Foreword

SILVERLIGHT IS A cross-browser, cross-platform plugin that enables developers to build rich media and RIA Web experiences. It allows developers to use .NET within the browser to create scenarios that can’t be achieved using any other Web technology, and to use high productivity tools like Visual Studio and Expression Studio when doing so.

The new Silverlight 3 release delivers more than 200 new features for developers to leverage, including the ability to run Silverlight applications in or out of the browser; stream HD and H.264 video; display 3D graphics and use GPU acceleration; take advantage of richer data-binding support; use new bitmap APIs and animation easing capabilities; and much, much more.

Ashraf Michail, the author of the book you are holding in your hands, is one of the star developers of the Silverlight team. He has been involved in the architecture since we first began working on it and brought to the effort his deep experience with and understanding of Silverlight’s “big brother,” the Windows Presentation Foundation. Ashraf knows intimately the ins and outs of the Silverlight runtime, and in this book he has distilled his understanding into a clear, concise exposition.

This isn’t the only book on Silverlight 3, by any means: I’m happy to say that several other good Silverlight 3 books are in the works. However, this is the only book that focuses on the Silverlight runtime and gives you an in-depth understanding of how it works. Once you know what’s really happening in the runtime engine, you’ll be in a position to take full advantage of Silverlight’s speed and power. Look for the “Under the Hood” section at the end of each chapter of this book: You won’t find that anywhere else.
Also look for the “Technical Insight,” “Debugging Tip,” and “Performance Tip” sections scattered throughout the book. These are the nuggets that you can take away and use to make your own applications fly. Is too much data slowing down your ComboBox display? You can filter your Items Source to speed that up, as explained in Chapter 10. Would your application benefit from GPU acceleration? Chapter 12 shows you how to make that happen, explains the four stages of the GPU acceleration pipeline—and explains the tradeoffs involved in GPU acceleration on each supported operating system so that you can understand why turning on GPU acceleration might not always help.

The book you are holding contains the keys to writing great Silverlight 3 applications. Use them well!

—Scott Guthrie,
corporate vice president,
Microsoft Developer Division
Silverlight is a rapidly growing Web technology designed to deliver media and rich Internet applications on multiple operating systems and Web browsers. Silverlight consists of a runtime, a set of tools, and libraries for your applications. End users can download the free runtime from http://www.silverlight.net to run Silverlight applications.

You can develop Silverlight applications with Visual Studio using familiar .NET languages. Designers can create artwork and animation for Silverlight using tools such as Expression Blend and Expression Design or import assets from other popular design tools. You can create video playback experiences with tools such as Expression Media Encoder or image zoom experiences with Deep Zoom Composer.

In addition to the default libraries built into the Silverlight runtime, you can download a variety of libraries from http://www.codeplex.com/Silverlight. You can use these libraries in your applications, modify the source code, or use it as an example for how to write Silverlight components.

In this book, you will learn how to build a Silverlight application and use features such as graphics, text, input, animation, layout, media, controls, and data binding. In addition to information on how to use Silverlight features, this book describes feature design principles and how those features work “under the hood.” The design principles help you understand whether the feature meets the needs of your application. The “under the hood” information gives you a look at how the feature is implemented in Silverlight so that you can get the most out of Silverlight. For example, the
Under the Hood sections give you a deeper understanding of the behavior, performance characteristics, and limits of a feature.

**Who Should Read This Book?**

If you want to get started writing your first Silverlight applications or if you are an expert Silverlight developer, you will learn a lot from this book. For the beginner, this book explains the concepts required to write your applications. For the advanced Silverlight developer, you will find in-depth information on how Silverlight works “under the hood” and how to use that information to get the most out of Silverlight.

**Prerequisites**

Before reading this book, you should be familiar with how to build a basic Web page and have a working knowledge of how to write .NET applications in C#.

**Organization**

Each chapter in this book contains three sections: Principles, Features, and Under the Hood. The Principles indicate the feature design goals to help you determine if that feature will meet your application needs. The Features section describes how you can use the feature. The Under the Hood section describes how the feature is implemented in Silverlight and how you can use that information to improve your applications. Throughout the book, there are many Performance Tips and Technical Insights.

The book consists of the following 12 chapters:

- Chapter 1, “Silverlight Overview,” is an introduction to Silverlight and the content in the remainder of the book. Chapter 1 highlights the features that are new in Silverlight 3 and not available in Silverlight 2.
- Chapter 2, “Applications,” explains how to create and deploy a Silverlight application including an explanation of all the critical components.
• Chapter 3, “Graphics,” presents an in-depth explanation of all the graphics primitives.
• Chapter 4, “Text,” explains how to display high quality text.
• Chapter 5, “Input Events,” provides information on using mouse and keyboard input in your application.
• Chapter 6, “Animation,” describes how to add animations to your applications.
• Chapter 7, “Layout,” describes how to use the layout system to automatically size and position elements in your application.
• Chapter 8, “Media,” explains how you can make a video player with Silverlight.
• Chapter 9, “Controls,” explains how you can use styles, customize built-in controls, and create your own customizable controls.
• Chapter 10, “Data Binding,” describes how you can connect data to your application user interface controls.
• Chapter 11, “Effects,” describes how to use built-in effects and create your own custom effects. Effects are a new feature in Silverlight 3.
• Chapter 12, “GPU Acceleration,” explains how to use your graphics process unit (GPU) to improve the performance of animation and video playback. GPU acceleration is a new feature in Silverlight 3.
In this chapter, you will learn how to add rich vector graphics and images to your application. You will also learn how to optimize performance and image quality of those graphics elements. In particular, Chapter 3 will discuss the following:

- The graphics system design principles
- The elements for displaying graphics
- The problems the Silverlight runtime solves “under the hood” and the problems your application must solve

Graphics Principles

The Silverlight graphics API makes it easy to add vector graphics, bitmap images, and text to your applications. This section describes the graphics API design principles.

Vector Graphics and Bitmap Images

Bitmap images are a common method of adding graphics to an application. However, bitmap images become blurry when scaled up, as shown in Figure 3.1, or aliased when scaled down, as shown in Figure 3.2. Unlike a bitmap image, if you scale a vector graphic, it will remain sharp as shown in Figure 3.3. Both vector graphics and bitmap images are useful in most
Figure 3.1: Scaling up an image of a circle

Figure 3.2: Scaling down an image of a circle

Figure 3.3: Scaling a vector graphic circle
applications. For example, a user interface control looks better at different sizes and resolutions with vector graphics instead of bitmap images. Bitmap images are useful for displaying content that is not easily expressible in vector form such as digital photographs or visual effects not supported by the runtime.

**New in Silverlight 3**

There are a number of techniques for scaling down an image to produce a result better than that shown in Figure 3.2. Silverlight 3 converts your bitmap to a set of smaller images at various sizes using a better algorithm. This conversion happens once per bitmap and Silverlight caches the result of the conversion for the lifetime of your bitmap. During an animation, Silverlight 3 dynamically selects the right resolution bitmap to display. This process can increase memory usage by 33%, but substantially improves the display quality of scaled down images and 3D transforms to get a result close to that shown in Figure 3.3.

When scaling up bitmaps in Silverlight 3, the image remains blurry as shown in Figure 3.1.

**Retained Mode**

There are two types of graphics API: retained mode and immediate mode. A retained mode API automatically responds to changes to a graph of objects. An immediate mode API requires you to issue all the draw commands to describe a change. For example, to remove the rectangle shown in Figure 3.4 in a retained mode system, simply call a function to remove that element. The retained mode graphics system is responsible for redrawing the background, the triangle beneath, and the ellipse above. To remove the same rectangle shown in Figure 3.4 with an immediate mode API, you need to make three calls to draw the background, the triangle beneath it, and the ellipse above as shown in Figure 3.5.
Figure 3.4: Removing a rectangle with a retained mode API

Figure 3.5: Removing a rectangle with an immediate mode API
A retained mode API enables you to do the following:

- Construct a set of graphics elements
- Change properties of the graphics elements
- Build a graph representing the relationship between those elements
- Manipulate the graph structure

A retained mode graphics API is easier to use than an immediate mode API and enables the underlying system to provide automatic performance optimizations such as drawing incrementally and avoiding the drawing of occluded shapes. Silverlight provides a retained mode system to optimize for ease of use, animating vector graphics content, and for building applications composed of UI controls.

In Silverlight, you can construct the retained mode graph declaratively with a XAML file:

```xml
<Canvas xmlns="http://schemas.microsoft.com/client/2007">
  <!-- triangle -->
  <Path Fill="Green"
        Data="F1 M 128,12 L 12,224 L 224,224"/>
  <!-- rectangle -->
  <Rectangle Fill="Blue"
              Canvas.Left="96"
              Canvas.Top="160"
              Width="256"
              Height="224"/>
  <!-- circle -->
  <Ellipse Fill="Red"
          Canvas.Left="230"
          Canvas.Top="288"
          Width="200"
          Height="200"/>
</Canvas>
```
Alternatively, you can construct the retained mode graph with code:

```csharp
Canvas canvas = new Canvas();

//
// Make the triangle
//

Path path = new Path();
path.Fill = new SolidColorBrush(Colors.Green);
PathFigure pathFigure = new PathFigure();
pathFigure.StartPoint = new Point(128, 12);
LineSegment lineSegment1 = new LineSegment();
lineSegment1.Point = new Point(12, 224);
pathFigure.Segments.Add(lineSegment1);
LineSegment lineSegment2 = new LineSegment();
lineSegment2.Point = new Point(224, 224);
pathFigure.Segments.Add(lineSegment2);
PathGeometry pathGeometry = new PathGeometry();
pathGeometry.Figures = new PathFigureCollection();
pathGeometry.Figures.Add(pathFigure);
path.Data = pathGeometry;

canvas.Children.Add(path);

//
// Make the rectangle
//

Rectangle rectangle = new Rectangle();
rectangle.Fill = new SolidColorBrush(Colors.Blue);
Canvas.SetLeft(rectangle, 96);
Canvas.SetTop(rectangle, 160);
rectangle.Width = 256;
rectangle.Height = 224;

canvas.Children.Add(rectangle);

//
// Make the circle
//
Cross Platform Consistency
An important goal for the Silverlight graphics engine is to enable a
developer to write his or her application once and have it run consistently
across a variety of operating systems and browsers. Each operating system
has a local graphics library. However, these local operating system graphics
libraries differ significantly in feature set, performance, and image quality.
To ensure cross-platform consistency and performance, Silverlight includes
its own rendering engine.

Tools
Silverlight is capable of loading vector and image content from designer tools
and integrating with developer written application code. For vector graphics
and animation, you can use Expression Design and Expression Blend to
generate XAML content for use with the Silverlight runtime. There are also a
variety of free XAML exporters available to generate XAML content including
an Adobe Illustrator exporter, an XPS print driver, and several others.

Balancing Image Quality and Speed
In addition to displaying static XAML, Silverlight provides real-time
animation at 60 frames per second. However, real-time performance is
highly dependent on the application content, the specific hardware
configuration of the target machine, the resolution of the target display, the
operating system, and the hosting Web browser.

When an application does not meet the 60 frame per second goal, the
Silverlight team uses the following two options to improve performance:

1. Make optimizations to components in the Silverlight runtime
2. Lower image quality to achieve better speed
Reducing image quality for speed is the most controversial optimization technique. The Silverlight application must look good. However, it is possible to trade off image quality in a manner that is generally acceptable to the end user. For example, vector graphics rendering makes some quality sacrifices but still maintains an acceptable visual bar. On the other hand, end users have a much higher standard for text quality and the Silverlight runtime spends more CPU time rendering text clearly.

**Graphics Elements**

As previously discussed, the Silverlight runtime can load and draw vector graphics XAML on a variety of platforms. The graphics elements can also be specified programmatically when you use C# or JavaScript. This section describes the elements you can use to display vector graphics and images in your applications.

**Shapes**

This section starts the graphics element discussion with the base class for all graphics primitives: the Shape element. The Shape element provides the following:

- A Fill property to specify how the shape interior is drawn
- A Stretch property to indicate how a Shape element stretches to a specified Width and Height
- Stroke and StrokeThickness properties to specify the pen behavior

The elements that derive from Shape define the shape’s geometry. These elements include Rectangle, Ellipse, Line, Polygon, Polyline, and Path.

**Rectangle**

To draw a rectangle at a specified position, place it in a Canvas element and specify its Canvas.Top, Canvas.Left, Width, Height, and Fill color:
You can add an outline to the rectangle as shown in Figure 3.6 by setting the Stroke property to specify the color and the StrokeThickness property to specify the thickness:

```xml
<Canvas xmlns="http://schemas.microsoft.com/client/2007">
  <Rectangle
    Fill="LightGray"
    Stroke="Black"
    StrokeThickness="10"
    Canvas.Left="50"
    Canvas.Top="50"
    Width="400"
    Height="400"
  />
</Canvas>
```

**Figure 3.6: Rectangle element with an outline**
You can use the Rectangle element to draw the rounded rectangles shown in Figure 3.7 by specifying the RadiusX and RadiusY properties:

![Rectangle element with rounded corners](image)

**Figure 3.7: Rectangle element with rounded corners**

```xml
<Canvas xmlns="http://schemas.microsoft.com/client/2007">
  <Rectangle
    Fill="LightGray"
    Stroke="Black"
    StrokeThickness="10"
    RadiusX="40"
    RadiusY="60"
    Canvas.Left="50"
    Canvas.Top="50"
    Width="400"
    Height="400"
  />
</Canvas>
```

**Ellipse**

As with the Rectangle element, you can position an Ellipse element with the same Canvas.Top, Canvas.Left, Width, and Height properties. Silverlight stretches the ellipse to fit the specified bounds as shown in Figure 3.8.
Line
To draw a line, you can use the Line element and set its $X1, Y1, X2, Y2$ properties. As with all other shapes, you can use the Stroke property to specify the fill color and the StrokeThickness property to specify the thickness as shown in Figure 3.9.
If you look closely at the pixels for the horizontal line shown in Figure 3.10, you will see it has a height of two pixels despite the one pixel StrokeThickness
specified in the XAML. Furthermore, the line is gray instead of the specified black color. To understand this rendered result, consider the following equivalent Rectangle element:

```xml
<Canvas xmlns="http://schemas.microsoft.com/client/2007">

<!-- one pixel horizontal line drawn as a rectangle -->
<Rectangle
    Fill="Black"
    Canvas.Left="99.5"
    Canvas.Top="59.5"
    Width="301"
    Height="1"
/>

</Canvas>
```

The previous Rectangle element has half integer values for its position. The reason for the half pixel coordinates is that the line expands by half StrokeThickness in either direction. Because StrokeThickness is equal to one, the line adjusts the top and left coordinates by -0.5. Because the rectangle is between two pixels, it gets anti-aliased and occupies two pixels with a lighter color. If you want sharp horizontal and vertical lines, you should draw a rectangle positioned at integer coordinates to get the result shown in Figure 3.11.
Figure 3.11: Sharp horizontal line drawn with a Rectangle element

Path
The Path element extends the Shape element by providing a Data property that specifies the geometry object. The Rectangle, Ellipse, and Line elements previously discussed are all expressible with the more general Path element. For example, instead of specifying a Rectangle element, we can specify a Path element with a RectangleGeometry:
The Path element syntax is more verbose than the specialized shapes syntax. However, because Silverlight converts all shapes internally to Path elements, if you understand how the Path element draws you will understand how all shapes draw.

In addition to expressing specialized shapes, the Path element can express a geometry consisting of a collection of Figure elements. A Figure element is a connected set of line segments and Bezier segments. The most common method to specify these figures, line segments, and curves is the path mini-language. For example, to draw the shape in Figure 3.12 you would do the following:

```
<Canvas>

  <Path Stroke="Black" StrokeThickness="10">
    <Path.Data>
      <RectangleGeometry Rect="96,160,256,224"/>
    </Path.Data>
  </Path>

</Canvas>
```

**Figure 3.12: Example path**
Chapter 3: Graphics

The commands supported by the mini-language include those shown in Table 3.1. Each command is followed by the action it takes.

An alternative form of specifying the geometry is to use the expanded XAML syntax:

<Canvas xmlns="http://schemas.microsoft.com/client/2007">
  <Path StrokeThickness="10" Stroke="Black" Fill="Red">
    <Path.Data>
      <PathGeometry>
        <PathGeometry.Figures>
          <PathFigure StartPoint="14, 16" IsClosed="true">
            <PathFigure.Segments>
              <BezierSegment Point1="14,16" Point2="-8,256" Point3="64,352" />
              <BezierSegment Point1="136,448" Point2="185,440" Point3="247,336" />
              <BezierSegment Point1="309,233" Point2="448,416" Point3="448,416" />
            </PathFigure.Segments>
          </PathFigure>
        </PathGeometry.Figures>
      </PathGeometry>
    </Path.Data>
  </Path>
</Canvas>
Table 3.1: Path Mini-language Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>M x, y</td>
<td>Move to position x, y</td>
</tr>
<tr>
<td>L x, y</td>
<td>Draw a line from the current position to position x, y</td>
</tr>
<tr>
<td>C x1, y1, x2, y2, x3, y3</td>
<td>Draw a cubic Bezier segment with control points consisting of the current position, (x1, y1), (x2, y2), and (x3, y3)</td>
</tr>
<tr>
<td>Q x1, y1, x2, y2</td>
<td>Draw a quadratic Bezier segment with control points consisting of the current position, (x1, y1), and (x2, y2)</td>
</tr>
<tr>
<td>H x</td>
<td>Draw a horizontal line from the current position x0, y0 to position x, y0</td>
</tr>
<tr>
<td>V y</td>
<td>Draw a vertical line from the current position x0, y0 to position x0, y</td>
</tr>
<tr>
<td>Z</td>
<td>Close a figure</td>
</tr>
<tr>
<td>F0</td>
<td>Specify EvenOdd fill rule</td>
</tr>
<tr>
<td>F1</td>
<td>Specify NonZero fill rule</td>
</tr>
</tbody>
</table>

One additional concept previously discussed is that of a Figure element. Because the Path element can have more than one figure, it can create an empty space in the middle of a filled space as shown in Figure 3.13.
Figure 3.13: Path with an empty space in the center

<Canvas xmlns="http://schemas.microsoft.com/client/2007">
  <Path
    StrokeThickness="10"
    Stroke="Black"
    Fill="LightGray"
    Data="M 50,50 L 50,450 450,450 450,50 50,50 z
         M 100,100 L 100,400 400,400 400,100 100,100 z"
  />
</Canvas>

**PERFORMANCE TIP**

These two forms of specifying a path may render identically, but they differ in performance and flexibility. The mini-language form parses faster, consumes less memory, and draws fastest at runtime. The mini-language is the recommended form for static content. However, it is not possible to bind an animation or change a segment property of a path generated with the mini-language because Silverlight does not create the path, figure, or segment API objects.

**Fill Rules**

The previous section explained how to use geometry to specify an outline of an area to fill. However, an outline does not uniquely specify the inside
and outside of the shape. For example, the outline in Figure 3.13 could generate any of the rendering results shown in Figure 3.14. A fill rule is a method to distinguish the inside of a shape from the outside of the shape.

One approach to specifying what is inside a shape is to cast a horizontal line through the shape and count the number of edges crossed from left to right. If the number of edges crossed is even, classify the horizontal line segment as outside the shape. If the number of edges is odd, classify that segment as inside the shape. This fill rule is the \texttt{EvenOdd} rule and is the default fill mode for \texttt{Path} elements. To specify the fill mode explicitly, you can specify the \texttt{FillRule} property on geometry or use \texttt{F0} for \texttt{EvenOdd} from the path mini-language:

```xml
<Canvas xmlns="http://schemas.microsoft.com/client/2007">

<!-- Path with fill rule F0 = EvenOdd -->

<Path
    StrokeThickness="10"
    Stroke="Black"
    Fill="LightGray"
    Data="F0
        M 50,50 L 50,450 450,450 450,50 50,50 z
        M 100,100 L 100,400 400,400 400,100 100,100 z"
    />

</Canvas>
```

\textbf{Figure 3.14: Different fills for the same outline}
An alternative rule is the NonZero rule, which considers the order points are specified in the input. If an edge goes up in the \( y \) direction, assign that edge a \( +1 \) winding number. If an edge goes down in the \( y \) direction, assign that edge a \( -1 \) winding number. The NonZero rule defines the interior of the shape to be all those segments where the sum of winding numbers from the left to the current segment is not zero. For example, if you specify the geometry shown in Figure 3.14 with the point order in the following markup, it would result in the winding numbers and filled segments shown in Figure 3.15.

![Figure 3.15: Winding mode numbers resulting in a filled center space](image)

If you specify the points in the following order, the shape would render differently as shown in Figure 3.16.
Figure 3.16: Different fill as a result of a different point order

<Canvas xmlns="http://schemas.microsoft.com/client/2007">

<!-- Path with fill rule F1 = NonZero -->

<Path
  StrokeThickness="10"
  Stroke="Black"
  Fill="LightGray"
  Data="F1
    M 50,50 L 50,450 450,450 450,50 50,50 z
    M 100,100 L 400,100 400,400 100,400 100,100 z"

/>

</Canvas>

**PERFORMANCE TIP**

The NonZero rule is more complicated than the EvenOdd rule and does render slower. For most vector graphics fills, the EvenOdd rule gives the desired result.

**Images**

In addition to the vector graphics elements previously discussed, the other fundamental graphics primitive is the Image element. To display an image,
you can use the Image element with a reference to a URI to produce the result shown in Figure 3.17.

```xml
<Canvas xmlns="http://schemas.microsoft.com/client/2007">
  <Image Source="silverlight.png"/>
</Canvas>
```

Figure 3.17: Image element example

The Source property can reference any image in JPG or PNG format. However, if the JPG or PNG contains DPI (dots per inch) information, Silverlight ignores this information because it is usually not accurate for display purposes. All references to URIs in XAML are relative to the location of the XAML file. For example, if the XAML file is in a XAP, Silverlight searches for silverlight.png in the XAP. If the XAML file is a resource in a managed assembly, Silverlight searches for silverlight.png in that same assembly.

If you do not specify the Width and Height properties of an image, Silverlight draws the image at its natural size, which results in a pixel perfect rendering of the original image data.

**Brushes**

All of the previous examples filled the Path element pixels with a single color. Silverlight also supports filling arbitrary shapes with image brushes, linear gradient brushes, and radial gradient brushes. A brush is a mathematical function that maps an \((x,y)\) position to a color. For example, SolidColorBrush is a simple function that ignores its position and always outputs the same color. This section describes the brushes available in Silverlight and includes the function used to map from screen position to color.
**Solid Color Brush**

SolidColorBrush returns a single color for all screen positions. When you specify a Fill or Stroke property value, Silverlight creates a solid color brush for the corresponding shape fill or pen stroke respectively:

```xml
<Canvas xmlns="http://schemas.microsoft.com/client/2007">
  <Rectangle
    Fill="Blue"
    Stroke="Black"
    StrokeThickness="10"
    Canvas.Left="96"
    Canvas.Top="160"
    Width="256"
    Height="224"
  />
</Canvas>
```

Alternatively, you can specify a brush with the expanded XAML syntax:

```xml
<Canvas xmlns="http://schemas.microsoft.com/client/2007">
  <Rectangle
    StrokeThickness="10"
    Canvas.Left="96"
    Canvas.Top="160"
    Width="256"
    Height="224"
  >
    <Rectangle.Fill>
      <SolidColorBrush Color="Blue"/>
    </Rectangle.Fill>
  </Rectangle>
</Canvas>
```

**Technical Insight**

As shown in the previous example, the Source property of an Image element can be set to a URI. Many references to the same Source URI cause Silverlight to download the image once and use it multiple times. If you remove all references to Image elements for a specific URI, Silverlight removes the image from the memory cache and a future reference causes Silverlight to initiate another download. Future downloads may be serviced from the browser cache or go over the network if the image is no longer in the browser cache.
The previous examples specified a named color. You can also specify a color explicitly by providing a hex string of the form #aarrggbb, which represents a hex alpha channel value, red channel value, green channel value, and blue channel value. For example, opaque green would be #ff00ff00.

From C#, you can specify a color by creating an instance of the Color class:

```csharp
Color green = Color.FromArgb(0xff, 0x0, 0xff, 0x0);
```

The alpha channel specifies the degree of transparency where 0x0 indicates completely transparent, 0xff indicates an opaque color, and intermediate values indicate partial transparency. A brush with a transparent color will blend its color to the background color using the following formula:

\[
\text{Color}_{\text{destination}} = (\alpha \times \text{Color}_{\text{source}} + (0xff - \alpha) \times \text{Color}_{\text{destination}})/256
\]

Silverlight will pass colors specified in hex format to the Web browser without a color space conversion. Typically, the browser will interpret the color as a standard RGB color space color (sRGB) that is, an 8-bit per channel 2.2 implied gamma space color. However, the visible color may vary with operating systems, Web browser, the monitor, and the operating system color profile. An alternative form of specifying colors is the floating point wide gamut linear RGB color space (scRGB):

```xml
<Canvas xmlns="http://schemas.microsoft.com/client/2007">
  <Rectangle
    Fill="sc#1, 1.0, 0.0, 0.0"
    Canvas.Left="96"
    Canvas.Top="160"
    Width="256"
    Height="224"
  />
</Canvas>
```

Silverlight converts scRGB colors to sRGB internally for blending. Consequently, specifying colors in native sRGB is desirable to avoid extra color conversion steps.
**Gradient Brushes**

Gradient brushes define a set of colors and positions along a virtual gradient line. The function that maps screen position to color first maps the screen position to a point along the gradient line and then interpolates a color based on the two nearest points.

Consider the following use of a `LinearGradientBrush`:

```xml
<Canvas xmlns="http://schemas.microsoft.com/client/2007">
    <Ellipse Width="450" Height="450">
        <Ellipse.Fill>
            <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
                <LinearGradientBrush.GradientStops>
                    <GradientStop Color="White" Offset="0"/>
                    <GradientStop Color="Black" Offset="1"/>
                </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>
        </Ellipse.Fill>
    </Ellipse>
</Canvas>
```

The preceding linear gradient brush produces the fill shown in Figure 3.18. A linear gradient brush maps a screen position to the point on the line
closest to that position. The brush then interpolates a color based on the two nearest points specified along the line as shown in Figure 3.18.

Alternatively, you can specify a radial gradient fill using RadialGradientBrush that takes the distance from the screen position to the center of the radial gradient and maps that distance to the specified gradient line of colors and positions. For example, the following XAML generates the rendering shown in Figure 3.19.

```xml
<Canvas xmlns="http://schemas.microsoft.com/client/2007"

    <Ellipse
        Width="450"
        Height="450"
    >
        <Ellipse.Fill>
            <RadialGradientBrush>
                <RadialGradientBrush.GradientStops>
                    <GradientStop Color="White" Offset="0"/>
                    <GradientStop Color="Black" Offset="1"/>
                </RadialGradientBrush.GradientStops>
            </RadialGradientBrush>
        </Ellipse.Fill>
    </Ellipse>
</Canvas>
```

**Figure 3.19: Radial gradient brush**
Another feature of RadialGradientBrush is the capability to move the point that maps to the start of our gradient line. In particular, in our previous example, the center of the radial gradient circle mapped to the start of our gradient line and the radius of the gradient circle mapped to the end of our gradient line. With this pattern, you always get a uniform ellipse. You can move the center using the GradientOrigin property to get the result shown in Figure 3.20.

![Focal gradient brush](image)

**Figure 3.20: Focal gradient brush**

```xml
<Canvas xmlns="http://schemas.microsoft.com/client/2007">
  
  <Ellipse>
    Width="450"
    Height="450"
  
  <Ellipse.Fill>
    <RadialGradientBrush GradientOrigin="0.25 0.25">
      <RadialGradientBrush.GradientStops>
        <GradientStop Color="White" Offset="0"/>
        <GradientStop Color="Black" Offset="1"/>
      </RadialGradientBrush.GradientStops>
    </RadialGradientBrush>
  </Ellipse.Fill>

  </Ellipse>

</Canvas>
```
One other feature of linear and radial gradients is the capability to specify the behavior when the display position maps to some position outside the range of the gradient line. The **SpreadMethod** property defines that behavior. The **Pad** mode repeats the closest point when off the line, the **Reflect** mode mirrors to a point on the line, and the **Repeat** mode simply takes the position modulo the length of the line as shown in Figure 3.21.

![SpreadMethod example](Figure 3.21: SpreadMethod example)

**Image Brushes**
The role of the image brush is to map a screen position to a pixel in the specified image. For example, the following XAML would result in the image brush rendering shown in Figure 3.22.

```xml
<Canvas xmlns="http://schemas.microsoft.com/client/2007">
  <Ellipse
```

![ImageBrush example](Figure 3.22: ImageBrush example)
Strokes

The previous section showed how to use a brush to color the fill of a shape. You can also use a brush to add color to the outline of a shape by setting the stroke properties. For example, the following XAML generates the output shown in Figure 3.23.

```xml
<Canvas xmlns="http://schemas.microsoft.com/client/2007">
    <Ellipse Stroke="Black" StrokeThickness="10"
        Canvas.Left="50" Canvas.Top="50" Width="400" Height="400">
        <Ellipse.Fill>
            <ImageBrush ImageSource="silverlight.png"/>
        </Ellipse.Fill>
    </Ellipse>
</Canvas>
```

Figure 3.23: Sample stroke applied to an ellipse
**Stroke**

A stroke transforms geometry to a widened form that describes the shape outline instead of the shape fill. Silverlight fills the widened geometry with exactly the same rendering rules as the main shape fill. For example, Figure 3.24 shows an example of a widened ellipse.

The widening process expands the original geometry by half the stroke thickness to form an outer outline. The widening process also shrinks the original geometry by half the stroke thickness to form an inner outline. The outer and inner outlines combine to form two figures Silverlight fills to produce the resulting stroke.

![Figure 3.24: The widening process applied to an ellipse](image)

**Technical Insight**

One side effect of the widening process is that local self-intersection can occur. For example, the process of widening a triangle generates several self-intersections as shown in Figure 3.25. One option is to run a loop removal algorithm to remove these intersections before rasterization. However, by simply filling the new geometry with the NonZero fill rule, these self intersections are not visible to the end user.
Dashes
To add dashes to your strokes, specify an array of distances alternating between the dash filled distance and the gap distance. For example, the simple dash array in the following XAML generates the output shown in Figure 3.26.
Technical Insight

Dashes are also a geometry modifier built on top of the stroke geometry modifier. When you specify a StrokeDashArray, Silverlight takes the output of the pen and subdivides it into smaller geometries. Large numbers of dashes can result in significant slowdowns in rendering speed and therefore you should use them sparingly.

Canvas

Every example shown so far has had a single root Canvas element with a set of Shape elements contained within it. In addition to providing a convenient container, the Canvas element also enables you to modify the rendering primitives it contains as a group. In particular, the Canvas element enables the following:

- Naming groups of elements
- Grouping shapes so that you can add or remove the group with a single operation
- Applying a transform to the group of elements
- Clipping a group of elements
- Apply an opacity or opacity mask effect to a group of elements

Transforms, clipping, and opacity effects are available on both individual shapes and the Canvas element.
**Perfomance Tip**

For individual shapes, it is faster to express clipping or opacity as a different geometry or a different brush color. For example, draw a Path with an ImageBrush to achieve the same result as applying a clip to an Image element. Similarly, you can add opacity to the brush color alpha channel instead of adding Opacity to the shape.

**Transforms**

A transform enables you to position, rotate, scale, or skew a shape or group of shapes. To transform a group of primitives, you can set the RenderTransform on the Canvas element as exemplified in the following listing to achieve the result shown in Figure 3.27.

**Figure 3.27**: RenderTransform example of overlapping a rectangle over an ellipse
Chapter 3: Graphics

As shown in the previous example, you can use a list of `ScaleTransform`, `TranslateTransform`, and `RotateTransform` elements in a `TransformGroup` element. Alternatively, you can specify an explicit matrix with a `MatrixTransform`:

```xml
<Canvas xmlns="http://schemas.microsoft.com/client/2007">
    <Canvas.RenderTransform>
        <TransformGroup>
            <MatrixTransform Matrix="1.30, 0.75,
                                 -0.50, 0.87,
                                 100.00, -10.00"/>
        </TransformGroup>
    </Canvas.RenderTransform>

    <Ellipse Fill="LightGray" Stroke="Black" StrokeThickness="20"
             Width="200" Height="200"/>

    <Rectangle Fill="Gray" Stroke="Black" StrokeThickness="20"
                Canvas.Left="100" Canvas.Top="100"
                Width="200" Height="200"/>

</Canvas>
```

As shown in the previous example, you can use a list of `ScaleTransform`, `TranslateTransform`, and `RotateTransform` elements in a `TransformGroup` element. Alternatively, you can specify an explicit matrix with a `MatrixTransform`:
3D Transforms (New in Silverlight 3)

In Silverlight 3, you can set the Projection property to a PlaneProjection to rotate a group of elements in 3D as shown in Figure 3.28.

![3D projection example](image)

**Figure 3.28: 3D projection example**
Each projection logically has its own camera. To position more than one object relative to the same perspective camera, position them all in the same place and use the GlobalOffsetX, GlobalOffsetY, and GlobalOffsetZ properties to move in the 3D world as shown in Figure 3.29.

![Figure 3.29: Position three rectangles in the same 3D projection camera](image)
The global offset properties apply after the rotation property. You can also use the LocalOffsetX, LocalOffsetY, and LocalOffsetZ properties on the PlaneProjection object to apply an offset before the rotation.

**Clipping**

Clipping is the process of restricting the display area to a specified shape. To clip an element, set the Clip property as shown Figure 3.30 and in the following listing:
Chapter 3: Graphics

PERFORMANCE TIP

A clipping operation is semantically equivalent to intersecting two geometries. Clipping a group of elements or a single shape does come with a significant performance penalty. You should avoid clipping when possible.
**Opacity**

Setting opacity on a brush or setting a transparent color on a brush introduces alpha blending. In particular, if a brush contains a transparent color, the brush blends its color with the content underneath using the following formula:

$$\text{Color}_{\text{destination}} = \alpha \times \text{Color}_{\text{source}} + (1 - \alpha) \times \text{Color}_{\text{destination}}$$

The other form of opacity is setting the `Opacity` property on a Canvas. This operation is not equivalent to changing the opacity of each of the shapes within the Canvas element as demonstrated by Figure 3.31.

**PERFORMANCE TIP**

Setting `Opacity` on a Canvas element resolves occlusion first and then blends content. This process is significantly slower at runtime than blending individual primitives. If possible, you should set opacity on a brush, brush color, or a `Path` element for maximum performance.

**OpacityMask**

The `OpacityMask` property on a `UIElement` provides a mechanism to blend brush per pixel alpha information with the content of a `UIElement`. For example, the following XAML would produce the result shown in Figure 3.32.
Chapter 3: Graphics

Figure 3.32: OpacityMask example

<Canvas xmlns="http://schemas.microsoft.com/client/2007">
  <Canvas.OpacityMask>
    <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
      <LinearGradientBrush.GradientStops>
        <GradientStop Color="Transparent" Offset="0"/>
        <GradientStop Color="White" Offset="1"/>
      </LinearGradientBrush.GradientStops>
    </LinearGradientBrush.OpacityMask>
  </Canvas.OpacityMask>

  <Ellipse
    Fill="LightGray"
    Stroke="Black"
    StrokeThickness="20"
    Width="200"
    Height="200"
  />

  <Rectangle
    Fill="Gray"
    Stroke="Black"
    StrokeThickness="20"
    Canvas.Left="100"
    Canvas.Top="100"
    Width="200"
    Height="200"
  />

</Canvas>
OpacityMask is slow at runtime. In some cases, it is faster to draw content on top that blends to the background instead of using the OpacityMask. For example, you can achieve the effect in Figure 3.32 with the following XAML:

```xml
<Canvas xmlns="http://schemas.microsoft.com/client/2007">
  <Ellipse
    Fill="LightGray"
    Stroke="Black"
    StrokeThickness="20"
    Width="200"
    Height="200"
  />
  <Rectangle
    Fill="Gray"
    Stroke="Black"
    StrokeThickness="20"
    Canvas.Left="100"
    Canvas.Top="100"
    Width="200"
    Height="200"
  />

  <!–– simulate opacity mask effect with a rectangle on top ––>
  <Rectangle Width="300" Height="300">
    <Rectangle.Fill>
      <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
        <LinearGradientBrush.GradientStops>
          <GradientStop Color="White" Offset="0"/>
          <GradientStop Color="Transparent" Offset="1"/>
        </LinearGradientBrush.GradientStops>
      </LinearGradientBrush>
    </Rectangle.Fill>
  </Rectangle>
</Canvas>
```

**Under the Hood**

Previous sections have discussed the graphics principles and the graphics API elements. This section goes “under the hood” to describe how Silverlight draws XAML content and displays it in the browser window.
Understanding this process will help you understand the Silverlight runtime performance characteristics. Furthermore, you will understand the problems solved by the runtime and the problems your application must solve.

In particular, this section discusses the following:

- The draw loop process that takes changes to the graph of objects and draws it to an off screen back buffer
- The rasterization process that converts vector graphics primitives to pixels in an offscreen back buffer
- Performance optimizations such as incremental redraw, occlusion culling, and multi-core
- How the offscreen back buffer gets displayed in the browser window

**Draw Loop**

Silverlight draws at a regular timer interval set by the `MaxFrameRate` property. On each tick of the timer, Silverlight does the following:

1. Checks for any changes to the properties of our graph of `Canvas` and `Shape` elements. If no changes exist, Silverlight does no further work for this timer tick.
2. Performs any pending layout operations. Chapter 7, “Layout,” will discuss these layout operations further.
3. Gathers rendering changes and prepares to rasterize them.
4. Incrementally rasterizes the changes for the current timer tick. The graphics state at the current timer tick is the current frame.
5. Notifies the browser that a frame (or an incremental delta to an existing frame) is complete for display.

**Rasterization**

After the draw loop has identified which elements need to be redrawn, Silverlight converts those elements to a set of pixels in our offscreen
The previous discussion of shapes described how to specify path outlines and a method of specifying the inside and the outside of the shape. However, the geometry describes an abstract infinite resolution outline of a shape and a screen has a finite number of pixels to color. Rasterization is the process of converting from a path outline to discrete pixels. This section describes how rasterization is accomplished.

The simplest method to convert geometry to pixels is a process called sampling. The sampling process uses a discrete number of sample points to convert from the infinite shape description to pixels. For example, consider the simple sample pattern consisting of a uniform grid of sample points with one sample point per pixel. If the sample point is contained within the geometry, light up the pixel. If the sample point is not contained within the geometry, do not light the pixel. For example, the circle specified by the following XAML would light the pixels shown in Figure 3.33.
You may have noticed that the integer coordinates were located at the top left of the pixel and the sample points were in the center of a pixel. This convention enables a symmetrical curved surface specified on integer coordinates to produce a symmetrical rasterization. If the sample points were on integer coordinates instead, the ellipse would lose symmetry as shown in Figure 3.34.

The rasterization shown in Figure 3.33 appears to have jagged edges. This jagged appearance is the consequence of aliasing. Aliasing is the loss of information that results from converting from a continuous curve to a discrete set of samples. Anti-aliasing is a term that refers to a technique that attempts to minimize aliasing artifacts.

The Silverlight anti-aliasing technique consists of sampling multiple times per pixel and applying a box filter to produce the final pixel color. Silverlight conceptually samples 64 times per pixel as shown in Figure 3.35.
The box filter averages the contribution of all samples within a rectangle bordering the pixel to produce a final pixel color. If some partial number of samples is in the box, Silverlight applies transparency to blend smoothly with what is underneath the geometry as shown in Figure 3.36. This anti-aliasing technique produces a smooth transition from inside the shape to outside the shape along edges.
**Technical Insight**

You may be wondering why there are 16 samples per pixel in the x direction and only 4 samples per pixel in the y direction. The reason for picking this sample pattern is that horizontal resolution is critical to being able to render text clearly. Furthermore, horizontal resolution is computationally cheap and vertical resolution is slower. The 16x4 sampling pattern balances image quality and speed.

Instead of a box pattern, it is also possible to accumulate samples in a circular pattern, weight samples unevenly, or even have a sample pattern that extends far beyond a single pixel in size. In fact, all of these other algorithms generate better image quality than a box filter but typically render more slowly. The Silverlight high-resolution box filter is a choice made to achieve good rendering performance with reasonable image quality.
One artifact of anti-aliasing is a visible seam that sometimes results from drawing two adjacent shapes. For example, the following two rectangles that meet in the middle of a pixel generate a seam:

```xml
<Canvas xmlns="http://schemas.microsoft.com/client/2007">
  <Rectangle
    Fill="Black"
    Width="100.5"
    Height="100.5"
  />
  <Rectangle
    Fill="Black"
    Canvas.Left="100.5"
    Width="100.5"
    Height="100.5"
  />
</Canvas>
```

The previous XAML results in the rasterization shown in Figure 3.37. Notice the gap between the two rectangles. The rectangles joined perfectly in the input XAML, so why is there a seam in the rendered result?

These seams are a result of the rasterization rules described in this section. Consider the rasterization process applied to a light gray pixel \( X \) shown in Figure 3.37. Rectangle \( A \) is covering exactly half the samples for pixel \( X \). Silverlight consequently draws that pixel of Rectangle \( A \) with 0.5 anti-aliasing alpha. Alpha is a term that refers to the transparency used to blend colors with a formula such as

![Figure 3.37: Anti-aliasing seam example](image)
Color_destination = alpha * Color_source + (1 – alpha) * Color_destination

In our example, \( \alpha = 0.5 \), \( Color\_source = Black \), and \( Color\_destination = White \). Blending along the edge of rectangle \( A \) results in a destination color of

\[
0.5 \times \text{Black} + (1 - 0.5) \times \text{White} = 0.5 \times \text{White}
\]

Rectangle \( B \) also covers half its sample points. Silverlight also blends pixel \( X \) of rectangle \( B \) with \( \alpha = 0.5 \) to a background color of \( 0.5 \times \text{White} \). Consequently, the resulting color is

\[
0.5 \times \text{Black} + (1 - 0.5) \times (0.5 \times \text{White}) = 0.25 \times \text{White}.
\]

The final pixel color has one quarter of the background color showing through as a visible seam.

**Technical Insight**

This result is an artifact of sampling each primitive independently. An alternative anti-aliasing mode is full screen anti-aliasing that processes all samples from all shapes simultaneously. However, Silverlight does not currently use full screen anti-aliasing because it results in slower runtime performance.

To avoid these seams, you should snap edges to pixel boundaries as shown in Figure 3.38. Snapping also produces a sharper edge between the

![Figure 3.38: Pixel snapped rasterization](image)
two shapes. However, pixel snapping only removes seams if you align the shapes edges with the \( x \)-axis or the \( y \)-axis. For the rotated edges shown in Figure 3.39, snapping does not remove the artifact. For rotated edges, the common technique to avoid this seam is to overlap the edges so that the background is no longer visible.

![Figure 3.39: Seams with a rotated edge](image)

**Bilinear Filtering**

The previous section discussed how Silverlight converts an arbitrary geometry to a set of pixels to fill. Silverlight then colors the filled pixels based on the brush specified. This process is straightforward for solid color brushes and gradient brushes. However, with image brushes, Silverlight must map from the destination pixels to the original image data, which may be at a different resolution. This section describes the mapping function used to achieve the image data stretch shown in Figure 3.40.

Nearest neighbor is a simple image scaling function that transforms the destination pixel to an image bitmap position and picks the nearest pixel color. Nearest neighbor sampling generates ugly aliasing artifacts when the image is displayed with a scale or rotation as shown in Figure 3.41. You will notice jagged lines if you look at the picture closely.

Silverlight uses nearest neighbor sampling in the special case where the brush image data maps exactly onto centers of pixels. For rotated, scaled, or non-integer translated images, bilinear filtering is used to produce the result shown in Figure 3.40.
Bilinear filtering maps the screen position to a position \((u,v)\) in image space. The bilinear filtering process then interpolates a color from pixels \((\text{floor}(u), \text{floor}(v))\), \((\text{floor}(u) + 1, \text{floor}(v))\), \((\text{floor}(u), \text{floor}(v) + 1)\), and \((\text{floor}(u) + 1, \text{floor}(v)+1)\). Figure 3.42 illustrates this process. Bilinear filtering generates good results for scales that are within a factor of two of the original image size. Figure 3.43 demonstrates the results of scaling an image in two sizes within reasonable limits.
With bilinear filtering, if you scale up an image significantly, it becomes blurry. Conversely, if you scale down an image significantly, it looks aliased. Figure 3.44 shows examples of both these artifacts.
New in Silverlight 3

There are a number of techniques for scaling down an image to produce a result better than Figure 3.44. However, these techniques can slow down your animations. Silverlight 3 adds support for mip-mapping that converts your image to a set of smaller images at various sizes using a better algorithm.

For example, if you have a 128x128 image, Silverlight also generates copies at 64x64, 32x32, 16x16, 8x8, 4x4, 2x2, and 1x1 resolutions resized with high quality. When displaying the image at a particular scale, Silverlight chooses the closest resolution to the display size or even uses multiple sizes at once when displaying in 3D.

This conversion happens only once per image and Silverlight caches the resulting images. Consequently, this approach uses 33% more memory to store images to achieve better image quality.

Incremental Redraw

In addition to drawing static objects for a single frame, Silverlight must constantly redraw objects as they are changing. If an object moves from one
position to another, it would be wasteful to redraw all the pixels on the screen. Instead, Silverlight marks the old position as needing a redraw and marks the new position as also needing a redraw. To simplify this marking algorithm, Silverlight uses the bounding box of a shape instead of the tight shape bounds.

For example, suppose the shape shown in the following XAML moves from position 0,0 to position 100,100. Figure 3.45 shows the area that is redrawn.

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007">
  <Ellipse Fill="Black" Width="100" Height="100"/>
</Canvas>
```

To view a visualization of these incremental redraw regions in an application, use the following JavaScript:

```javascript
function loadHandler()
{
  plugin = document.getElementById("MySilverlightPlugin");
  plugin.settings.EnableRedrawRegions = true;
}
```
This visualization blends a transparent color on top of any content drawn and cycles to a different color each frame. Consequently, any content that is flashing represents content that Silverlight is constantly redrawing. Any content that stabilizes on a single color has not changed for several frames.

**Occlusion Culling**
The most expensive operation in the draw loop is the rasterization process, which writes each of the destination pixels. For example, a full screen animation can consist of processing several hundred million pixels per second. Each of these pixels applies at least one brush operation. If there are overlapping brushes, the computational requirements can multiply by a factor of 3 to 10.

As the graph of elements gets more complicated, it may no longer render at the desired frame rate. To optimize the rasterization process, Silverlight avoids brush operations for completely occluded brush pixels. For example, if you draw a full screen background and an almost full screen image, Silverlight computes all the image pixels and only those background pixels not covered by the image itself. For complicated graphs of elements, this optimization can produce a 3–10x speedup.

---

**PERFORMANCE TIP**
Occlusion culling only applies to brush pixel color optimizations. If a complicated geometry is behind a big opaque rectangle, the rasterizer walks the geometry before it realizes that the pixel operations are not necessary. Consequently, it is still important to remove hidden content from the element tree for maximum performance.

---

**Multi-Core Rendering**
Silverlight takes advantage of multiple CPU cores to produce faster rendering throughput. In particular, Silverlight subdivides a frame into a set of horizontal bands and distributes the rasterization of those bands across CPU cores as shown in Figure 3.46. Currently, only the frame rasterization step, effects, mip-map generation, and media operations run in parallel.
across CPU cores. Systems such as layout, control templating, application user code, and animation all run on a single thread. Consequently, you can determine if your application is rasterization bound by simply setting your framerate to 10000 frames per second and measuring your CPU usage percentage. If you achieve almost 100% CPU usage on a dual core machine, you are almost entirely rasterization bound. If you achieve 70% CPU usage on a dual core machine (at 10000 frames per second), that means 30% of the work is not running in parallel.

How Content Gets to the Screen
As previously discussed, the draw loop first draws a frame to an offscreen back buffer and then it notifies the browser that the frame is ready for display. With windowless=false mode, Silverlight content goes to the screen without browser intervention on most operating systems. With windowless=true, the browser copies the offscreen frame to its display area. This extra step is both slow and can result in visual tearing effects in a number of browsers. The worst mode of operation is when windowless=true is specified with a transparent color for the background of the control. The transparent color causes the Web browser to redraw the content underneath the control each time any control content has changed.
Where Are We?

This chapter described the following:

- The graphics system design principles
- The elements for displaying graphics
- The problems the Silverlight runtime solves “under the hood” and the problems your application must solve

In addition, you have learned a number of important performance optimization techniques for use with your application.
Index

**Numbers**
3D transforms
  - applying to content, 200
  - overview of, 75–77
Silverlight 3 features, 7

**A**
AAC (Advanced Audio Coding)
  - media delivery and, 179
  - media source and, 182
  - Silverlight 3 audio formats, 9
Accessibility, input events and, 127
ActualHeight and ActualWidth properties,
  - TextBlock element, 101–102
Adaptive bit-rate streaming, 177, 179
Adobe Illustrator, exporter for
  - XAML files, 47
Advanced Streaming Redirector (ASX)
  - media delivery and, 179
  - as media source, 182–183
Alignment
  - Grid element properties, 166
  - TextAlignment property, 103
Alpha, 239–240
Animation
  - applications, 37–38
  - changing property values over time,
    134–135
  - customizing, 135–136
  - easing functions, 145–147
  - elements, 136–138
  - frames per second, 47
  - interpolation and keyframe animations,
    140–144
  - overview of, 131
  - performance factors in, 136
  - principles, 132
  - review, 152
  - screen display and, 151
  - setting property change speed, 149–151
Silverlight 3 features, 8
  - starting, 138–139
  - steps in, 148
  - of text, 107
  - time-based vs. frame-based, 132–134
  - timelines and, 145
  - timer setup, 148
Anti-aliasing
  - gamma correction for anti-aliased text, 113
  - rasterization and, 84–87
Application model
  - designer/developer collaboration and, 13
Silverlight 3 features, 7
Application.Current.IsRunningOutOfBrowser property, 21
ApplicationManifest.xml, 21
Applications
  - animation, 37–38
  - automatic resizing, 156–157
  - CLR and, 36
  - components, 15–20
  - creating in Visual Studio, 14–15
  - cross platform support, 13
  - designer/developer collaboration and, 13
  - downloader for, 34–35
Applications (continued)
element tree, 36
event system, 37
layout system, 38–39
libraries, 12–13
out of browser, 20–21
overview of, 11
plug-in and, 33–34
principles, 11
rendering, 39
runtime architecture, 32–33
security, 14
Web deployment and out of browser operation, 12
XAML custom classes, 27–30
XAML event handlers, 26–27
XAML file for constructing application objects, 22–23
XAML namespace, 23–24
XAML parser, 35
XAML type converters, 24–26
XAP packages and, 30–31
AppManifest.xml file, 15–17
ArrangeOverride method, WrapPanel class, 173
Artifacts of animation, 134
of anti-aliasing, 87–88
ASX (Advanced Streaming Redirector) media delivery and, 179
as media source, 182–183
Asynchronous firing of events, 129
Audio capabilities in Silverlight, 8–9
formats, 9, 179, 182
Automatic application resize, 156–157
Automatic sizing and positioning layout elements for, 159
overview of, 154–155

B
background property, Silverlight plug-in, 33
BeginTime property, managing animations on timeline, 144
Bilinear filtering
GPU and, 252
graphics and, 88–92
Binding modes, 219–221
Binding object
connecting data sources to elements, 215–217
controlling synchronization and, 220
how it works, 229–230
BindingExpression, 229–230
Bit-rate streaming, 177, 179
Bitmap images
compared with vector graphics, 41–42
scaling, 42–43
blur( ) method, focus and, 116
BlurEffect
built-in effects, 234–235
performance and, 250
running, 249
Bold format, text, 104
Border element
automatic sizing and positioning of elements, 154, 155
overview of, 159–160
Broadcast streaming
Silverlight supporting, 177, 179
sources for, 191
Brushes
adding color to text, 105
animating color, 150
converting from Priority type to Brush type, 225–226
dashes added to strokes, 71–72
gradient brushes, 65–68
image brushes, 68–69
opacity and, 79
overview of, 62
solid color brush, 63–64
strokes and, 69–71
BufferingProgress, MediaElement, 187
Built-in controls
customizing, 198, 207–209
list of, 196–197
Built-in effects, 234
Button element
adding button to XAML file, 15
positioning, 155
as subclass of ContentControl element, 204–206
Buttons
adding button to XAML file, 15
creating custom controls, 207–209
pulsing, 131
Index

C

C#
CLR for writing code in, 2
defining event handler for input events, 119
writing application code, 5, 36
CacheMode property, UIElement, 254–259
Caching glyphs and character related information, 112
Canvas element
BlurEffect nested in, 249
creating custom XAML elements, 27–28
limitations of and when to use, 158–159
measure and arrange passes and, 176
opacity and, 79
overview of, 72
positioning elements with, 158
RenderTransform example, 73
ClearType
readability of text and, 113
support in Silverlight 3, 99
Clipping, graphics display area, 77–78
Closed captioning, 178
CLR (Common Language Runtime)
data sources and, 214
options for writing application code, 36
writing code in C# or Visual Basic, 2
writing Silverlight code in, 7
Code
execution, 12–13
writing application code, 5, 36
writing code in C# or Visual Basic, 2
writing code using CLR, 7
Collections, binding with ItemsControl, 221–224
ColorAnimation element, 140–141
Colors
adding to text, 105
animating brush color, 150
solid color brushes, 64
ColumnDefinition properties, Grid element, 164–165
Comboboxes, binding data to, 223–224
Common Language Runtime. See CLR (Common Language Runtime)
Connections, data source, 214
Content delivery, media, 178–179
Content property, ContentControl element, 199–201
ContentControl element
Button element as subclass of, 204–206
creating custom control by inheriting from, 207
data binding and, 203
overview of, 199–201
ContentPresenter element, 200
Control base class, 195
Controls
adding to media players, 183–187
built-in, 196–197
ContentControl, 199–201, 203
custom, 198
custom controls for input events, 117
customizing built-in, 207–209
elements, 198–199
focus of, 127
GoToState method of visual state manager, 210–211
instantiation of, 209–210
ItemsControl, 201–203
keyboard event handlers, 126
overview of, 195–196
principles, 196
review, 211
Silverlight 3 features, 9
in Silverlight toolkit, 197
sliding menu of, 131
styling, 206–207
UI controls in applications, 13
VisualStateManager and, 203–206
CornerRadius property, 160
CPUs
access to system memory, 264
GPU capable of higher fill rates than CPUs, 252
multi-core rendering, 94–95
SIMD (Single Instruction Multiple Data) and, 234
Cross platform
applications, 13
graphics, 47
input events, 117
Silverlight and, 5–6
text, 98
CurrentStateChanged event handler, 186
Cursors, flashing, 131
Custom
animation, 135–136
easing functions, 147
Custom (continued)
fonts, 106
layout, 169–173
XAML classes, 27–30
Custom controls
creating, 198
for input events, 117

D
Dashes, added to strokes, 71–72
Data binding
automatic connection and synchronization
and, 214
binding collections, 221–224
Binding object and, 229–230
connecting data to elements, 215–218
ContentControl element and, 203
data sources and, 214
data synchronization and binding modes, 219–221
data validation and, 227–228
DataContext inheritance, 218
declarative XAML model and, 214–215
element to element binding, 228–229
ItemsControl element and, 203, 230–231
overview of, 213
principles, 213–214
review, 230
Silverlight 3 features, 9
value converters and, 225–227
Data files, XAPs (deployment packages)

and, 30
Data services, 13
Data sources
connecting to elements, 215
creating collection-based data source, 222–223
for data binding, 214
Data synchronization
automatic connection and
synchronization, 214
data binding and, 219–221
Data validation, 227–228
Data values, converters for, 225–227
DataContext property
FrameworkElement, 215–216
inheritance and, 218
DateTime object, time format in, 137
Debugging, HLSL pixel shader, 244–245
Declarative XML instantiation language.
See XAML
Deep Zoom Composer
creating images in deep zoom format, 7
tools for creating Silverlight content, 1
Dependency properties, binding parameters
to, 245–246
Deployment
deploying applications, 12
Web deployment of Silverlight
applications, 3
Deployment packages. See XAPs
(deployment packages)
Deployment.OutOfBrowserSettings
property, 21
Designers, collaboration in application
development, 13
Developers, collaboration in application
development, 13
Development, ease of Silverlight
development, 5
Digital Rights Management (DRM)
media delivery and, 179
media source and, 182
Direct X
HLSL (High Level Shader Language) pixel
shader. See HLSL (High Level Shader
Language) pixel shader
prerequisites for GPU acceleration, 253
for use with Windows OSs, 263
Display
clipping display area, 77–78
connecting video markers to application
display, 189–190
Dots per inch (DPI), image formats and, 62
Double, type conversion, 25–26
DoubleAnimation element
BeginTime property, 144
linear interpolation based on From and To
values, 140–141
overview of, 137
DoubleAnimationUsingKeyFrames
element, 141–142
Downloader, application, 34–35
DownloadProgress, MediaElement, 187
DPI (dots per inch), image
formats and, 62
Draw loop process, in graphics, 82–83
DRM (Digital Rights Management)  
media delivery and, 179  
media source and, 182  
Dropout control, readability of text and, 113  
DropShadowEffect  
built-in effects, 234–235  
performance and, 250  
running, 249  
Durations, animation, 137

E
EaseMode properties, 146
Easing functions  
creating custom, 147  
list of, 146  
for nonlinear interpolation animations, 144–146
East Asian fonts, 98, 106
Effect property, UIElement, 234, 238
Effects. See also HLSL (High Level Shader Language) pixel shader  
applying, 234–235  
BlurEffect and DropShadowEffect, 249  
built-in, 234  
creating, 236  
elements, 234  
overview of, 233  
principles, 233  
real-time speed, 233–234  
review, 249  
Silverlight 3 features, 7  
surface generation, 248
Element tree, applications, 36
Elements  
animation, 136–138  
binding data to, 215–218  
changing size, 151  
connecting data sources to, 215  
control, 198–199  
effect, 234  
element to element data binding, 228–229  
graphics, See Graphics layout, 158  
media, 180  
text, 99–100
Ellipses  
animation changing rectangle into, 143–144  
creating custom XAML elements, 27–28  
drawing, 50–51  
element tree model when drawing, 36  
limiting HLSL pixel shader to ellipse shape, 239–240  
stroke widening process applied to, 70  
EnableCacheVisualization flag, on plug-in, 260  
EnableFrameRateCounter flag, on plug-in, 260  
EnableGPUAcceleration flag, on plug-in, 253–254  
EvenOdd rule, fill rules, 59, 61  
Event bubbling, mouse, 124–125  
Event handlers  
CurrentStateChanged event handler, 186  
delegating control actions to event handlers, 184  
keyboard, 126  
MarkerReached event handler, 189–190  
mouse, 120  
options for input events, 118–119  
XAML, 26–27
Events  
application, 37  
input. See Input events  
layout, 173–174  
startup and exit, 18  
triggers responding to, 137  
Exit event, saving state when leaving applications, 18
Expression Blend  
creating Silverlight content, 1  
creating Silverlight graphics, 7, 47  
customizing controls, 198  
editing application layout and graphic design, 13  
editing XAML with, 18  
Silverlight development and, 5
Expression Design  
creating Silverlight content, 1  
as Silverlight graphics tool, 7, 47
Expression Media Encoder  
connecting video markers to application display, 189  
creating Silverlight content, 1
Index

F
Figure element, Shapes, 58
File formats
audio files, 9
image files, 62
Fill property, Shape element, 48
Fills
fill rules for shapes, 58–61
solid color brushes, 63
Filters, bilinear, 88–92
Flashing cursors, 131
float types, HLSL pixel shader and, 240
Floating-point parameter, defining for effects, 245
Focus
controlling keyboard focus, 127
releasing/calling, 116
FontFamily property, 106
Fonts
caching and, 112
cross-platform consistency, 98
downloading, 111–112
enumerating font files, 110–111
fallback list, 111
FontFamily and FontSize properties, 106–107
FontWeight and FontStyle properties, 104–105
readability and, 99
XAPs (deployment packages) and, 30
FontSize property, 106–107
FontStyle property, 104–105
FontWeight property, 104–105
Foreground property, adding color to text, 105
Formats
audio files, 9
image files, 62
Formats, time, 137
Formatting text, 103–105
Frame-based animation, vs. time-based, 132–134
Framerates, diagnosing performance problems, 259
Frames per second, setting property change speed for animation, 149
FrameworkElement
DataContext property, 215–216
SetBinding property, 229
Full screen
adding capability to video player, 188–189
delivering full screen experience, 179–180
performance and, 189
fxc.exe, DirectX compiler, 237
G
Gamma correction, for anti-aliased text, 113
Gaussian blur, BlurEffect, 235
GDI (Graphics Device Interface), 99
Geometry
converting to pixels, 83
type conversion, 25–26
GetPosition method, for information on mouse position, 123–124
Glyphs element
for advanced typography, 97
caching glyphs, 112
displaying mathematical formula with, 107–110
font download, 111–112
function of, 107
OriginY property, 110
uniform scaling, 98–99
GoToState method, of VisualStateManager, 207–211
GPU acceleration
better performance with, 252
CacheMode property, 254–259
creating graphic devices and, 263
diagnosing performance problems, 259–263
enabling, 253–254
image quality and, 252
multiple monitors and, 265
overview of, 251
prerequisites for, 253
principles, 251–252
rendering stages when enabled, 265
review, 265
Silverlight 3 features, 7
stability and, 252–253
video memory allocation and, 264
video performance and, 189
GPU (graphics processing unit)
bilinear filtering and, 252
rendering graphics with, 251
GPU (continued)
  video memory and, 254
Gradient brushes
  focal, 67
  linear, 65–66
  radial, 66–67
  SpreadMethod property, 68
Graphic devices, creating, 263
Graphics
  3D transforms, 75–77
  animation, 8
  balancing image quality and speed, 47–48
  bilinear filtering, 88–92
  brushes, 62
Canvas element, 72
  clipping, 77–78
  cross platform consistency, 47
  dashes added to strokes, 71–72
draw loop process, 82–83
  ellipses, 50–51
  fill rules, 58–61
  getting content to screen, 95
  gradient brushes, 65–68
  image brushes, 68–69
  images, 61–63
  immediate mode API, 43–44
  incremental redraw, 92–94
library, 12
  lines, 51–55
  multi-core rendering, 94–95
  occlusion culling, 94
  opacity, 79
  OpacityMask property, 79–81
  overview of, 41
Path element, 55–58
  principles, 41
  rasterization, 83–88
rectangles, 48–50
  retained mode API, 43–47
review, 96
shapes, 48
Silverlight 3 features, 7
  solid color brush, 63–64
  strokes, 69–71
  tools for, 47
  transforms, 73–75
vector graphics and bitmap images, 41–43
Graphics adapters, multiple monitors
  and, 265
Graphics Device Interface (GDI), 99
Graphics drivers, 252–253
Graphics processing unit. See GPU (graphics processing unit)
Grid element, 162–167
  arranging Button elements in grid pattern, 162–163
centering TextBlock in cell of, 166
  as child of another element, 163–164
  having single element span multiple columns or rows, 163
  positioning elements in grid pattern, 162
  setting maximum widths and heights, 167
  specifying row/column widths/heights, 164–165
text layout and, 103
H
  H264 video
    media delivery and, 178
    as media source, 182
  Silverlight 3 media features, 9, 177
  HD video, 8, 177
Height property
  automatic sizing and positioning of elements, 174
  ImageElement, 62
  MediaElement, 181
High Level Shader Language. See HLSL (High Level Shader Language) pixel shader
Hinting, TrueType program for, 112
Hit-testing, mouse events, 125–126
HLSL (High Level Shader Language) pixel shader
  applying, 237–238
  binding parameters to dependency properties, 245–246
code for, 236
  compiling, 237
debugging, 244–245
defining inputs to, 242–243
  functions of, 236
  image distortion effect, 247
  limiting effect to ellipse, 239–240
  list of common functions, 240–242
  Mask property applied to image brush, 243–244
tex2D function, 240
Index

HorizontalAlignment property, Grid element, 166
HTML controls, Silverlight controls compared with, 9
HTML page application components, 16
in structure of Silverlight applications, 20
Web deployment of Silverlight applications, 12

Illustrator (Adobe), exporter for XAML files, 47
Image distortion effect, HLSL pixel shader, 247
Image element applying effect directly to, 249
overview of, 61–63
video samples compared to, 191
ImageBrush element Mask property applied to, 243–244
overview of, 68–69
VideoBrush compared with, 191
Images balancing image quality and speed, 47–48
bitmap images, 41–43
GPU acceleration improving image quality, 252
resolving seaming problems due to non-integer positioning, 168–169
scaling, 42–43
Immediate mode graphics API compared with retained mode, 45
overview of, 43
removing rectangle with, 44
Incremental redraw, graphic objects, 92–94
Inheritance
DataContext property and, 218
TextBlock element and, 105
Initialization, XAML parser and, 35–36
INotifyCollectionChanged, 230
INotifyPropertyChanged, 219–220
Input events accessibility and, 127
asynchronous firing, 129
controlling keyboard focus, 127
cross-platform consistency, 117
custom controls, 117
event handler options for, 118–119
hit-testing mouse events, 125–126
keyboard event handlers, 126
keyboard event process, 129
mouse capture, 119–124
mouse event bubbling, 124–125
mouse event handlers, 120
mouse event process, 128
multiple languages and, 118
overview of, 115
principles, 115–116
receiving Web browser events, 116–117
review, 130
Silverlight 3 features, 8
Instantiation of controls, 209–210
XAML parser and, 35
Internet Explorer, 35. See also Web browsers
Interpolation animations easing functions for nonlinear animation, 144–146
linear interpolation based on From and To values, 140–141
overview of, 140
IsHitTestVisible property, 126
Italic format, text, 104
Items collection, 202
ItemsControl element binding data to collections, 221–224
binding data to comboboxes, 223–224
binding data to lists, 222, 230–231
creating custom control by inheriting from, 207
data binding and, 203
data validation and, 227
overview of, 201–203
ItemsPanel element, 202
ItemsPresenter element, 202
ItemsSource property, ItemsControl element, 222, 230
ItemsTemplate property, 202
IValueConverter, converting from Priority type to Brush type, 225–226

JavaScript, accessing application libraries from, 5
JPG image format, 62
K

Keyboard
asynchronous firing, 129
controlling focus, 127
cross-platform consistency, 117
delegation to appropriate element, 117
event handlers, 126
input events and, 8
multiple language support, 118
process of, 129
receiving Web browser events, 116
KeyDown event handler, 126, 129
Keyframe animations
animation between different objects, 143–144
discrete key frame animation without interpolation, 142–143
DoubleAnimationUsingKeyFrames element, 141–144
overview of, 140
KeyUp event handler, 126, 129

L

Language Integrated Query (LINQ), 214
Languages, multiple language support, 118
Layout
algorithm for, 174
application layout, 38–39
automatic sizing and positioning, 154–157
Border element, 159–160
building custom, 169–173
Canvas element, 158–159
changing element size, 151
elements, 158
events, 173–174
Grid element, 162–167
overview of, 153
performance tips for, 176
principles, 153–154
readability of text and, 99
review, 176
rounding, 167–169
scale independence, 157
Silverlight 3 features, 8
StackPanel element, 161
steps in, 175
TextBlock element, 100–103
LayoutUpdated event, 173–174

Libraries, Silverlight, 12–13
Linear gradient brushes, 65
LineBreak element, TextBlock element, 101
Lines, drawing, 51–55
LINQ (Language Integrated Query), 214
LINQ to SQL, 214
LINQ to XML, 214
Linux, 5–6
List controls, ItemsControl element for, 201–203
ListBox element
data binding and, 231
implementing, 199
Lists, binding data to, 222
Live broadcast streaming. See Broadcast streaming
Loaded event trigger
overview of, 137
starting an animation with, 138

M

Mac OSs
keyboard shortcuts and, 117
prerequisites for GPU acceleration, 253
Silverlight cross platform support, 5
video memory managers, 264
MarkerReached event handler, 189–190
Markers, connecting video markers to application display, 189–190
Mask property, applied to image brush, 243–244
MaxFrameRate, draw loop process and, 82
MaxWidth and MaxHeight properties, Grid element, 167
MeasureOverride method, WrapPanel class, 170, 173
Media
adding controls to media players, 183–187
connecting video markers to application display, 189–190
content delivery, 178–179
creating media players, 180
elements, 180
full screen capability, 179–180, 188–189
integrating with other application content, 178
media players, 191–192
overview of, 177
Media (continued)
  playing video, 180–183
  principles, 178
  review, 193
  sources, 190–191
  video and audio capabilities in
  Silverlight, 8–9
  video effects and transitions, 187–188
  writing media applications, 4–5
Media players
  adding controls to, 183–187
  adding full screen capability to video
  player, 188–189
  creating, 180
  steps in operation of, 191–192
MediaElement
  applying effect directly to, 249
  delegating control actions to event
  handlers, 184
  displaying contents of, 180
  playing video with, 180–181
  Position property, 187
Memory
  allocation, 264
  CPU vs. GPU access, 264
  diagnosing performance problems, 259
  video memory, 254
Menus, sliding menu of controls, 131
Microsoft
  LINQ (Language Integrated Query), 214
  Windows (OSs). See Windows OSs
  Word, 107
MinWidth and MinHeight properties, Grid
  element, 167
Mip-maps
  effect surface generation and, 248
  scaling and, 92
Mode property, Binding object and, 220
Moonlight project, Novell, 5–6
Motion blur, reducing jumpy quality of
  animation with, 134
Mouse
  asynchronous firing of events, 129
  capture, 119–124
  cross-platform consistency, 117
  delegation to appropriate element, 117
  event bubbling, 124–125
  event handlers, 120
  hit-testing, 125–126
  input events and, 8
  process of, 128
  receiving Web browser events, 116
  starting animation in response to mouse
  event, 138–139
MouseEnter events, 122–123
MouseLeave events, 122–123
MouseLeftButtonDown event handler,
  120–122
MouseLeftButtonUp event handler, 120–122
MouseMove events, 122–123
MouseWheel event handler, 119
MP3 audio format, 9
MPEG audio format
  media delivery and, 178–179
  media source and, 182
Multi-core rendering, graphics, 94–95
Multiple monitors, GPU acceleration
  and, 265

N
Namespace, XAML, 23–24
.NET CLR. See CLR (Common Language
  Runtime)
Networking
  retrieving data for client-side
  applications, 13
  Silverlight 3 features, 9
NonZero rule, fill rules, 59–61
NotifyOnValidationError, 227–228
Novell Moonlight project, 5–6

O
Objects
  Binding object, 215–217, 220, 229–230
  constructing application objects with
    XAML, 18–20, 22–23
  Date-Time object, 137
  incremental redraw, 92–94
  keyframe animations between, 143–144
  Occlusion culling, graphics, 94
Opacity
  changing element graphics, 150–151
  overview of, 79
Opacity property
  exceptions to visibility rule in
    hit-testing, 126
  passing to pixel shader, 245–246
### Index

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpacityMask property</td>
<td>79–81</td>
</tr>
<tr>
<td>OpenGL2.x, 253, 263</td>
<td></td>
</tr>
<tr>
<td>Orientation property, StackPanel element</td>
<td>161</td>
</tr>
<tr>
<td>OSs (operating system) crossing platform support, 5–6, 13</td>
<td></td>
</tr>
<tr>
<td>Mac OSs. See Mac OSs</td>
<td></td>
</tr>
<tr>
<td>multiple languages support, 118</td>
<td></td>
</tr>
<tr>
<td>Windows (OSs). See Windows OSs</td>
<td></td>
</tr>
<tr>
<td>Padding property</td>
<td>160</td>
</tr>
<tr>
<td>Panel class, WrapPanel derived from</td>
<td>170</td>
</tr>
<tr>
<td>Parser, XAML, 35</td>
<td></td>
</tr>
<tr>
<td>Path element</td>
<td></td>
</tr>
<tr>
<td>extending Shape element, 55–57</td>
<td></td>
</tr>
<tr>
<td>mini-language commands, 57</td>
<td></td>
</tr>
<tr>
<td>Pause operation, media players, 184</td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td></td>
</tr>
<tr>
<td>animation and, 136</td>
<td></td>
</tr>
<tr>
<td>BlurEffect and DropShadowEffect and, 250</td>
<td></td>
</tr>
<tr>
<td>diagnosing performance problems, 259–263</td>
<td></td>
</tr>
<tr>
<td>GPU acceleration and, 189, 252</td>
<td></td>
</tr>
<tr>
<td>isolating slowest changing properties in animation, 149</td>
<td></td>
</tr>
<tr>
<td>layout and, 176</td>
<td></td>
</tr>
<tr>
<td>Web browsers and, 34</td>
<td></td>
</tr>
<tr>
<td>XAPs (deployment packages) benefits, 31</td>
<td></td>
</tr>
<tr>
<td>Pixel shader. See HLSL (High Level Shader Language) pixel shader</td>
<td></td>
</tr>
<tr>
<td>Pixels</td>
<td></td>
</tr>
<tr>
<td>converting geometry to, 83</td>
<td></td>
</tr>
<tr>
<td>GPU capable of higher fill rates than CPUs, 252</td>
<td></td>
</tr>
<tr>
<td>uniform scaling and, 99</td>
<td></td>
</tr>
<tr>
<td>Plug-in, Silverlight</td>
<td></td>
</tr>
<tr>
<td>EnableCacheVisualization flag on, 260</td>
<td></td>
</tr>
<tr>
<td>EnableFrameRateCounter flag on, 260</td>
<td></td>
</tr>
<tr>
<td>EnableGPUAcceleration flag on, 253–254</td>
<td></td>
</tr>
<tr>
<td>hosting with windowless=false, 121</td>
<td></td>
</tr>
<tr>
<td>as primary entry point for Silverlight, 33–34</td>
<td></td>
</tr>
<tr>
<td>releasing/calling focus, 116</td>
<td></td>
</tr>
<tr>
<td>setting focus, 127</td>
<td></td>
</tr>
<tr>
<td>PNG image format, 62</td>
<td></td>
</tr>
<tr>
<td>Point sizes, fonts, 107</td>
<td></td>
</tr>
<tr>
<td>Point, type conversion, 25–26</td>
<td></td>
</tr>
<tr>
<td>PointAnimation element</td>
<td></td>
</tr>
<tr>
<td>BeginTime property, 144</td>
<td></td>
</tr>
<tr>
<td>linear interpolation based on From and To values, 140–141</td>
<td></td>
</tr>
<tr>
<td>Position property, MediaElement, 187</td>
<td></td>
</tr>
<tr>
<td>Positioning elements, layout, 155</td>
<td></td>
</tr>
<tr>
<td>Prerequisites, for GPU acceleration, 253</td>
<td></td>
</tr>
<tr>
<td>Principles</td>
<td></td>
</tr>
<tr>
<td>animation, 132</td>
<td></td>
</tr>
<tr>
<td>application, 11</td>
<td></td>
</tr>
<tr>
<td>control, 196</td>
<td></td>
</tr>
<tr>
<td>data binding, 213–214</td>
<td></td>
</tr>
<tr>
<td>effect, 233</td>
<td></td>
</tr>
<tr>
<td>GPU acceleration, 251–252</td>
<td></td>
</tr>
<tr>
<td>graphic, 41</td>
<td></td>
</tr>
<tr>
<td>input event, 115–116</td>
<td></td>
</tr>
<tr>
<td>layout, 153–154</td>
<td></td>
</tr>
<tr>
<td>media, 178</td>
<td></td>
</tr>
<tr>
<td>Silverlight, 2–3</td>
<td></td>
</tr>
<tr>
<td>text, 98</td>
<td></td>
</tr>
<tr>
<td>Progress indicator, playing video and, 181</td>
<td></td>
</tr>
<tr>
<td>Progressive download</td>
<td></td>
</tr>
<tr>
<td>Silverlight supporting, 177, 179</td>
<td></td>
</tr>
<tr>
<td>sources for, 190–191</td>
<td></td>
</tr>
<tr>
<td>Projections, 3D transforms, 75</td>
<td></td>
</tr>
<tr>
<td>Property callback method, 245</td>
<td></td>
</tr>
<tr>
<td>Property change speed changing quickly for animation performance, 136</td>
<td></td>
</tr>
<tr>
<td>changing values over time for smooth animation, 134–135</td>
<td></td>
</tr>
<tr>
<td>setting for animation, 149–151</td>
<td></td>
</tr>
<tr>
<td>Quality, balancing image quality and speed, 47–48</td>
<td></td>
</tr>
<tr>
<td>Query languages, 214</td>
<td></td>
</tr>
<tr>
<td>Radial gradient brushes, 66–67</td>
<td></td>
</tr>
<tr>
<td>RadiusX and RadiusY properties, for rounding corners of rectangles, 50</td>
<td></td>
</tr>
<tr>
<td>Rasterization, 83–88</td>
<td></td>
</tr>
<tr>
<td>anti-aliasing and, 85–86</td>
<td></td>
</tr>
<tr>
<td>artifacts of anti-aliasing, 87–88</td>
<td></td>
</tr>
<tr>
<td>effect surface generation and, 248</td>
<td></td>
</tr>
<tr>
<td>occlusion culling and, 94</td>
<td></td>
</tr>
<tr>
<td>readability of text and, 99, 112–113</td>
<td></td>
</tr>
<tr>
<td>sampling and, 83–85</td>
<td></td>
</tr>
<tr>
<td>Readability, rendering text for, 99</td>
<td></td>
</tr>
</tbody>
</table>
Real-time speed, effects, 233–234
Rectangles
  automatic sizing and positioning, 174
  changing to ellipse, 143–144
drawing, 48–50
Red, green, blue (RGB) color, 64
Redrawing objects, incremental
  redraw, 92–94
RenderAtScale property, 257–259
Rendering
  applications, 39
cross platform consistency, 47
multi-core rendering, 94–95
text for readability, 99
RenderTransform, Canvases, 73
RepeatBehavior property, managing
  animations on timeline, 144
Resources, performance tips for, 34
Retained mode graphics API
  compared with immediate mode, 45
options for constructing, 46–47
overview of, 43
removing rectangle with, 44
RGB (red, green, blue) color, 64
RIAs (rich Internet applications), 4–5
Rich Internet applications (RIAs), 4–5
RotateTransform, 73
Rotation
  shapes, 150
transforms for, 73
Rounding layout sizes, to resolve seaming
  problems, 167–169
RowDefinition properties, Grid element,
  164–165
Runtime architecture, applications, 32–33
RuntimeVersion property, Silverlight
  versions, 17

S
Sampling
  converting geometry to pixels, 83–85
  converting time-based animation into
  frames, 134
Scalable vector graphics (SVG), 4
ScaleTransform, 73
Scaling
  bilinear filtering and, 91–92
  mip-mapping and, 92
  scale independent layout, 157
  shapes, 150
text, 98–99
  transforms for, 73
Screen
  animation, 151
  getting graphic content to, 95
scRGB colors, 64
SDK (Software Development Kit)
  DateTime object, 137
  using Visual Studio with, 5
Security
  applications, 14
  Silverlight, 3–4
SetBinding property,
  FrameworkElement, 229
Setter elements, setting styles with, 207
Shapes
  changing position of (animation), 150
eellipses, 50–51
  Figure element, 58
  fill rules, 58–61
  lines, 51–55
  path element, 55–58
  rectangles, 48–50
  rotating or scaling, 150
Shape element, 48
transforms for, 73
Silverlight overview, 1–10
cross platform support, 5–6
ease of development, 5
introduction to, 1–2
media and rich Internet applications, 4–5
principles, 2–3
review, 10
security, 3–4
Silverlight 3 features, 6–9
Web deployment, 3
SIMD (Single Instruction Multiple Data), 234
SizeChanged event, layout events, 173–174
Sizing elements
  automatic application resize, 156–157
  changing element or text size, 151
  layout, 154–155
Skews, transforms for, 73
Software Development Kit (SDK)
  DateTime object, 137
  using Visual Studio with, 5
Solid color brush, 63–64
source parameter, XAP package, 16
Source property
images, 62
MediaElement, 182, 190–191
Silverlight plug-in, 33
Sources, for data binding, 214
Speed, balancing image quality with, 47–48
SpreadMethod property, Gradient
brushes, 68
sRGB colors, 64
Stability, GPU acceleration and, 252–253
StackPanel element
arranging elements in horizontal or vertical
stacks, 161
positioning elements, 155
resolving seaming problems due to non-
integer positioning, 168–169
text layout and, 103
Startup event handler, 18
Stop operation, media players, 184
Stop(), Storyboard element, 144
Storage services, storing local state with, 13
Storyboard element
BeginTime property, 144
list of animations to run, 137
Stop() method, 144
Streaming media
Silverlight 3 media features, 177
sources for, 190–191
String, type conversion, 25–26
Strokes
adding dashes, 71–72
changing display of, 150
drawing lines, 51
drawing rectangles, 49
drawing shapes, 48
overview of, 69
solid color brushes, 63
widening process, 70–71
Style element, controls, 206–207
Subpixel positioning, text characters, 99
Surface generation, effects, 248
SVG (scalable vector graphics), 4
Synchronization. See Data synchronization

T
Tab operations, setting focus via, 127
Template property
ContentControl element, 199–201
ItemsControl element, 202
Templates
ContentControl element, 199–201
control templates, 210
ItemsControl element, 202
Tex2D function, HLSL pixel shader, 240
Text
animation of, 107, 150
caching and, 112
changing size, 151
cross-platform consistency, 98
elements, 99–100
exceptions to visibility rule in
hit-testing, 126
font download, 111–112
font enumeration, 110–111
font fallback list, 111
fonts, 106–107
formatting, 103–105
Glyphs element, 107–110
layout, 100–103
layout sizing and, 156
overview of, 97–98
principles, 98
rasterization and, 112–113
rendering for readability, 99
review, 113
scale independent layout, 98–99, 157
Silverlight 3 features, 7–8
subpixel positioning of text
characters, 99
TextAlignment property, text layout, 103
TextBlock element
Border element and, 159–160
centering within grid cell, 166
displaying paragraph of text with, 100
enumerating font files/families, 110–111
font download, 111–112
font fallback list, 111
fonts, 106–107
formatting text, 103–105
layout of text, 100–103
overview of, 97
Style element, 207
uniform scaling, 98–99
TextBox element
font fallback list, 111
mapping several keys to single key, 129
multiple language support, 118
text editing with, 97
Index

TextDecorations property, TextBlock element, 104–105
TextWrapping property, TextBlock element, 100–101
Time-based animation, vs. frame-based, 132–134
Time formats, 137
Timelines, animation and, 145
Timers
setting up in animation, 148
window.setInterval, 134–135
Toolkit, controls in, 197
Transforms
3D transforms, 75–77
animating brush transform, 150
overview of, 73–75
Transitions, video, 187–188
TranslateTransform, 73
Transparent backgrounds
causing slowdown of animation, 151
windowless=true and, 96
Triangles, stroke widening process
applied to, 71
Triggers collection, Canvas element, 137
Triggers, responding to events, 137
TrueType, 112
Type converters, XAML, 24–26

U
UI (user interface)
animation and, 131
defining UI controls, 183–184
using UI controls, 13
UIElement
CacheMode property, 254–259
Effect property, 234, 238
OpacityMask property, 79
Underlining text, 104
URI, referencing components in XAPs, 30–31
User interface. See UI (user interface)
UserController element
building control derived from, 195
c Control instantiation and, 209–210

V
ValidateOnExceptions flag, 227–228
Value converters, data binding and, 225–227
VC1 decoder, 9, 178
Vector graphics
compared with bitmap images, 41–42
scaling, 42–43
Versions, RuntimeVersion property, 17
VerticalAlignment property, Grid element, 166
Video
adding full screen capability to video player, 188–189
decoders, 5–6
delivering full screen experience, 179–180
effects and transitions, 187–188
memory allocation, 264
playing video, 180–183
Silverlight capabilities, 8–9
Video markers, connecting to application display, 189–190
Video memory
diagnosing performance problems, 259
GPU access to, 254, 264
Video memory managers, 264
VideoBrush
compared with ImageBrush, 191
displaying MediaElement contents, 180
effects and transitions, 187–188
Visual Basic, 2, 5
Visual Studio
creating Silverlight applications with, 14–15
Silverlight development and, 5
tools for creating Silverlight content, 1
writing application code, 13
VisualStateManager
customizing built-in controls, 203–206
GoToState method of, 207–211
overview of, 195

W
Web browsers
cross platform support, 13
delivering full screen experience, 179–180
downloading through, 35
graphics and media limitations of, 4–5
keyboard events and, 129
mouse events and, 128
performance tips for, 34
receiving Web browser events, 116–117
Web browsers (continued)
  taking applications out of browser, 12, 20–21
text and, 98
Web deployment
  hosting applications on Web pages, 12
  of Silverlight applications, 3
Web pages
  deploying applications from, 14
  hosting applications on, 12
  improving loading times, 34
Web sites, designer/developer collaboration
  and, 13
Width property
  automatic sizing and positioning of elements, 174
Grid element, 165
ImageElement, 62
MediaElement, 181
TextBlock element, 100–101
windowless=false
  hosting Silverlight plug-in with, 121
  receiving Web browser events and, 116
  for smooth animation, 151
windowless=true
  receiving Web browser events and, 116
  transparent backgrounds and, 96
Windows Media Audio. See WMA (Windows Media Audio)
Windows Media Server, 191
Windows Media Video. See WMV (Windows Media Video)
Windows OŚs
  GDI (Graphics Device Interface), 99
  keyboard shortcuts and, 117
  prerequisites for GPU acceleration, 253
  Silverlight cross platform support, 5
  video memory managers, 264
windows, resizing, 156
window.setInterval, changing property values over time, 134–135
WMA (Windows Media Audio)
  media delivery and, 179
  media source and, 182
  Silverlight 3 audio formats, 9
WMV (Windows Media Video)
  Silverlight 3 video formats, 177
  with VC1 decoder, 178
Word, Microsoft, 107
WrapPanel class, 169–173
Wrapping text, 100–101
WYSIWYG tools, editing XAML with, 18

X
XAML
  adding status field and name to controls, 185–186
  connecting data sources to elements, 216–217
  connecting event handlers to input events, 118–119
  constructing application objects, 22–23
  constructing application objects with, 18–20
  custom classes, 27–29
  customizing controls, 198
  data binding and, 214–215
  defining element content, 29–30
  defining UI controls, 183–184
  event handlers, 26–27
  exporters, 47
  for graphics and layout, 13
  namespace, 23–24
  type converters, 24–26
  visual display of Silverlight application as, 5
  XAPs (deployment packages) and, 7, 30
XAML parser, 35
XAPs (deployment packages)
  application deployment and, 12
  application structure and, 20
  downloader, 34–35
  packing XAML files, data files, and manifests as, 30–31
  performance benefits of, 31
  placing fonts in, 112
  in Silverlight application model, 7
XML
  data sources, 214
  declarative. See XAML
XPS printer driver
  exporter for XAML files, 47
  support for Glyphs element, 107

Z
ZIP files, 20. See also XAPs (deployment packages)
Zoom, creating images in deep zoom format, 7