Projects panel 7
- tools panel 8, 9

brushes

- about 13
- gradient brush 15-17
- properties 13

- solid color brush 14, 15
- tile brush 18, 19

Brush tools

- Eyedropper 9
- Gradient 9
- Paint bucket 9

built-in behaviors

- adding 70
- animation behaviors 70
- conditional behaviors 70
- data behaviors 70
- motion behaviors 70
- types 70

C

C#

- code 40
- used, for creating Windows Store apps 181

canvas

- about 32
- using 32, 33

code-behind class

- about 44
- named element, using 44

Common controls 9

Common Language Runtime (CLR) 2

compound keyframes 92

conditional behaviors

- about 70-73
- CallMethodAction 73
- ChangePropertyAction 73
- ControlStoryboardAction 74
- GoToStateAction 74
- HyperlinkAction 74
- InvokeCommandAction 74
- LaunchUriOrFileAction 74
- PlaySoundAction 74
- RemoveElementAction 74
- SetDataStoreValueAction 74

custom control

- about 150
- creating 150-155
- versus user control 142, 143

D

data behaviors 70

DataBinding

- about 107-109
- background, with SelectedValue 122
- control to control 114
- properties to control 110
- to DataSource, as collection 117-121
- to ones own property 111-113
- to properties, of different control 114-116

DataBinding model 110

DataBinding, modes

- about 109
- Default 109
- OneTime 109
- OneWay 109
- OneWayToSource 109
- TwoWay 109

DataSource

- DataBinding to, as collection 117-120
- using 117

data state behaviors

- about 74
- CallMethodAction 74
- DataStateBehavior 74
- FuildMoveSetTagBehavior 74
- InvokeCommandAction 74
- SetDataStoreValueAction 74

default properties 45

dependency object

- URL 108

dependency properties 107, 108

Device panel

- about 170
- Accent option 170
- Clip to display option 171
- Deploy target option 171
- Display option 170
- Orientation option 170
- Show chrome option 171
- Theme option 170

documentation

- using 21, 22

DPI awareness

- Not DPI-aware applications 138
- Per-monitor-DPI-aware Applications 138
- System-DPI-aware applications 138
- URL 138

DreamSpark program

- URL 2

dynamic stage 76

E

easing functions

- about 101-103
- types 102
- using 101

elements

- adding in XAML, by hand-coding 45, 46

Extensible Application Markup Language. *See* XAML

F

frame-based animation 85

G

gradient brush

- about 15
- background color of grid, modifying 16, 17
- linear gradient brush 18
- radial gradient brush 18

graphics

- importing 130-132

grid layout

- about 24
- used, for creating Run window 25-31

H

help menu

- using 21

I

implicit keyframes 93

INotifyPropertyChanged

- URL 110

integrated development environment (IDE) 6

ItemControls

- URL 117

K

keyframe

- about 92
- compound keyframes 92
- editing 96
- implicit keyframes 93
- object-level keyframes 92
- simple keyframes 92
- using 93

KeySpline 104

L

layout containers

- about 35
- border 36
- popup 36
- ScrollViewer 36
- UniformGrid 36
- ViewBox 36

Layout panels 9

linear gradient brush 18

Line tool 132

M

merged dictionaries 67

motion behaviors

- about 70, 74
- MouseDownElementBehavior 74
- TranslateZoomRotateBehavior 75

MSDN subscription

- URL 2

N

namespace

- adding, in XAML 42, 43

naming elements

- about 43
- in code-behind class 44

non-attribute syntax

- about 46
- gradient, defining for grid 46

O

Object and Timeline panel 7

object-level keyframes 92

Open documents 6

P

panels

canvas 23

DockPanel 23

grid 23

StackPanel 23

WrapPanel 23

Paths

about 134

modifying 134-136

Path tools

Pen 9

Pencil 9

Pencil tool 134

Pen tool

about 132

used, for creating shape 132, 133

Projects panel 7

properties

expressing, as attributes 45

Properties panel 91

R

radial gradient brush 18

raster graphics 126

resource dictionaries

about 55

creating 62, 63

resources

application level 52

creating 53-55

document level 52

dynamic resources 52

element level 52

static resources 52

Rich Internet Applications (RIAs) 2

S

SelectedValue

used, for DataBinding background 122

Selection tools

Direct selection 8

Selection 8

shapes

about 129

adding 129

creating, Pen tool used 132, 133

URL 129

Shape tools 9

Silverlight 2

simple styles

about 56

changing 58

colors, changing 58

control templates, changing 59

simple styled control, creating 56

solid color brush

about 14

options 14

text color, modifying 15

StackPanel

about 33

using 34, 35

static stage 76

store

application, submitting 176, 177

storyboards

about 86, 87

adding 87-89

properties 97, 98

URL 86

XAML 98-100

styles

about 52

creating 51

defining 47

defining, in XAML 48, 49

key, specifying 60

specification 59

TargetType, specifying 60

using 48-51

T

templates

- about 64, 180
- Blank App (XAML) 180
- Class Library (Windows Store apps) 180
- control template 64
- data template 64
- editing 64-67
- Grid App (XAML) 180
- Split App (XAML) 180
- Windows Runtime Component 180

Text controls 9

tile brush

- about 18
- background of grid, modifying 18

time-based animation. *See* keyframe

timelines

- about 90
- recording 90

Timeline zoom feature 97

tools panel

- about 8
- assets 10
- Assets 10
- Brush tools 9
- Common controls 9
- Layout panels 9
- Path tools 9
- Selection tools 8
- Shape tools 9
- text, adding to TextBlock 13
- TextBlock, adding 10-12
- Text controls 9
- View tools 9

transforms

- using 94

transition

- between keyframes 101

transitions stage 76

U

user control

- adding, in window 148-150
- creating 143

- creating, that selects background color 144, 145
- event handlers, adding 146-148
- versus custom controls 142, 143

user interfaces

- building 36

V

vector graphics

- about 125, 126
- WPF control, zooming in to 126-128

View tools

- Camera orbit 9
- Pan 9
- Zoom 9

Visual State Manager

- about 76
- modifying, with visual states 77-83
- URL 76

visual states

- about 76
- dynamic stage 76
- static stage 76
- transition stage 76

Visual Studio

- application, integrating 20
- URL 2

Visual Studio 2012

- with Blend 1, 2

W

Windows 8 Store app

- creating 181-184

Windows Phone

- about 158
- application, creating 162-169
- application, guidelines 160
- developer, URL 176
- features 158, 159
- Windows Phone SDK, installing 158
- Windows Phone SDK, URL 158

Windows Phone Emulator

- about 160
- features 160-162
- URL 160

Windows Phone, features

- accelerometer 159
- accelerometer, URL 159
- camera and photos 159
- compass 159
- compass, URL 159
- gyroscope 159
- gyroscope, URL 159
- location 158
- lock screen 158
- maps and navigation 159
- permanent back button 159
- speech, URL 159
- tiles 158
- toast notifications 158
- wallet, URL 159

Windows Presentation Foundation (WPF) 2**Windows Store**

- apps creating, C# used 181
- apps creating, XAML used 181
- app, submitting to 184-189
- URL 184

WPF

- panels 23

X**XAML**

- about 39
- basics 40
- code 39-42
- comments 47
- elements, adding by hand-coding 45, 46
- for storyboard 98-100
- in Silverlight, URL 49
- in WPF, URL 49
- namespaces, adding 42, 43
- style, defining 48
- styles 47
- used, for creating Windows Store apps 181



Thank you for buying **Blend for Visual Studio 2012 by Example Beginner's Guide**

About Packt Publishing

Packt, pronounced 'packed', published its first book, *Mastering phpMyAdmin for Effective MySQL Management*, in April 2004, and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

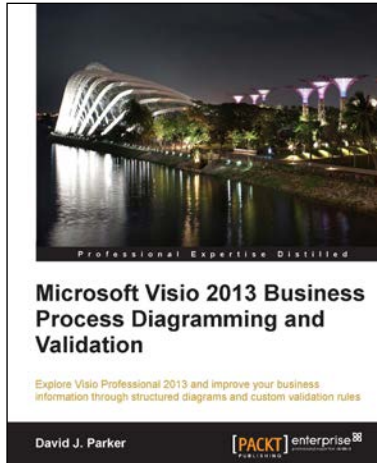
Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution-based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern yet unique publishing company that focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website at www.PacktPub.com.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, then please contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

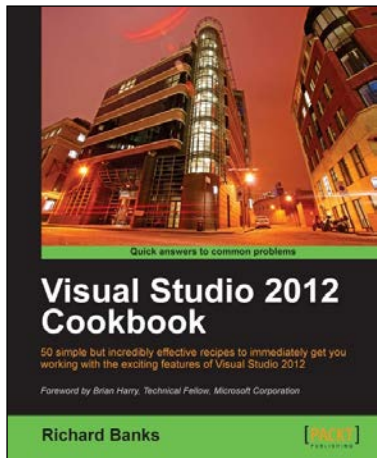


Microsoft Visio 2013 Business Process Diagramming and Validation

ISBN: 978-1-78217-800-2 Paperback: 416 pages

Explore Visio Professional 2013 and improve your business information through structured diagrams and custom validation rules

1. Optimize your business information visualization by mastering out-of-the-box structured diagram functionality with features like basic and cross-functional flowcharts.
2. Create and analyze custom validation rules for structured diagrams using Visio 2013 Professional.
3. Get to grips with the validation logic for business process diagramming with Visio 2013 Professional with the provided Rules Tools add-on.



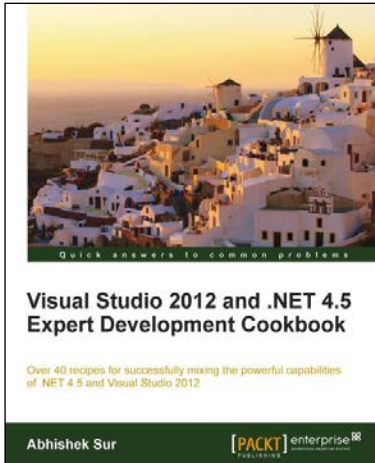
Visual Studio 2012 Cookbook

ISBN: 978-1-84968-652-5 Paperback: 272 pages

50 simple but incredibly effective recipes to immediately get you working with the exciting features of Visual Studio 2012

1. Take advantage of all of the new features of Visual Studio 2012, no matter what your programming language specialty is!
2. Get to grips with Windows 8 Store App development, .NET 4.5, asynchronous coding and new team development changes in this book and e-book.
3. A concise and practical First Look Cookbook to immediately get you coding with Visual Studio 2012.

Please check www.PacktPub.com for information on our titles

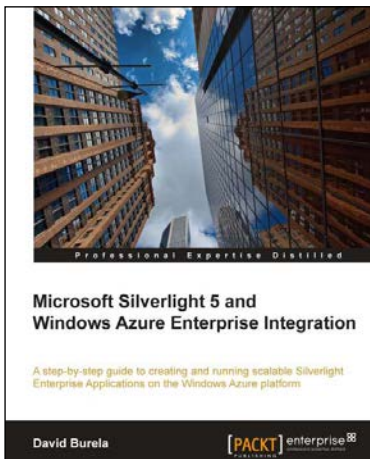


Visual Studio 2012 and .NET 4.5 Expert Development Cookbook

ISBN: 978-1-84968-670-9 Paperback: 380 pages

Over 40 recipes for successfully mixing the powerful capabilities of .NET 4.5 and Visual Studio 2012

1. Step-by-step instructions to learn the power of .NET development with Visual Studio 2012.
2. Filled with examples that clearly illustrate how to integrate with the technologies and frameworks of your choice.
3. Each sample demonstrates key concepts to build your knowledge of the architecture in a practical and incremental way.



Microsoft Silverlight 5 and Windows Azure Enterprise Integration

ISBN: 978-1-84968-312-8 Paperback: 304 pages

A step-by-step guide to creating and running scalable Silverlight Enterprise Applications on the Windows Azure platform

1. This book and e-book details how enterprise Silverlight applications can be written to take advantage of the key features of Windows Azure to create scalable applications.
2. Provides an overview of the Windows Azure platform and how the different technologies can be integrated within your enterprise application.
3. Examines ways that distributed asynchronous systems can be created to allow scalable processing.

Please check www.PacktPub.com for information on our titles