Student Guide

Information in this document is subject to change without notice. Companies, names and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Object Innovations.

Product and company names mentioned herein are the trademarks or registered trademarks of their respective owners.

® is a registered trademark of Object Innovations.

Author: Robert J. Oberg

Special Thanks: Ernani Junior Cecon

Copyright ©2012 Object Innovations Enterprises, LLC All rights reserved.

Object Innovations
877-558-7246
www.objectinnovations.com

Published in the United States of America.
# Table of Contents (Overview)

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1</td>
<td>Introduction to Silverlight</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>Using XAML</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>Shapes and Lines</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>Text and Fonts</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>Brushes and Images</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>Layout</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>Handling Events</td>
</tr>
<tr>
<td>Chapter 8</td>
<td>Animation and Media</td>
</tr>
<tr>
<td>Chapter 9</td>
<td>Application Structure and Networking</td>
</tr>
<tr>
<td>Chapter 10</td>
<td>Web Services</td>
</tr>
<tr>
<td>Chapter 11</td>
<td>Data Controls and Data Binding</td>
</tr>
<tr>
<td>Chapter 12</td>
<td>Data Access</td>
</tr>
<tr>
<td>Chapter 13</td>
<td>Additional Features</td>
</tr>
<tr>
<td>Chapter 14</td>
<td>New Features in Silverlight 5</td>
</tr>
<tr>
<td>Appendix A</td>
<td>Learning Resources</td>
</tr>
<tr>
<td>Appendix B</td>
<td>Transforms</td>
</tr>
</tbody>
</table>
Directory Structure

- Install the course software by running the self-extractor `Install_SilverCs_50.exe`.

- The course software installs to the root directory `C:\OIC\SilverCs`.
  - Example programs for each chapter are in named subdirectories of chapter directories `Chap01`, `Chap02` and so on.
  - The **Labs** directory contains one subdirectory for each lab, named after the lab number. Starter code is frequently supplied, and answers are provided in the chapter directories.
  - The **Demos** directory is provided for performing in-class demonstrations led by the instructor.

- **Data files install to the directory `C:\OIC\Data`**.
Table of Contents (Detailed)

Chapter 1: Introduction to Silverlight ................................................................. 1
  What Is Silverlight? ............................................................................................. 3
  Silverlight and AJAX .......................................................................................... 4
  Silverlight and Flash .......................................................................................... 5
  Silverlight Tools ............................................................................................... 6
  XAML .................................................................................................................. 7
  XHTML ............................................................................................................... 8
  Automatic Plug-In Download ........................................................................... 9
  Silverlight Programming Models ..................................................................... 10
  Using Visual Studio 2010 ................................................................................ 11
  Hosting the Silverlight Application ............................................................... 12
  MainPage.xaml ............................................................................................... 13
  MainPage.xaml.cs ......................................................................................... 14
  Designer Support ............................................................................................ 15
  Layout Panels ................................................................................................. 16
  SimpleHello Example ...................................................................................... 17
  Using a StackPanel .......................................................................................... 18
  Handling Events in Managed Code ............................................................... 22
  Events Window ............................................................................................... 23
  Adding Events in XAML .................................................................................. 24
  Code-Behind File ............................................................................................. 25
  Names of XAML Elements ............................................................................ 26
  Silverlight Controls ....................................................................................... 27
  Application Package (.xap) ........................................................................... 28
  SimplePackage Example .............................................................................. 29
  A More Elaborate Example ........................................................................... 32
  Lab 1 ................................................................................................................ 33
  Summary ......................................................................................................... 34

Chapter 2: Using XAML ..................................................................................... 39
  Windows Presentation Foundation ................................................................. 41
  What Is XAML? .............................................................................................. 42
  Root Element ................................................................................................. 43
  Default Namespace ....................................................................................... 44
  XAML Namespace ....................................................................................... 45
  Canvas ........................................................................................................... 46
  Canvas Example ........................................................................................... 47
  Canvas Example XAML ............................................................................... 48
  An Opaque Ellipse ....................................................................................... 49
  Property Element ......................................................................................... 50
  Property Element Example ......................................................................... 51
Chapter 3: Shapes and Lines ................................................................. 81
Shapes ........................................................................................................ 83
Size and Position .................................................................................. 84
Simple Shapes Example ......................................................................... 85
Rectangle and Ellipse .......................................................................... 86
MainPage.xaml ..................................................................................... 87
MainPage.xaml.cs ............................................................................... 88
More Shape Properties ........................................................................ 89
Opacity and Visibility .......................................................................... 90
Extended Rectangle Example .......................................................... 91
Receiving Mouse Events ...................................................................... 92
Polygon ............................................................................................... 93
Line ...................................................................................................... 94
Dashed Lines ....................................................................................... 95
Polyline .............................................................................................. 96
Path ..................................................................................................... 97
Arc Segments ...................................................................................... 98
Path Demonstration ........................................................................... 99
Bezier Curves ...................................................................................... 100
Bezier Curve Examples ...................................................................... 101
Clipping ............................................................................................... 102
Lab 3 ................................................................................................... 105
Summary ............................................................................................. 108
Chapter 4: Text and Fonts ................................................................. 117
  TextBlock .................................................................................. 119
  Text Wrapping ........................................................................... 120
  Font Attributes .......................................................................... 121
  Font Families ............................................................................ 122
  Font Weights ............................................................................ 123
  Additional TextBlock Properties ............................................. 124
  TextBlock Dimensions .............................................................. 125
  ActualWidth and ActualHeight .............................................. 126
  Rich Text Content ..................................................................... 127
  Run and LineBreak .................................................................. 128
  Lab 4 ....................................................................................... 129
  Summary .................................................................................. 130

Chapter 5: Brushes and Images .................................................. 137
  Brushes ..................................................................................... 139
  SolidColorBrush ....................................................................... 140
  Predefined Color Names ........................................................... 141
  RGB Color Space ..................................................................... 142
  Enhanced RGB Color Space .................................................. 143
  Solid Brush Demo ..................................................................... 144
  Gradient Brushes ..................................................................... 145
  Linear Gradient Brushes .......................................................... 146
  Linear Gradient Brush Examples ......................................... 147
  Default Gradient Axis ............................................................... 148
  Horizontal and Vertical Gradient Axis .................................. 149
  Color Interpolation Mode .......................................................... 150
  Spread Methods ....................................................................... 151
  Repeat and Reflect ................................................................... 152
  Alpha Channel Changes ........................................................... 153
  Painting Other Shapes ............................................................... 154
  Radial Gradient Brushes .......................................................... 155
  GradientOrigin ......................................................................... 156
  ImageBrush .............................................................................. 157
  Using an ImageBrush with Text ............................................... 158
  Image ....................................................................................... 159
  Stretch Property ....................................................................... 160
  Stretch Demonstration ............................................................. 161
  Example XAML ......................................................................... 162
  Code-Behind File ...................................................................... 163
  Lab 5 ....................................................................................... 164
  Summary .................................................................................. 165

Chapter 6: Layout ............................................................................. 177
  Layout in Silverlight ................................................................. 179
  Controlling Size ....................................................................... 180
<table>
<thead>
<tr>
<th>Chapter 11: Data Controls and Data Binding</th>
<th>323</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListBox Control</td>
<td>325</td>
</tr>
<tr>
<td>BookList Example</td>
<td>326</td>
</tr>
<tr>
<td>Book.cs</td>
<td>327</td>
</tr>
<tr>
<td>Auto-Implemented Properties</td>
<td>328</td>
</tr>
<tr>
<td>Using Items Property</td>
<td>329</td>
</tr>
<tr>
<td>Deleting a Book</td>
<td>330</td>
</tr>
<tr>
<td>Selection</td>
<td>331</td>
</tr>
<tr>
<td>Binding to a Collection</td>
<td>332</td>
</tr>
<tr>
<td>DataPager Control</td>
<td>333</td>
</tr>
<tr>
<td>Demo: Using DataPager</td>
<td>334</td>
</tr>
<tr>
<td>DisplayMode</td>
<td>336</td>
</tr>
<tr>
<td>BookPager Step 2</td>
<td>337</td>
</tr>
<tr>
<td>DataGrid Control</td>
<td>338</td>
</tr>
<tr>
<td>Simple DataGrid Example</td>
<td>339</td>
</tr>
<tr>
<td>Paging with DataGrid</td>
<td>340</td>
</tr>
<tr>
<td>Sorting with DataGrid</td>
<td>341</td>
</tr>
<tr>
<td>More about Data Binding</td>
<td>342</td>
</tr>
<tr>
<td>Dependency Properties</td>
<td>343</td>
</tr>
<tr>
<td>Binding Object</td>
<td>344</td>
</tr>
<tr>
<td>ElementName Binding</td>
<td>345</td>
</tr>
<tr>
<td>Binding to a CLR Object</td>
<td>346</td>
</tr>
<tr>
<td>Change Notification</td>
<td>347</td>
</tr>
<tr>
<td>Updating the Book Object</td>
<td>348</td>
</tr>
<tr>
<td>INotifyPropertyChanged</td>
<td>349</td>
</tr>
<tr>
<td>Notification with Collections</td>
<td>350</td>
</tr>
<tr>
<td>Book Class</td>
<td>351</td>
</tr>
<tr>
<td>MainPage.xaml.cs</td>
<td>352</td>
</tr>
<tr>
<td>Updating Collection Elements</td>
<td>353</td>
</tr>
<tr>
<td>Notification by Collection Elements</td>
<td>354</td>
</tr>
<tr>
<td>Notification in Book Class</td>
<td>355</td>
</tr>
<tr>
<td>Lab 11</td>
<td>356</td>
</tr>
<tr>
<td>Summary</td>
<td>357</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 12: Data Access</th>
<th>363</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessing Data on the Server</td>
<td>365</td>
</tr>
<tr>
<td>SmallPub Database</td>
<td>366</td>
</tr>
<tr>
<td>ADO.NET Entity Framework</td>
<td>368</td>
</tr>
<tr>
<td>EDM Example</td>
<td>369</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction to Silverlight
Introduction to Silverlight

Objectives

After completing this unit you will be able to:

• Explain the role of Silverlight in creating and delivering rich Internet applications.

• Describe the differences between Silverlight and other client-side web technologies such as AJAX and Flash.

• Download the Silverlight plug-in to run Silverlight Web applications.

• Use a XAML file to declaratively specify the user interface presented by your Silverlight content.

• Describe the two programming models available for creating Silverlight applications:
  – The JavaScript API for Silverlight
  – The Managed API for Silverlight

• Use Visual Studio 2010 to create a Silverlight application project using the Managed API.

• Use the Managed API and C# to create interactive Silverlight Web applications.
What Is Silverlight?

- Microsoft Silverlight is a client-side web technology for creating rich Internet applications incorporating features such as vector graphics, animation and multimedia.

- It is implemented as a browser plug-in, available for:
  - Internet Explorer
  - Firefox
  - Safari
  - Google Chrome

- The Silverlight 5.0 plug-in is a small download.

- Silverlight is also cross-platform, available for Windows, Mac OS X and various mobile platforms.
  - Microsoft is also supporting the development by Novell of a version running on Linux, called Moonlight.

- Silverlight 5.0 has two programming models:
  - The JavaScript API for Silverlight, using JavaScript code interpreted in the browser. This was the only API available in Silverlight 1.0.
  - The Managed API for Silverlight, using compiled code in C# or Visual Basic running on the Common Language Runtime (CLR). Also, dynamic languages like Python and Ruby can be run on the Dynamic Language Runtime (DLR).
Silverlight and AJAX

- **AJAX** is another important client-side Web technology for delivering rich Internet applications.

  - The table compares AJAX and Silverlight.

<table>
<thead>
<tr>
<th>AJAX</th>
<th>Silverlight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does not require a browser plug-in, and thus runs in a wide range of browsers on many platforms.</td>
<td>The Silverlight plug-in currently runs only with Internet Explorer, Firefox, and Safari. Supported platforms are Windows and Mac OS.</td>
</tr>
<tr>
<td>Relies on DHTML, CSS and JavaScript for achieving UI.</td>
<td>Supports a very rich UI model by virtue of its plug-in.</td>
</tr>
<tr>
<td>Programming limited to JavaScript.</td>
<td>JavaScript, C#, Visual Basic, Ruby and Python.</td>
</tr>
<tr>
<td>Rich set of controls.</td>
<td>Controls furnished by Microsoft and others by third parties</td>
</tr>
<tr>
<td>Supports out-of-band communication between client and server.</td>
<td>Supports on-demand downloading of content.</td>
</tr>
</tbody>
</table>

- It is entirely feasible to use both AJAX and Silverlight on a web page.
For several years a popular option for providing visually rich content on web pages has been Adobe’s Flash.

- The table compares Flash and Silverlight.

<table>
<thead>
<tr>
<th></th>
<th>Flash</th>
<th>Silverlight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Both a runtime component and a design tool.</td>
<td>Only a runtime component. Tools include Silverlight SDK, Visual Studio and Expression.</td>
<td></td>
</tr>
<tr>
<td>Several years old and most users already have the runtime installed.</td>
<td>Newer and so is not as ubiquitous.</td>
<td></td>
</tr>
<tr>
<td>Big learning curve for the design tool.</td>
<td>Easy to learn programming model.</td>
<td></td>
</tr>
<tr>
<td>Some visual effects that Silverlight lacks.</td>
<td>Supports higher quality video and pressure-sensitive input devices.</td>
<td></td>
</tr>
<tr>
<td>Content not very discoverable by search engines due to use of compiled script.</td>
<td>Content that is more discoverable by search engines by virtue of use of XML.</td>
<td></td>
</tr>
</tbody>
</table>
Silverlight Tools

• The Microsoft Silverlight 5 Tools for Visual Studio 2010 is an add-on for Visual Studio 2010 for developing Silverlight 5 applications\(^1\).
  
  - You may download it from the official Microsoft Silverlight Site:

  http://www.microsoft.com/silverlight/

  - It contains the Silverlight 5 runtime, the Silverlight 5 SDK, Visual Studio templates, documentation and samples.

• In this course we will use Visual Studio in our Silverlight development.
  
  - With the Visual Studio templates you can create a Silverlight project in Visual Studio, enabling programming in C# or Visual Basic.

• Silverlight applications typically consist of files of the following types:
  
  - HTML
  - XAML
  - JavaScript
  - C# or Visual Basic

\(^1\) Visual Studio 2010 comes with Silverlight 3. You install Silverlight 5 on top of it.
XAML

- In Silverlight user interface elements are created via a declarative language called XAML.
  - XAML stands for Extensible Application Markup Language.
  - XAML is based on XML.

- In our first example, we will specify a text block and an ellipse as children of a canvas container.

```xml
<Canvas>
  <Ellipse ... />
  <TextBlock>
    ...
  </TextBlock>
</Canvas>
```

- XAML is quite intuitive and is discussed in detail later in the course.
XHTML

- XHTML specifies the HTML 4.01 standard as an XML application.

- XHTML is stricter, leading to greater uniformity in how pages are rendered by different browsers.
  - It also provides a specification allowing the Web to be used by other devices, such as handheld computers and mobile phones.

- **Important differences between XHTML and HTML:**
  - The document begins with a DOCTYPE declaration.
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 ... >
  - Include a reference to the XML namespace within the HTML element.
    <html xmlns="http://www.w3.org/1999/xhtml" >
  - Supply all element names in lower case (XML is case sensitive).
  - The <head> and <body> elements **must** be included.
  - Tags must always be closed and nested correctly.
  - Attribute values must always be denoted by quotation marks (either single quote or double quote is OK).

- **Visual Studio will automatically generate valid XHTML 1.1 test pages.**
Automatic Plug-In Download

- If a Silverlight application is opened by a browser where Silverlight is not installed, this will be displayed:

- Clicking the button will automatically install the Silverlight plug-in.
Silverlight Programming Models

- **Silverlight 5 provides two distinct programming models for implementing Silverlight applications:**
  - The JavaScript API for Silverlight, available since Silverlight 1.0
  - The Managed API for Silverlight, introduced in Silverlight 2. *This course uses only the Managed API for Silverlight.*

- **These two models are disjoint.**
  - You cannot use both of them at the same time within a single instance of the Silverlight plug-in.

- **Use of the JavaScript API for Silverlight does not require any knowledge of .NET.**
  - Programming is done with HTML controls, JavaScript and the DOM.

- **The Managed API for Silverlight uses managed code running in one of these environments:**
  - The Common Language Runtime (CLR) for Silverlight. Supported languages include C# and Visual Basic.
  - The Dynamic Language Runtime (DLR) for Silverlight. Supported languages include IronPython, IronRuby and Managed JScript.

- **A subset of the .NET Framework class library is available to programs using the Managed API for Silverlight.**
Using Visual Studio 2010

- You can use Visual Studio 2010 to create and build Silverlight application projects.
  - You also need Silverlight 5 Tools for Visual Studio 2010, which also includes the Silverlight 5 SDK and the Silverlight 5 developer runtime.
  - You will then have available several Silverlight Application templates.
  - For this course you may use the free version, Visual Web Developer 2010 Express.

- As a demo, let’s create a Silverlight application *SimpleHello* in the *Demos* folder of the course.

  - See Chap01\SimpleHello for the final version of our demo.
Hosting the Silverlight Application

• A Silverlight application must be hosted in a Web page.

• Visual Studio provides two alternatives to hosting:
  – Host the Silverlight application in a new Web site.
  – Generate an HTML test page as part of the project build.

• In this chapter we’ll mainly use the second approach, which creates a simpler project and does not require ASP.NET on the server.
MainPage.xaml

- The most important file in the wizard-generated application is MainPage.xaml.

  MainPage.xaml contains XAML specifying the appearance of a Web page.

```xml
<UserControl x:Class="SimpleHello.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">

    <Grid x:Name="LayoutRoot">

    </Grid>

</UserControl>
```
MainPage.xaml.cs

- **MainPage.xaml.cs** is a code-behind file containing C# managed code.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Shapes;

namespace SimpleHello
{
    public partial class MainPage : UserControl
    {
        public MainPage()
        {
            InitializeComponent();
        }
    }
}
```
Designer Support

- Silverlight Tools for Visual Studio 2010 provides designer support for editing XAML files.
  - A Toolbox contains controls which can be dragged on to a design surface or into the XAML file.

- There is both a Design view and a XAML view.
  - By default they are shown with a horizontal split screen.
  - You can also split the screen vertically or show either view by itself.
Layout Panels

- Silverlight provides three Panel classes that can be used for the layout of child elements.
  - **Canvas** positions elements by explicit coordinates
  - **StackPanel** positions elements in a single line that can be oriented either vertically (the default) or horizontally.
  - **Grid** positions elements in a flexible grid area consisting of columns and rows.

- **Starter XAML created by Visual Studio uses Grid, which is the most flexible (but also the most complex) panel.**
  - We’ll discuss Grid later.

- **In our example we use Canvas.**

```xml
<Canvas Background="White">
  <Ellipse Width="200"
           Height="100"
           Fill="Blue">
  </Ellipse>
  <TextBlock Canvas.Top="110"
             FontSize="20">
    Hello Silverlight
  </TextBlock>
</Canvas>
```

  - Type this in, or copy from Chap01\SimpleHello.
SimpleHello Example

• You will then see the result in the Design view.

```xml
<Canvas Background="white">
  <Ellipse Width="200"
           Height="100"
           Fill="Blue">
    </Ellipse>
  <TextBlock Canvas.Top="110"
              FontSize="20">
    Hello Silverlight
  </TextBlock>
</Canvas>
```

• Build and run using Ctrl + F5.
Using a StackPanel

- Our next example illustrates using StackPanel layout.
  - See Chap01\HelloManaged\Step1.
  - Here is the visual appearance:

![Visual Appearance](image)

- We’ll create the user interface using the Designer.

1. Create in the Demos directory a new Silverlight Application called HelloManaged that is not hosted in a Web site.

2. In XAML view delete the Grid.

3. Drag a StackPanel from the Toolbox into the XAML view. Delete the closing “/>” and type “>” so that you have both a start tag and an end tag. Press Enter.

```xml
<StackPanel>
</StackPanel>
```

4. Start typing the attribute “Background” in the start tag. IntelliSense will kick in making it easy. Press = and select Beige for the background color.

```xml
<StackPanel Background="Beige">
</StackPanel>
```

5. Observe the beige color appear in Design view.
6. Drag a StackPanel nested within the first StackPanel. In the same manner as before, assign the “Orientation” attribute to be horizontal (the default is vertical).

```xml
<StackPanel Background="Beige">
  <StackPanel Orientation="Horizontal">
  </StackPanel>
</StackPanel>
```

7. Drag a TextBlock into the nested StackPanel and type “Name:” between the start and end tag. Observe this text appear in the Design window.

```xml
<StackPanel Background="Beige">
  <StackPanel Orientation="Horizontal">
    <TextBlock>
      Name:
    </TextBlock>
  </StackPanel>
</StackPanel>
```

8. Drag a TextBox into the position just below the TextBlock. Assign the x:Name attribute to be “txtName”. Observe the design view as you change the XAML file.

```xml
<StackPanel Background="Beige">
  <StackPanel Orientation="Horizontal">
    <TextBlock>
      Name:
    </TextBlock>
    <TextBox x:Name="txtName"/>
  </StackPanel>
</StackPanel>
```
Using a StackPanel (Cont’d)

9. Drag another TextBlock just below the nested StackPanel. Assign x:Name to be “tbMessage”.

10. Finally, drag a Button below the TextBlock. Provide an end tag. Assign x:Name to be “btnGreet” and content between the start and end tags of “Greet”.

   <StackPanel Background="Beige">
     <StackPanel Orientation="Horizontal">
       <TextBlock>
         Name:
       </TextBlock>
       <TextBox x:Name="txtName"/>
     </StackPanel>
     <TextBlock x:Name="tbMessage"/>
     <Button x:Name="btnGreet">
       Greet
     </Button>
   </StackPanel>

11. Observe the user interface in Design view. Not very pretty!

12. Let’s make the following changes. Experiment using both XAML and the Properties window in Design view.

   – Margin of 5 for all controls
   – Width of 100 for the TextBox and Button
   – Height of 25 for the second TextBlock
   – HorizontalAlignment of Left for the Button
Using a StackPanel (Cont’d)

13. Here is the final XAML for Step 1:

```xml
<StackPanel Background="Beige">
    <StackPanel Orientation="Horizontal">
        <TextBlock Margin="5">
            Name:
        </TextBlock>
        <TextBox x:Name="txtName" Margin="5" Width="100" />
    </StackPanel>
    <TextBlock x:Name="tbMessage" Height="25" Margin="5" />
    <Button x:Name="btnGreet" Width="100" HorizontalAlignment="Left" Margin="5">
        Greet
    </Button>
</StackPanel>
```

14. Here is the user interface in Design view:
Handling Events in Managed Code

- In our example we will handle the *Click* event of the button.
  
  - See Chap01\HelloManaged\Step2.

1. In Design view double-click the button. This will add an event handler to the code-behind file `MainPage.xaml.cs`.

2. Provide the following code:

   ```csharp
   private void btnGreet_Click(object sender, RoutedEventArgs e)
   {
       tbMessage.Text = "Hello, " + txtName.Text;
   }
   ```

3. Examine the XAML for the TextBox and for the second TextBlock. The `x:Name` attribute gives a name to a control that can then be used in the code-behind file.

   ```xml
   <StackPanel Background="Beige">
       <StackPanel Orientation="Horizontal">
           <TextBlock Margin="5">
               Name:
           </TextBlock>
           <TextBox x:Name="txtName" Margin="5" Width="100" />
       </StackPanel>
       <TextBlock x:Name="tbMessage" Margin="5" Height="25">
           Hello, <TextBlock Margin="5" x:Name="txtName" Width="100" />
       </TextBlock>
       <Button x:Name="btnGreet" Width="100" HorizontalAlignment="Left" Margin="5">
           Greet
       </Button>
   </StackPanel>
   ```
Events Window

• Double-clicking a control adds a handler for the control’s primary event.

• To add a handler for other events, click the button in the Properties window.
  – This will display a list of all the events associated with the control.
  – Double click on an event to add a handler.
Adding Events in XAML

- Visual Studio also makes it easy to create event handlers using XAML.
  - In the XAML for the control, prepare to add an attribute. You should see an IntelliSense list box showing all the properties and events.
  - Select the event of interest.

```xml
<TextBox x:Name="txtName" Margin="5" Width="100" LostFocus="txtName_LostFocus" />
</StackPanel>
<Button x:Name="btnGreet" Width="100" HorizontalAlignment="Greet"
    Margin="5">
  <%: btnGreet_Click() %>
</Button>
</StackPanel>
```

- Press the Enter key, and you will be given an opportunity to add a new event handler or use an existing handler.

```xml
LostFocus="" />
</Button>

<New Event Handler>
btnGreet_Click
</New Event Handler>
```

- Select <New Event Handler>. This will add an event handler, with proper code in both the XAML file and the code-behind file. You will be left in the XAML file.

```xml
<TextBox x:Name="txtName" Margin="5" Width="100"
LostFocus="txtName_LostFocus" />
```
4. You can navigate to the event handler in the code-behind file by right-clicking over the name of the event handler.

```xml
<TextBox Name="txtName" LostFocus="txtName_LostFocus" />
```

5. Select “Navigate to Event Handler” from the context menu.

6. You will be brought to the event handler in the code-behind file `MainPage.xaml.cs`, where you can add the desired code.

   ```csharp
   private void txtName_LostFocus(object sender, RoutedEventArgs e) {
     Debug.WriteLine("Leaving the TextBox");
   }
   ```


8. Build and run under the debugger (F5). Enter a name in the TextBox and then tab out of it. You should see the message displayed in Visual Studio’s Output window.

   - The final step of this demo is `Chap01\HelloManaged\Step3`.  

Names of XAML Elements

- To access XAML elements by name, use the \texttt{x:Name} attribute to specify a name.
  - Some XAML elements also have the \texttt{Name} attribute, which could also be used when available. All XAML elements have the \texttt{x:Name} attribute.

```xml
<StackPanel Orientation="Horizontal">
    <TextBlock Margin="5">
        Name:
    </TextBlock>
    <TextBox \texttt{x:Name="txtName"} Margin="5" Width="100"
            LostFocus="txtName_LostFocus" />
</StackPanel>
<TextBlock \texttt{x:Name="tbMessage"} Height="25"
            Margin="5" />

- You can then just use this name in the code-behind file to refer to the element.

```csharp
private void btnGreet_Click(object sender, RoutedEventArgs e)
{
    tbMessage.Text = "Hello, " + txtName.Text;
}
```
Silverlight Controls

- Our example program illustrates a number of Silverlight controls, including TextBlock, TextBox and Button

- Silverlight Tools for Visual Studio 2010 provides in the Toolbox a large palette of controls you can use.
  - You can create the corresponding XAML by dragging the control onto either the Design or XAML window.
Application Package (.xap)

- More elaborate Silverlight applications typically consist of several different elements:
  - Assemblies
  - Resource files, such as images
  - An application manifest that identifies the assemblies and the application entry point.

- The build process creates an application package, with extension .xap, that contains these elements.
  - An application package is a zip file.

- To illustrate, let’s create a new Silverlight app called SimplePackage, hosted in a new Web site.
SimplePackage Example

- The solution now has two projects:
  - The Silverlight application SimplePackage
  - The ASP.NET Web application SimplePackage.Web

- Finish creating the example by doing the following:

  1. Copy the XAML for the Canvas from the SimpleHello project into the new project, overwriting the Grid.

  ```xml
  <Canvas Background="White">
    <Ellipse Width="200"
             Height="100"
             Fill="Blue">
    </Ellipse>
    <TextBlock Canvas.Top="110"
                FontSize="20">
      Hello Silverlight
    </TextBlock>
  </Canvas>
  ```
SimplePackage Example (Cont’d)

2. Build the new solution. In the Output window notice the messages pertaining to the creation of a .xap file:

Begin Xap packaging
Creating file SimplePackage.xap
Adding SimplePackage.dll
Adding AppManifest.xaml
Xap packaging completed successfully

3. Run the new project.


   ... 
   <object data="data:application/x-silverlight-2," 
   ... 
   <param name="source" 
       value="ClientBin/SimplePackage.xap"/>
   ...
5. Examine the folder **ClientBin** in the SimplePackage.Web project. It contains the .xap file **SimplePackage.xap**.

6. Examine the file **SimplePackage.xap** in a zip program, such as WinZip.

![Image of WinZip with files]

7. It contains the application manifest **AppManifest.xaml** and the application assembly **SimplePackage.dll**.

8. Examine the file **AppManifest.xaml** in the folder **bin\Debug** in the SimplePackage project.

```xml
<Deployment xmlns="http://schemas.microsoft.com/... 
  <Deployment.Parts>
    <AssemblyPart x:Name="SimplePackage"
      Source="SimplePackage.dll" />
  </Deployment.Parts>
</Deployment>
```
A More Elaborate Example

• As a more elaborate example, build and run the Calculator solution in the chapter directory.

• The .xap file involves several additional features.

Begin Xap packaging
Creating file Calculator.xap
Adding Calculator.dll
Adding SilverlightMath.dll
Adding System.Xml.Linq.dll
Adding PLUS.jpg
Adding AppManifest.xaml
Xap packaging completed successfully

− We’ll look at this example later in the course!
Lab 1

A Dynamic Circle

In this lab, you will implement a program to display a circle whose diameter is specified in a text box. The circle will be Blue in color in a Canvas with a Seashell background. The dimensions of the canvas should be 200 x 200 pixels. The top-left of the bounding square of the circle should be located 25 pixels down and to the left of the top-left of the canvas.

Detailed instructions are contained in the Lab 1 write-up at the end of the chapter.

Suggested time: 45 minutes
Summary

- Microsoft Silverlight is a client-side web technology for creating rich Internet applications.

- Silverlight is an alternative to Adobe’s Flash and can be used in conjunction with AJAX.

- A user can download the Silverlight plug-in to run Silverlight Web applications.

- You can use XAML files to declaratively specify the user interface presented by your Silverlight content.

- Silverlight 5 provides two programming models for creating Silverlight applications:
  - The JavaScript API for Silverlight
  - The Managed API for Silverlight

- You can use Visual Studio 2010 to create a Silverlight application project.
  - Visual Studio 2010 provides an effective Designer to help you lay out controls.

- With the Managed API for Silverlight and C# you can create interactive Silverlight Web applications.
  - User input is provided through Silverlight controls.
Lab 1

A Dynamic Circle

Introduction

In this lab, you will implement a program to display a circle whose diameter is specified in a text box. The circle will be Blue in color in a Canvas with a Seashell background. The dimensions of the canvas should be 200 x 200 pixels. The top-left of the bounding square of the circle should be located 25 pixels down and to the left of the top-left of the canvas.

Suggested Time: 45 minutes

Root Directory: OIC\SilverCs

Directories: Labs\Lab1 (do your work here)
            Chap01\HelloManaged (sample with StackPanel)
            Chap01\DynamicCircle (answer)
Part 1. Layout

1. Use Visual Studio to create a new Silverlight application **DynamicCircle** in the working directory. Leave “Create a directory for solution” checked and choose to automatically generate a test page to host Silverlight at build time.

2. Review the XAML of the **HelloManaged** example for the usage of StackPanel.

3. We’re going to layout our controls using StackPanel and Canvas. Just under the UserControl there will be a StackPanel with the default vertical orientation. The top element will be a Canvas, and the second element will be a StackPanel with horizontal orientation. The second StackPanel will have as elements a TextBlock, a TextBox, and a Button. In doing the layout use either Design view or XAML view, or a mixture, whichever works best for you.

   Here is a first cut at the XAML:

   ```xml
   <StackPanel>
     <Canvas Background="SeaShell" Width="200" Height="200">
       <Ellipse Width="50" Height="50" Fill="Blue">
       </Ellipse>
     </Canvas>
     <StackPanel Orientation="Horizontal">
       <TextBlock>Diameter:</TextBlock>
       <TextBox Width="50" Text="50">
       </TextBox>
       <Button Width="75" Height="25" Content="Refresh">
       </Button>
     </StackPanel>
   </StackPanel>
   ```

4. Build and run. Compare with the screen capture showing the desired look. Note these issues:

   a. The canvas is centered with respect to the Silverlight control, rather than aligned to the left.

   b. There is no margin around the canvas or any of the controls.

   c. The circle is bumped up against the top-left of the canvas.

5. Specify a left horizontal alignment for the canvas and an overall margin of 5 pixels.

   ```xml
   <Canvas Background="SeaShell" HorizontalAlignment="Left"
   ```
6. Provide suitable margins for the controls. Try a uniform margin of 5 pixels for the text block and a left-only margin of 5 for the button. Observe the appearance in the preview window, and build and run. It’s good enough!

```xml
<TextBlock Margin="5">Diameter:</TextBlock>
<TextBox Width="50"
    Text="50">
</TextBox>
<Button Margin="5,0,0,0"
        Width="75"
        Height="25"
        Content="Refresh">
</Button>
```

7. Finally, the top-left of the bounding square of the circle should be located 25 pixels down and to the left of the top-left of the canvas. Do this by setting the `<Canvas.Left>` and `<Canvas.Right>` attributes of the Ellipse.

```xml
<Ellipse Width="50"
        Height="50"
        Canvas.Left="25"
        Canvas.Top="25"
        Fill="Blue">
</Ellipse>
```

**Part 2. Event Handler**

1. In the XAML for the Button start to type a Click attribute. You should see an IntelliSense box pop up making it easy for you to enter Click. Or you can double-click on the Button in Design view.

2. Add the New Event Handler as offered. The XAML for the Button should now look like this:

```xml
<Button Margin="5,0,0,0"
        Width="75"
        Height="25"
        Content="Refresh"
        Click="Button_Click">
</Button>
```

3. Right-click over the “Button_Click” to navigate to the event handler in the code-behind file. As a temporary implementation, show a message box with this code:

```csharp
private void Button_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Button clicked!");
}
```

4. Build and run the program and click the Refresh button. You should see a suitable message displayed.
5. Now we want to implement the real code. We need to be able to access the Ellipse, so provide a suitable \texttt{x:Name} attribute.

\begin{verbatim}
<Ellipse Width="50"
    Height="50"
    Canvas.Left="25"
    Canvas.Top="25"
    Fill="Blue"
    \texttt{x:Name="ellip"}>
</Ellipse>
\end{verbatim}

6. We also need to be able to access the TextBox, so provide a suitable \texttt{x:Name} attribute for that element as well.

\begin{verbatim}
<TextBox Width="50"
    Text="50"
    \texttt{x:Name="txtDiameter"}>
</TextBox>
\end{verbatim}

7. We can now provide code in the event handler to set the Width and Height of the Ellipse based on the value entered in the Textbox. Comment out the message box.

\begin{verbatim}
private void Button_Click(object sender, RoutedEventArgs e) {
    //MessageBox.Show("Button clicked!");
    double diameter = Convert.ToDouble(txtDiameter.Text);
    ellip.Width = diameter;
    ellip.Height = diameter;
}
\end{verbatim}

8. Build and run. Try out a new diameter of 150.
Chapter 3

Shapes and Lines
Shapes and Lines

Objectives

After completing this unit you will be able to:

• Use Silverlight to create basic shapes:
  – Rectangle
  – Ellipse
  – Polygon

• Use Silverlight to create lines and curves:
  – Line
  – Polyline
  – Path

• Draw arcs and Bezier curves.

• Create various kinds of strokes in your Silverlight applications.

• Explain the use of geometries in Silverlight applications, and use geometries for clipping.
• **In Silverlight shapes** are UIElements that enable you to draw shapes on the screen.
  
  − **UIElement** is an abstract base class from which various concrete shape classes are derived.

• **Shapes in Silverlight include the following:**
  
  − Rectangle
  − Ellipse
  − Polygon
  − Line
  − Polyline
  − Path

• **Common properties of shape objects include:**

  Fill  Brush used for painting interior of the shape
  Stroke  Brush used for painting the outline of the shape
  StrokeThickness  Thickness of the shape’s outline
Size and Position

• Additional data properties of shape objects specify size or coordinates.
  - For example, Ellipse and Rectangle have **Width** and **Height** properties.
  - A Line has coordinates **X1**, **Y1**, **X2** and **Y2**.

• Shape objects can be used inside a Canvas.

• The position of a shape can then be specified by the attached properties **Canvas.Left** and **Canvas.Top**.

• Here is an example of an ellipse, rectangle and line on a canvas.
  - See **SimpleShapes** in the chapter folder.

```xml
<Canvas Background="Seashell"  >
  <Ellipse Width="150" Height="100" Fill="Red" />
  <Rectangle Width="150" Height="100"
    Stroke="Black" StrokeThickness="1"
    Canvas.Left ="70" Canvas.Top="30"/>
  <Line X1="200" Y1="10" X2="150" Y2="160"
    Stroke="Blue" StrokeThickness="5" />
</Canvas>
```
Simple Shapes Example

- Here is how our simple shapes example will be shown in the browser:
Rectangle and Ellipse

- The most basic shapes are Rectangle and Ellipse, which share the same set of properties.

- An ellipse fits perfectly inside a rectangle of the same dimensions.

- A sample program will let you experiment with dynamically setting some basic properties of a rectangle, shown with a fixed ellipse.
  
  - See Rectangle. Click the Refresh button to update the display with the properties entered.

![Diagram of a rectangle and an ellipse with properties entered]

- Width: 200  Height: 100
- Top: 30  Left: 70
- Fill: Yellow
- Stroke: DarkGray  StrokeThickness: 5
- Refresh
MainPage.xaml

• The XAML file draws an ellipse and rectangle of the same size,

  - A x:Name is assigned to the rectangle for use by C# code.

<Canvas HorizontalAlignment="Left" Margin="5"
    Width="300" Height="200"
    Background="Seashell">

        <Ellipse Width="150" Height="100" Fill="Red" />

        <Rectangle Width="150" Height="100"
            Stroke="Black" StrokeThickness="1"
            x:Name="rect" />

</Canvas>

  - Here is how the initial shapes are drawn:
MainPage.xaml.cs

- The code-behind file dynamically sets properties of the rectangle, using coding techniques discussed previously.

```csharp
private void Button_Click(object sender, RoutedEventArgs e)
{
    rect.Height = Convert.ToDouble(txtHeight.Text);
    rect.Width = Convert.ToDouble(txtWidth.Text);
    Canvas.SetTop(rect,
                  Convert.ToDouble(txtTop.Text));
    Canvas.SetLeft(rect,
                  Convert.ToDouble(txtLeft.Text));
    rect.Fill = BrushFromString(txtColor.Text);
    rect.Stroke = BrushFromString(txtStroke.Text);
    rect.StrokeThickness =
        Convert.ToDouble(txtStrokeThickness.Text);
}
```
More Shape Properties

• You may create rounded corners.
  – **RadiusX** is the x-axis radius of an ellipse used to round the corners of a rectangle.
  – **RadiusY** is the y-axis radius of an ellipse used to round the corners of a rectangle.

• The *Stretch* property specifies how content is resized to fill its allocated shape.
  – **Fill** (the default for ellipses and rectangles) causes the shape to fill the destination dimensions.
  – **Uniform** makes the shape’s width and height equal, choosing the smaller of Width and Height, fitting inside the allocated dimensions.
  – **UniformToFill** makes the shape’s width and height equal, choosing the larger of Width and Height, clipping the shape to fit inside the allocated dimensions.
  – **None** causes the shape to occupy no space except for its strokes.
Opacity and Visibility

- **Opacity** can be set to any value between 0 and 1.
  - 0 is completely transparent.
  - 1 is completely opaque.

- **Visibility** controls whether a UI element is shown:
  - **Visible** means that the element is shown.
  - **Collapsed** means that the element is not shown.

- Note that an element with **Opacity** 0 still receives input events, such as mouse clicks, while an element whose **Visibility** is **Collapsed** does not.
Extended Rectangle Example

- An extended version of our rectangle example demonstrates these additional properties.
  - See RectangleEx.
Receiving Mouse Events

• An event handler is specified in the ellipse for MouseLeftButtonDown events.

```xml
<Ellipse Width="150" Height="100" Fill="Red"
  MouseLeftButtonDown="onMouseDown" />
```

• The C# code displays a message box.

```csharp
private void onMouseDown(object sender, MouseButtonEventArgs e) {
    MessageBox.Show("Mouse clicked over ellipse");
}
```

• To see the effect of the Visibility property, try clicking the mouse over a portion of the ellipse covered by the rectangle.

• If Visibility for the rectangle is set to Visible, the ellipse will not receive the mouse events (the rectangle will), even if the rectangle cannot be seen by virtue of Opacity being 0.
  
  – Set rectangle’s Visibility to Collapsed, and the ellipse will receive the mouse event.
Polygon

- A polygon is specified by a collection of Point objects.
- A built-in type converter converts a sequence of pairs of numbers to a sequence of points.
- Here is XAML defining a triangle:

```xaml
<Polygon Fill ="Blue"
    Points="200 20  100 100 300 100" />
```

- And here is how the polygon is displayed:

![Triangle](image)

- See Polygons\Triangle for the code example. Polygons\Pentagon displays a pentagon.
- You could conduct further experiments by creating additional XAML files.
• **A Line element draws a straight line between two points.**
  
  - The first point is specified by properties X1 and Y1.
  - The second point is specified by properties X2 and Y2.
  - In order to show, the line must also specify **Stroke** and **StrokeThickness** properties.

• **The example Lines, in the chapter folder, provides examples of some lines.**

  ```xml
  <Line Stroke ="Blue" StrokeThickness="2" X1="20" Y1="20" X2="120" Y2="20" />
  
  <Line Stroke ="Red" StrokeThickness="8" X1="20" Y1="40" X2="120" Y2="120" />
  
  <Line Stroke ="Black" StrokeThickness="5" X1="220" Y1="20" X2="220" Y2="180" StrokeDashArray="3 2" />
  ```

• **Here is the output:**

```
\[ 
\]
```
Dashed Lines

- You can create different kinds of dashed lines using the **StrokeDashArray** property.

  - This property takes a string value containing a pair of double values that indicate the pattern of dashes and gaps.

  - In this example, the dashes are 3 pixels\(^1\) and the gap is 2 pixels.

    `<Line Stroke ="Black" StrokeThickness="5" X1="220" Y1="20" X2="220" Y2="180" StrokeDashArray="3 2" />

  - Here is the output:

  ![Dashed Line Example]

- The Silverlight documentation specifies other ways to customize the stroke, including:

  - StrokeStartLineCap

  - StrokeEndLineCap

  - StrokeLineJoin

---

\(^1\) In Silverlight, the unit of measurement is pixel, as opposed to WPF where the unit of measurement is device-independent pixel.
Polyline

• A Polyline draws a sequence of connected line segments.
  – As with a Polygon you specify a sequence of Point objects, represented as pairs of numbers.
  – A Polyline is like a Polygon, except it does not automatically close the shape.
  – You will normally specify Stroke and StrokeThickness, and you can specify a Fill.

• Here is an example with two line segments.

<Polyline Fill="LightBlue"
  Stroke="Black" StrokeThickness="8"
  Points="200 20  100 100 300 100" />

• Here is the output:

– See Polylines\Twolines and Polylines\Fourlines for two code examples.
Path

- The *Path* element enables you to draw complex shapes and curves.

- The shapes and curves are described by a *Geometry* element, which is used to provide a value of the *Data* property of *Path*.

- Here is an example of a *Path* consisting of several lines and curves.

  - See Path\Step1.

```xml
<Path Stroke="Black" StrokeThickness="1">
  <Path.Data>
    <PathGeometry>
      <PathFigure StartPoint="10,10"
                  IsClosed="True">
        <LineSegment Point="10,100"/>
        <ArcSegment Point="100,100" Size="45,45" />
        <ArcSegment Point="200,100" Size="75,25"
                     SweepDirection="Clockwise"
                     IsLargeArc="True"
                     RotationAngle="10" />
      </PathFigure>
    </PathGeometry>
  </Path.Data>
</Path>
```

  - The path begins at the point (10,10) and consists of one line segment and two arc segments. The ellipse defining the last arc is rotated 10 degrees about the x-axis.

  - Since the path is closed, the terminus of the last arc segment is connected to the starting point by another line segment.
Arc Segments

• An ArcSegment represents a segment that curves along an ellipse.
  
  − The starting point is provided by the path, either the initial starting point of the path, or else the last point reached by a segment in the path.
  
  − The terminal point is specified by the Point property.
  
  − The dimensions of the ellipse are specified by RadiusX and RadiusY properties.
  
  − The SweepDirection can be set to either Counterclockwise (the default) or Clockwise.
  
  − RotationAngle specifies the angle in degrees by which the ellipse is rotated about the x-axis.

• Here is an example of ArcSegment:

  <ArcSegment Point="200,100" Size="75,25"
                SweepDirection="Clockwise"
                IsLargeArc="True"
                RotationAngle="10" />

• Here is the output:
Path Demonstration

- A good way to gain an appreciation for how *Path* and *ArcSegment* work is incrementally build a XAML file and observe the output after each step.

  - Open the project in the *Demos\Path* directory, which is backed up in *Path\Step0*. Edit the file *MainPage.xaml*.

1. Examine the XAML for the PathFigure. There starting point is (10,10) and ending point is (10, 100).

   ```xml
   <PathFigure StartPoint="10,10" IsClosed="False">
     <LineSegment Point="10,100"/>
   </PathFigure>
   ```

2. Build and run. As expected, you should see a vertical line.

3. Add this *ArcSegment* to the XAML file:

   ```xml
   <PathFigure StartPoint="10,10" IsClosed="False">
     <LineSegment Point="10,100"/>
     <ArcSegment Point="100,100" Size="45,45" />  
   </PathFigure>
   ```

4. Build and run.
Path Demonstration (Cont’d)

5. Add a second ArcSegment:

```
<PathFigure StartPoint="10,10" IsClosed="False">
  <LineSegment Point="10,100" />
  <ArcSegment Point="100,100" Size="45,45" />
  <ArcSegment Point="200,100" Size="75,25" />
</PathFigure>
```

6. Build and run.

7. So far we’ve accepted the default value of **SweepDirection**. Now set the **SweepDirection** of the second to **Clockwise**.

```
<PathFigure StartPoint="10,10" IsClosed="False">
  <LineSegment Point="10,100" />
  <ArcSegment Point="100,100" Size="45,45" />
  <ArcSegment Point="200,100" Size="75,25" SweepDirection="Clockwise" />
</PathFigure>
```
Path Demonstration (Cont’d)

8. Build and run.

9. Set **IsLargeArc** to True.

```xml
<PathFigure StartPoint="10,10" IsClosed="False">
    <LineSegment Point="10,100"/>
    <ArcSegment Point="100,100" Size="45,45"/>
    <ArcSegment Point="200,100" Size="75,25" SweepDirection="Clockwise"
        IsLargeArc="True"/>
</PathFigure>
```

10. Build and run.
11. Next, let’s rotate the last arc segment.

```xml
<PathFigure StartPoint="10,10" IsClosed="False">
  <LineSegment Point="10,100"/>
  <ArcSegment Point="100,100" Size="45,45"/>
  <ArcSegment Point="200,100" Size="75,25"
    SweepDirection="Clockwise"
    IsLargeArc="True"
    RotationAngle="10"/>
</PathFigure>
```

12. Build and run.

13. Finally, close the path:

```xml
<PathFigure StartPoint="10,10" IsClosed="True">
  ...
</PathFigure>
```

14. Build and run. We are at Step 1.
Bezier Curves

- **Bezier curves** (named after engineer Paul Bezier) are used to represent smooth curves.

- Besides end points, a Bezier curve has one or more control points that act like a “center of gravity” pulling the curve towards them.
  
  - A quadratic Bezier curve has one control point.

  ![Quadratic Bezier Curve](image)

  - A Bezier curve has two control points (also called a cubic Bezier curve).

  ![Cubic Bezier Curve](image)
Beziers Curve Examples

- Our example programs illustrate both kinds of Beziers curves.
  - See Bezier\Cubic and Bezier\Quadratic.

- Here is the XAML for a (cubic) Bezier curve.
  - We use an EllipseGeometry to show the endpoints and the control points.
  - We use a PathGeometry to draw the BezierSegment.
  - Multiple geometry elements can be contained in a GeometryGroup.

```xml
<Path Stroke="Black" StrokeThickness="1">
  <Path.Data>
    <GeometryGroup>
      <EllipseGeometry Center="50,100" RadiusX="2" RadiusY="2"/>
      <EllipseGeometry Center="100,50" RadiusX="2" RadiusY="2"/>
      <EllipseGeometry Center="200,150" RadiusX="2" RadiusY="2"/>
      <EllipseGeometry Center="250,100" RadiusX="2" RadiusY="2"/>
      <PathGeometry>
        <PathFigure StartPoint="50,100">
          <BezierSegment Point1="100,50"
                          Point2="200,150"
                          Point3="250,100"/>
        </PathFigure>
      </PathGeometry>
    </GeometryGroup>
  </Path.Data>
</Path>
```
Clipping

- All UI elements have a *Clip* property that is used to “clip” a portion of the element as defined by a geometry.

- Our example shows an ellipse clipped by a rectangle.
  - *Clip\Unclipped* show the ellipse and the rectangle that will be used for clipping.

- *Clip\Clipped* shows the clipped ellipse.
Here is the XAML that accomplishes the clipping:

```xml
<Path Fill="Red" Opacity=".2">
  <Path.Data>
    <EllipseGeometry Center="100,50" RadiusX="100" RadiusY="50"/>
  </Path.Data>
</Path>

<Path Stroke="Black" StrokeThickness="1" StrokeDashArray="2 2">
  <Path.Data>
    <RectangleGeometry Rect="50,50,200,125"/>
  </Path.Data>
</Path>

<Ellipse Width="200" Height="100" Fill="Red">
  <Ellipse.Clip>
    <RectangleGeometry Rect="50,50,200,125"/>
  </Ellipse.Clip>
</Ellipse>
```
Lab 3

Properties of Rectangles

In this lab, you will implement a program to dynamically update the properties of a rectangle. The lab will both provide a review of XAML and C# programming and will also let you experiment with various properties (many of which are shared by other shapes in Silverlight).

Detailed instructions are contained in the Lab 3 write-up at the end of the chapter.

Suggested time: 60 minutes
Summary

• With Silverlight you can create basic shapes:
  – Rectangle
  – Ellipse
  – Polygon

• You can also create lines and curves:
  – Line
  – Polyline
  – Path

• You can draw arcs and Bezier curves.

• You can create various kinds of strokes, such as ones with dashes.

• Geometry objects enable you to define the geometry of two-dimensional shapes.
  – There are a number of applications of geometries, including specifying more complex objects to draw and defining clip regions.
Lab 3

Properties of Rectangles

Introduction

In this lab, you will implement a program to dynamically update the properties of a rectangle. The lab will both provide a review of XAML and C# programming and will also let you experiment with various properties (many of which are shared by other shapes in Silverlight).

Suggested Time: 60 minutes

Root Directory: OIC\SilverCs

Directories: Labs\Lab3 (do your work here)
            Chap03\Rectangle (answer to Part 1)
            Chap03\RectangleEx (answer to Part 2)

Part 1. Basic Rectangle Demonstration

1. Use Visual Studio to create a new Silverlight application Rectangle in the working directory. Leave “Create a directory for solution” checked and choose to automatically generate a test page to host Silverlight at build time.

2. We’re going to layout our controls using StackPanel and Canvas. Just under the UserControl there will be a StackPanel with the default vertical orientation. The top element will be a Canvas, and the remaining elements will be StackPanels with horizontal orientation. Make the Width and Height of the User Control 400 and 300.

3. Add to the XAML file markup to create an ellipse in the top left of the canvas and a rectangle with the same dimensions. Fill the ellipse with red, and outline the rectangle with a black stroke. Provide the name “rect” for the rectangle to make it easy to programmatically change its properties.
4. Build and run and examine the appearance.

![Ellipse and Rectangle]

5. Below the Canvas add a StackPanel with horizontal orientation. Provide two pairs of `<TextBlock>` and `<TextBox>` elements to let the user enter a Width and a Height. Use margins to achieve an attractive appearance. Provide `x:Name` attributes of `txtWidth` and `txtHeight` for the text boxes.

```xml
<StackPanel Orientation="Horizontal">
  <TextBlock Margin="5,4,5,0">Width:</TextBlock>
  <TextBox Width="50"
    x:Name="txtWidth"
    Text="200">  
  </TextBox>
  <TextBlock Margin="5,4,5,0">Height:</TextBlock>
  <TextBox Width="50"
    x:Name="txtHeight"
    Text="100">
  </TextBox>
</StackPanel>
```

6. Add another StackPanel containing a button with Content of “Refresh”.

```xml
<StackPanel Orientation="Horizontal">
  <Button Margin="5,5,0,0"
    Width="75" Height="25"
    Content="Refresh" />
</StackPanel>
```

7. Examine the controls you’ve added in the Design window.

![Width: 200 Height: 100]

8. Add a handler for the Click event of the Refresh button.

```xml
<Button Margin="5,5,0,0"
  Width="75" Height="25"
  Content="Refresh" Click="Button_Click" />
```
9. Implement the handler of the Refresh button. All you need to do is to set the Height and Width properties of the rectangle to the values entered in the text boxes.

```csharp
private void Button_Click(object sender, RoutedEventArgs e)
{
    rect.Height = Convert.ToDouble(txtHeight.Text);
    rect.Width = Convert.ToDouble(txtWidth.Text);
}
```

10. Examine the output in the browser and click Refresh. You should see the rectangle widen.

![Rectangle widen](image)

11. Add XAML to `MainPage.xaml` that will let the user enter new values for the `Canvas.Top`, `Canvas.Left`, `Fill`, `Stroke`, and `StrokeThickness` properties. Provide initial values as shown.

   ![Input fields for rectangle properties](image)

12. You will see the bottom part of your controls cut off because the UserControl does not have enough DesignHeight. Increase the DesignHeight to allow for all your controls, including additional ones you’ll add in Part 2.

   ```xml
   <UserControl x:Class="Rectangle.MainPage"
               ...
               d:DesignHeight="450" d:DesignWidth="400">
   ...
   </UserControl>
   ```

13. Enter code in `MainPage.xaml.cs` that will update the rectangle with the values supplied by the user. Use the `BrushFromString()` helper method we introduced in the previous chapter.

```csharp
private void Button_Click(object sender, RoutedEventArgs e)
{
    rect.Height = Convert.ToDouble(txtHeight.Text);
    rect.Width = Convert.ToDouble(txtWidth.Text);
    Canvas.SetTop(rect, Convert.ToDouble(txtTop.Text));
    Canvas.SetLeft(rect, Convert.ToDouble(txtLeft.Text));
    Canvas.SetTop(rect, Convert.ToDouble(txtTop.Text));
    Canvas.SetLeft(rect, Convert.ToDouble(txtLeft.Text));
```
rect.Fill = BrushFromString(txtColor.Text);
rect.Stroke = BrushFromString(txtStroke.Text);
rect.StrokeThickness = Convert.ToDouble(txtStrokeThickness.Text);
}

private Brush BrushFromString(string strColor)
{
    string xamlString = "<Canvas
        Background="" + strColor + ">
    Canvas c = (Canvas) XamlReader.Load(xamlString);
    SolidColorBrush br = (SolidColorBrush)c.Background;
    return br;
}


using System.Windows.Markup;

15. Build and run. Examine the output after you click Refresh.

Part 2. Extended Rectangle Demonstration

1. Add XAML to MainPage.xaml that will let the user enter new values for the RadiusX, RadiusY, and Opacity properties. Provide initial values as shown.

   RadiusX: 20  RadiusY: 20
   Opacity: .5

2. Enter code in MainPage.xaml.cs that will update the rectangle with the values supplied by the user.

   private void Button_Click(object sender, RoutedEventArgs e)
   {
   ...rect.RadiusX = Convert.ToDouble(txtRadiusX.Text);
   rect.RadiusY = Convert.ToDouble(txtRadiusY.Text);
   rect.Opacity = Convert.ToDouble(txtOpacity.Text);
   }
3. Build and run. Examine the output.

4. Add XAML to `MainPage.xaml` that will let the user enter new values for the `Visibility` and `Stretch` properties. Use combo boxes.

```xml
<StackPanel Orientation="Horizontal">
    <TextBlock Margin="5,9,5,0">Opacity:</TextBlock>
    <TextBox Margin="0,5,0,0" Width="50"
        x:Name="txtOpacity"
        Text="0.5">
    </TextBox>
    <TextBlock Margin="5,9,5,0">Visibility:</TextBlock>
    <ComboBox Margin="0,5,0,0" Width="90"
        x:Name="cmbVisibility">
    </ComboBox>
</StackPanel>

<StackPanel Orientation="Horizontal">
    <TextBlock Margin="5,9,5,0">Stretch:</TextBlock>
    <ComboBox Margin="0,5,0,0" Width="110"
        x:Name="cmbStretch">
    </ComboBox>
</StackPanel>

5. We will initialize the combo boxes in code. To this end, add to the outermost `StackPanel` a handler for the `Loaded` event.

```xml
<StackPanel x:Name="LayoutRoot" Background="White"
            Loaded="LayoutRoot_Loaded">
```

6. For `Visibility` the options should be Visible and Collapsed, with the initial selection Visible. For `Stretch` the options should be Fill, Uniform, UniformToFill and None, with the initial selection Fill. This initialization will be done in `LayoutRoot_Loaded()`.

```csharp
private void LayoutRoot_Loaded(object sender, RoutedEventArgs e)
{
    cmbVisibility.Items.Add("Visible");
    cmbVisibility.Items.Add("Collapsed");
    cmbVisibility.SelectedIndex = 0;
    cmbStretch.Items.Add("Fill");
    cmbStretch.Items.Add("Uniform");
    cmbStretch.Items.Add("UniformToFill");
    cmbStretch.Items.Add("None");
```
7. Build and run. For the new elements you should see this:

Opacity: 0.5 Visibility: Visible
Stretch: Fill

8. In `MainPage.xaml.cs` provide a helper method `VisibilityFromString()` that will obtain the Visibility corresponding to the selection in the combo box. Similarly, provide a helper method `StretchFromString()`.

```csharp
private Visibility VisibilityFromString(string str)
{
    Visibility result = Visibility.Visible;
    if (str == "Collapsed")
        result = Visibility.Collapsed;
    return result;
}

private Stretch StretchFromString(string str)
{
    Stretch result = Stretch.None;
    if (str == "Fill")
        result = Stretch.Fill;
    else if (str == "Uniform")
        result = Stretch.Uniform;
    else if (str == "UniformToFill")
        result = Stretch.UniformToFill;
    return result;
}
```

9. Add code to the handler of the Refresh button that will update the rectangle with the values supplied by the user in the two combo boxes.

```csharp
private void Button_Click(object sender, RoutedEventArgs e)
{
    ...
    rect.Visibility = VisibilityFromString(cmbVisibility.SelectedItem.ToString());
    rect.Stretch = StretchFromString(cmbStretch.SelectedItem.ToString());
}
```

10. Build and run. For Stretch select “Uniform”. Click Refresh. The rectangle will now assume a square shape, as shown in the screenshot on the following page.
11. For Visibility select “Collapsed” and click Refresh. The rectangle will now disappear!

12. In **MainPage.xaml** specify a handler in the Ellipse for the **MouseLeftButtonDown** event.

```xml
<Ellipse Width="150" Height="100" Fill="Red"
    MouseLeftButtonDown="Ellipse_MouseLeftButtonDown" />
```

13. In **MainPage.xaml.cs** implement this handler by displaying a message box.

```csharp
private void Ellipse_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)
{
    MessageBox.Show("Mouse clicked over ellipse");
}
```

14. Build and run. With the rectangle still collapsed, verify that clicking anywhere over the ellipse will bring up the message box. Now make the rectangle visible and verify that clicking over the part of the ellipse covered by the rectangle will *not* bring up the message box. Verify that this behavior still occurs if the opacity of the rectangle is set to 0.

**Part 3. Experiments.**

In this final part you will perform various experiments with setting properties to various values. This exercise is open-ended. Do at least try the following two experiments with setting values of **RadiusX** and **RadiusY**.

1. Leaving **RadiusX** at 20, make **RadiusY** equal to 40. The rounded corners will now not be symmetrical. Set the **Stroke** to Black, making the outline clearer. Here is the output:
2. Make **RadiusX** equal to half the **Width**. Make **RadiusY** equal to half the **Height**. Now you will get an ellipse!
Chapter 8

Animation and Media
Animation and Media

Objectives

After completing this unit you will be able to:

• Implement animation in your Silverlight applications.

• Use the MediaElement object to play video and audio files.

• Store media files as resources in your Silverlight applications.
Manual Animation

• The ordinary use of the term “animation” refers to simulating motion via a succession of images with small changes from one to the next.

• In Silverlight, “animation” has a specific definition: varying a property over time.

• You can achieve this kind of animation manually using XAML and .NET without any special additional features of Silverlight.
  – You can use the BackgroundWorker class in the System.ComponentModel namespace.
  – For an example, see ManualStretch in the chapter folder.
Manual Animation Example

- Let’s demonstrate a Silverlight application to stretch an ellipse.
  - See ManualStretch.
  - Build and run. Click the Stretch button.

- This works, but there a couple of drawbacks:
  - The animation is not very smooth.
  - The code is a bit complicated (don’t worry about it – we’ll see a simpler way!)
Silverlight Animation

- Silverlight makes it possible for you to specify an animation in a declarative manner using XAML.
  - See SimpleAnimation for a simple example.
  - It performs the same animation we did manually.

```xml
<Ellipse Width="50" Height="50" Fill="Red"
    x:Name="ellip">
  <Ellipse.Resources>
    <Storyboard x:Name="story"
        Storyboard.TargetName="ellip">
      <DoubleAnimation
          From="50"
          To="150"
          Duration="0:0:2.5"
          Storyboard.TargetProperty="Width"/>
    </Storyboard>
  </Ellipse.Resources>
</Ellipse>
```

- We are going to animate the **Width** property of an ellipse named **ellip**.

- The **Storyboard** object provides target object and target property information for its child animations via attached properties. In this case the target object is **ellip**.

- There may be multiple child animations. In this case there is a single **DoubleAnimation**, which specifies the animation of the **Width** property of the target object.
Storyboard Methods

- The Storyboard object has methods for controlling the animation:
  - Begin()
  - Stop()
  - Pause()
  - Resume()

- The XAML specifies a name for the storyboard.
  
  <Storyboard x:Name="story"
              Storyboard.TargetName="ellip">

- This name is used by the associated C# code.

  private void btnBegin_Click(object sender, RoutedEventArgs e)
  {
      story.Begin();
  }

  private void btnStop_Click(object sender, RoutedEventArgs e)
  {
      story.Stop();
  }

  private void btnPause_Click(object sender, RoutedEventArgs e)
  {
      story.Pause();
  }

  ...
User Interface

- The user interface shows the ellipse and provides buttons for controlling the animation.

- The Begin button starts the animation. Notice how much smoother the movement is than that of the manual implementation!

- The Stop button terminates the animation and restores the property to the starting condition.

- The Pause button pauses the animation where it is, and it can be picked up from where it left off by the Resume button.
AnimatedStretch Example

- We provide a more full-blown version of the example, allowing the user some control over the animation.
  - See AnimatedStretch.

- The XAML is more concise, because some of the properties are set in C# code.

```xml
<Ellipse Width="50" Height="50" Fill="Red"
  x:Name="ellip">
  <Ellipse.Resources>
    <Storyboard x:Name="story"
      Storyboard.TargetName="ellip">
      <DoubleAnimation x:Name="anim"
        Storyboard.TargetProperty="Width"/>
    </Storyboard>
  </Ellipse.Resources>
</Ellipse>
```
private void btnBegin_Click(object sender, RoutedEventArgs e)
{
    string from = txtFrom.Text;
    string to = txtTo.Text;
    string by = txtBy.Text;
    string duration = txtDuration.Text;
    if (from != "")
        anim.From = Convert.ToDouble(from);
    if (to != "")
        anim.To = Convert.ToDouble(to);
    if (by != "")
        anim.By = Convert.ToDouble(by);
    if (duration != "")
    {
        char[] seps = {':'};
        string[] parts = duration.Split(seps);
        int hours = Convert.ToInt32(parts[0]);
        int minutes = Convert.ToInt32(parts[1]);
        double seconds = Convert.ToDouble(parts[2]);
        int sec = (int)seconds;
        int milli = Convert.ToInt32((seconds - sec) * 1000.0);
        TimeSpan ts = new TimeSpan(0, hours, minutes, 0, sec, milli);
        anim.Duration = new Duration(ts);
    }
    story.Begin();
}

private void btnStop_Click(object sender, RoutedEventArgs e)
{
    story.Stop();
}
DoubleAnimation

- The *DoubleAnimation* object animates the value of a numeric property between two target variables over a specified duration.
  - The numeric value is *double*.
  - Linear interpolation is used in the animation.

- You can specify the target values with the properties:
  - *From* is the starting value. If not specified, use the base value.
  - *To* is the ending value. If not specified, use the base value.
  - *By* is the total amount by which the property changes. If both *From* and *By* are specified, the ending value is *From* + *By*.

- *Duration* specifies the length of time to run the animation.
  - The most common format of the string is hours:minutes:seconds.
  - For example, 0:0:2.5 represents 2.5 seconds.
ColorAnimation

- **The ColorAnimation object animates the value of a Color property between two target variables over a specified duration.**
  
  - For color animation to work you must have an actual **Color** property, such as in a brush, and not rely on a type converter in a property such as **Fill** or **Foreground**.

  - See **AnimateColor** for an example.
ColorAnimation Example

- Here is the XAML.
  - Note use of **SolidColorBrush** so that we have a Color property.

```xml
<Rectangle Width="300" Height="200"
  Name="rect">
<Rectangle.Fill>
  <SolidColorBrush x:Name="brush" Color="Green" />
</Rectangle.Fill>
<Rectangle.Resources>
  <Storyboard x:Name="story"
    Storyboard.TargetName="rect">
    <ColorAnimation x:Name="anim"
      Storyboard.TargetName="brush"
      Storyboard.TargetProperty="Color"/>
  </Storyboard>
</Rectangle.Resources>
</Rectangle>
```

- A helper method used in the code-behind:

```csharp
private Color ColorFromString(string strColor)
{
    string xamlString = "<Canvas
Background="" + strColor + "/">
    Canvas c = (Canvas) XamlReader.Load(xamlString);
    SolidColorBrush br =
        (SolidColorBrush)c.Background;
    return br.Color;
}
```
private void btnBegin_Click(object sender, RoutedEventArg s e)
{
    string fromColor = txtFromColor.Text;
    string toColor = txtToColor.Text;
    string duration = txtDuration.Text;
    if (fromColor != "")
        anim.From = ColorFromString(fromColor);
    if (toColor != "")
        anim.To = ColorFromString(toColor);
    if (duration != "")
    {
        char[] seps = { ':' };
        string[] parts = duration.Split(seps);
        int hours = Convert.ToInt32(parts[0]);
        int minutes = Convert.ToInt32(parts[1]);
        double seconds = Convert.ToDouble(parts[2]);
        int sec = (int)seconds;
        int milli = Convert.ToInt32(
            (seconds - sec) * 1000.0);
        TimeSpan ts = new TimeSpan(0, hours, minutes,
        sec, milli);
        anim.Duration = new Duration(ts);
    }
    story.Begin();
}

private void btnStop_Click(object sender, RoutedEventArg s e)
{
    story.Stop();
}
• The \textit{PointAnimation} object animates the value of a \textit{Point} property between two target variables over a specified duration.

• This kind of animation is useful for animating the motion of an object whose center is specified by a \textit{Point} by means of a geometry object.
  
  – See \texttt{MoveCircle\Step2} for an example.

    

    FromColor: Blue  
    ToColor: Red  

    From: 25,25  
    To: 300,175  

    Duration (sec): 5  

    Begin  
    Stop  
    Pause  
    Resume  

    
    – You will implement this example in the lab!
Audio and Video

- Silverlight makes it very easy to play audio and video files with the *MediaElement* object.
  - *MediaElement* defines a rectangular region that can display video on its surface and/or play audio.
  - If there is audio alone, there is no visual display, but *MediaElement* still acts as a player object.
  - You can explicitly set *Width* and *Height* of *MediaElement*, but for better performance it’s usually better to not specify them and let the media display at its natural size.

- Here are some important properties and methods of *MediaElement*:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoPlay</td>
<td>If true (the default), the media will begin playback automatically when the <em>Source</em> property is set.</td>
</tr>
<tr>
<td>Source</td>
<td>A string that specifies the source of the media</td>
</tr>
<tr>
<td>Play</td>
<td>Play the media from its current position</td>
</tr>
<tr>
<td>Pause</td>
<td>Pause the media at its current position</td>
</tr>
<tr>
<td>Stop</td>
<td>Stop and reset the media, so that it will next play from the beginning position</td>
</tr>
</tbody>
</table>
MediaElement Example

- See *Media\Resource*.
  - The nice butterfly and bear file is from Microsoft’s Silverlight SDK.
MediaElement Example (Cont’d)

• Here is the XAML:

```xml
<MediaElement Source="ButterflyBear.wmv"
               AutoPlay="False"
               x:Name="me"/>
```

• Here is the code-behind:

```csharp
private void btnPlay_Click(object sender, RoutedEventArgs e)
{
    me.Play();
}

private void btnPause_Click(object sender, RoutedEventArgs e)
{
    me.Pause();
}

private void btnStop_Click(object sender, RoutedEventArgs e)
{
    me.Stop();
}
```
**Resources**

- Silverlight employs the resource system of .NET.

- A resource is a part of a program other than code.
  - You can think of resources as images, audio files, video files, and so on.

- A typical way of defining a resource is to add the file to the Visual Studio project and set a Build Action of *Resource* for it.
  - We did that with the **ButterflyBear.wmv** video file in the *Media\Resource* example.

- By doing this, the resource will be packaged into the project assembly.
Loose Files as Resources

- An alternative to this is to use the **Content** build action.
  - We did this in the **Media\Content** example.
  - But if you do so, you should change the **Copy to Output Directory** property to “Copy always” or “Copy if newer”, because the resource won’t be embedded in the assembly.
  - As the resource will now be a loose file in the output directory, you can easily update the file without needing to rebuild or deploy the application again.
Lab 8

Animating a Circle

In this lab, you will implement a program that animates a circle, moving between two designated points on the canvas. The fill color will also change as the circle moves. The movement is automatically reversed, and the whole operation is repeated. On the repetition the color stays the same. You are provided with a starter user interface as a XAML file and stub C# code.

Detailed instructions are contained in the Lab 8 write-up at the end of the chapter.

Suggested time: 45 minutes
Summary

- You can implement animation in your Silverlight applications either manually via the .NET `BackgroundWorker` class or in a declarative manner with XAML.

- You can use the `MediaElement` object to play video and audio files.

- You can store a media file as a resource embedded in the assembly or as a loose file in the output directory.
Lab 8

Animating a Circle

Introduction

In this lab, you will implement a program that animates a circle, moving between two designated points on the canvas. The fill color will also change as the circle moves. The movement is automatically reversed, and the whole operation is repeated. On the repetition the color stays the same. You are provided with a starter user interface as a XAML file and stub C# code.

Suggested Time: 45 minutes

Root Directory: OIC\SilverCs

Directories: Labs\Lab8\MoveCircle (do your work here)
             Chap08\MoveCircle\Step0 (backup of starter code)
             Chap08\MoveCircle\Step1 (answer to Part 1)
             Chap08\MoveCircle\Step2 (answer to Part 2)

Part 1. Animating Movement of the Circle

In this first part you will animate the movement of the circle. You’ll need to provide the proper XAML declarations and C# code. Before starting to code, think about the various things you’ll need to do. The basic job is to provide the appropriate storyboard and animation in the XAML file. But what kind of animation? A PointAnimation will be ideal, as this enables animating the center of the circle. How do you draw a circle given its center? The Ellipse object won’t work, but the EllipseGeometry object will.

1. Open up MainPage.xaml in the starter code in Visual Studio. Replace the XAML using Ellipse by the equivalent using EllipseGeometry, nested inside a <Path.Data> element.

   <Path Fill="Blue">
     <Path.Data>
       <EllipseGeometry Center="25,25"
                         RadiusX="25" RadiusY="25"
                         x:Name="ellipGeo" />
     </Path.Data>
   </Path>

2. Provide a storyboard nested inside <Path.Resources>. The storyboard should have a name so that you can reference it in code-behind. Nested inside the storyboard is a <PointAnimation> element, which should also have a name for referencing in code. Specify also the Storyboard.TargetName and Storyboard.TargetProperty attached properties.
3. In `onBegin()`, initialize the `From`, `To` and `Duration` properties with the values entered in the user interface. Call the `Begin()` method of the storyboard.

```csharp
private void onBegin(object sender, RoutedEventArgs e)
{
    string from = txtFrom.Text;
    string to = txtTo.Text;
    string duration = txtDuration.Text;

    char[] seps = { ',', ' '};
    string[] parts = from.Split(seps);
    double x = Convert.ToDouble(parts[0]);
    double y = Convert.ToDouble(parts[1]);
    animP.From = new Point(x, y);

    parts = to.Split(seps);
    x = Convert.ToDouble(parts[0]);
    y = Convert.ToDouble(parts[1]);
    animP.To = new Point(x, y);

    int seconds = Convert.ToInt32(duration);
    TimeSpan ts = new TimeSpan(0, 0, 0, seconds);
    animP.Duration = new Duration(ts);

    story.Begin();
}
```

4. Test. You should see the circle move from the top-left to the bottom-right.

5. But you can’t Stop, Pause or Resume the circle. There is stub code for handlers of these buttons, which you can fill in, calling the appropriate method of the storyboard in each case.

```csharp
private void onStop(object sender, RoutedEventArgs e)
{
    story.Stop();
}

private void onPause(object sender, RoutedEventArgs e)
{
    story.Pause();
}
```
private void onResume(object sender, RoutedEventArgs e) {
    story.Resume();
}

Part 2. Animating the Color

1. In MainPage.xaml replace the initialization of Fill using a type converter by explicitly providing a SolidColorBrush, which is given a x:Name. This is necessary for us to be able to specify a target property in the ColorAnimation that we are going to use.

```xml
<Path>
    <Path.Fill>
        <SolidColorBrush x:Name="brush" Color="Blue" />
    </Path.Fill>
</Path>
```

2. Add the XAML for a ColorAnimation within the storyboard.

```xml
<Storyboard x:Name="story">
    <PointAnimation x:Name="animP"
        Storyboard.TargetName="ellipGeo"
        Storyboard.TargetProperty="Center"/>
    <ColorAnimation x:Name="animC"
        Storyboard.TargetName="brush"
        Storyboard.TargetProperty="Color"/>
</Storyboard>
```

3. In onBegin(), initialize the From and To properties of animC.

```csharp
private void onBegin(object sender, RoutedEventArgs e) {
    string from = txtFrom.Text;
    string to = txtTo.Text;
    string fromColor = txtFromColor.Text;
    string toColor = txtToColor.Text;
    string duration = txtDuration.Text;
    ...

    if (fromColor != "")
        animC.From = ColorFromString(fromColor);
    if (toColor != "")
        animC.To = ColorFromString(toColor);

    int seconds = Convert.ToInt32(duration);
    TimeSpan ts = new TimeSpan(0, 0, 0, seconds);
    animP.Duration = new Duration(ts);

    story.Begin();
}
4. Test. The animation basically works, but the color animation completes first, because we’ve not initialized its **Duration** property. Initialize it to be the same as the **Duration** of the **PointAutomation** object.

```csharp
int seconds = Convert.ToInt32(duration);
TimeSpan ts = new TimeSpan(0, 0, 0, seconds);
animP.Duration = new Duration(ts);
animC.Duration = animP.Duration;
```

5. Test. It should now work!

6. In the XAML file specify that for both animations **AutoReverse** is true.

```xml
<Storyboard Name="story">
  <PointAnimation x:Name="animP"
      Storyboard.TargetName="ellipGeo"
      Storyboard.TargetProperty="Center"
      AutoReverse="True"/>
  <ColorAnimation x:Name="animC"
      Storyboard.TargetName="brush"
      Storyboard.TargetProperty="Color"
      AutoReverse="True"/>
</Storyboard>
```

7. Test.

8. Finally, for just the **PointAnimation**, specify a 2x repetition.

```xml
<PointAnimation x:Name="animP"
    Storyboard.TargetName="ellipGeo"
    Storyboard.TargetProperty="Center"
    AutoReverse="True" RepeatBehavior="2x"/>
```

9. Test. The final behavior should now be operational.