Programmatic Differences Between Silverlight and WPF

A Whitepaper by Wintellect, LLC

Version 1.1
Build 0000

TABLE OF CONTENTS

INTRODUCTION

The Windows Presentation Foundation (WPF) is Microsoft's latest development platform for building next-generation Windows client applications. Silverlight, a subset of WPF, extends the platform to the web via the add-on mechanism available in most current browsers. As a subset of WPF, the ability to create rich Internet applications is unprecedented; however, there are some significant differences between the capabilities of Silverlight and WPF, as well as important differences in the programming features of the two technologies.

There are several architectural reasons for the discrepancies between WPF and Silverlight. First and foremost is that Silverlight 2 is a downloadable plug-in running within a browser. In order to insure that the plug-in is a small as possible, Microsoft built a smaller version of the .Net Framework, one that is highly optimized for size and is a small subset of the full .NET Framework, to embed within it. WPF, on the other hand, has full access to the main .Net Framework and all its associated assemblies. The difference between the smaller, downloadable .Net Framework and the full desktop version is one of the major disparities between the two platforms. In addition, the fact that Silverlight is rooted within the browser, and inherits the limitations of that environment, further differentiates the two platforms.

This whitepaper documents both the identical (or nearly so) functionality as well as the differences. WPF and Silverlight have many technological concepts in common: Dependency Properties, Data Binding, Custom Controls and Animation to name a few. In addition, this paper documents functionality implementations that may be available in only one technology or the other. For example, WPF implements a large library of controls for document handling, including printing and formatting large documents.

Finally, this whitepaper outlines some strategies in obtaining code reuse across both technologies. Due to minor differences in implementations of common elements, developers need to carefully plan their development in order to reuse code in both platforms. Of course, since certain functionality may not exist in one platform, developers may need to reduce the scope of the application in those cases.

One last note about this whitepaper: where appropriate, some information about the future version of Silverlight 3 has been provided. Please note that Silverlight 3 is currently in beta and any information may be subject to change. In addition, more Silverlight 3 information can be found at www.silverlight.net.

SIMILARITIES BETWEEN SILVERLIGHT AND WPF

Silverlight is generally considered to be a subset of WPF. There are very few pieces of functionality that exist solely within the Silverlight platform. That stated, though the two platforms have their differences, much of the implementations are similar.

Dependency Properties

Dependency properties are a key foundation for both Silverlight and WPF. Both platforms implement dependency properties in nearly identical ways; WPF allows for slightly more fine tuning the metadata when registering a dependency property.

The template for defining a dependency property looks as follows:

C#

```
public string MyProperty
{
    get { return (string)GetValue(MyPropertyProperty); }
    set { SetValue(MyPropertyProperty, value); }
}

public static readonly DependencyProperty MyPropertyProperty =
    DependencyProperty.Register("MyProperty", typeof(string),
                                typeof(MyClass), new PropertyMetadata(""));
```

Visual Basic

```
Property MyProperty() As String
   Get
      Return CType(GetValue(MyPropertyProperty), String)
   End Get
   Set(ByVal value As String)
      SetValue(MyPropertyProperty, value)
   End Set
End Property

Public Shared ReadOnly MyPropertyProperty As DependencyProperty = _
   DependencyProperty.Register("MyProperty", GetType(String), _
   GetType(MyCustomClass), New PropertyMetadata(""))
```

In essence, the backing store of a standard class' property is an object of type DependencyProperty instead of what would normally be a private field. By convention, the backing store appends the word "Property" to the end of the actual class' property name.

The Register() static method accepts a class of type PropertyMetadata that, among other things, sets the default value of the property. The differences between Silverlight and WPF reside in the metadata structure: Silverlight only supports the PropertyMetadata class here, while WPF supports several classes (all derived from PropertyMetadata).

Controls

Silverlight and WPF have a number of controls in common. Those controls not available in Silverlight can usually be found in either the Silverlight SDK or the Silverlight Toolkit. However, the implementation of the controls is not exactly the same in both platforms in most cases.

Microsoft is committed to bringing both platforms closer in terms of supported controls and functionality, and with each successive version the control libraries become more similar.

ANIMATIONS

Both Silverlight and WPF implement an animation mechanism. Animation is a complex topic, so this paper will just focus on some of the rudimentary aspects of it. At a basic level, animation is simply controlling a dependency property's value through time. For instance, moving an element from left to right involves changing its Canvas.Left property through time.

One way WPF and Silverlight implement this functionality is through linear interpolation, a method of smoothly transitioning a property from one value to another over time. Below is a snippet of XAML showing a simple animation declaration to move a button from left to right over the course of one second.

```
<Storyboard x:Name="ButtonAnimation">
   <DoubleAnimation
      Storyboard.TargetName="MyButton"
      Storyboard.TargetProperty="(Canvas.Left)"
      Duration="0:0:1"
      From="0"
      To="200" />
</Storyboard>
```

There are many ways to initiate the animation: through triggers (for WPF), the VisualStateManager (for Silverlight), EventTriggers within styles, or through procedural code.

The linear interpolation can be modified somewhat by adding AccelerationRatio and DecelerationRatio properties to the animation. These attributes essentially create three linear interpolations for the entire animation in order to modify the starting and stopping speeds. For example, a designer would use these attributes to have an object gradually pick up speed or stop suddenly. Unfortunately, Silverlight does not implement these two attributes, but the effect can be duplicated using keyframe animations with linear interpolation.

For more complex animations, WPF and Silverlight support keyframe animations, a method of animation specifying a value at a specific point in time (a keyframe). The animation system then interpolates the transition from one keyframe to the next. The method of interpolation can be as simple as a basic linear method seen before, discrete (that is, no interpolation), or more complex spline interpolations.

One aspect of animation present in WPF is path-based animations. Silverlight, unfortunately, does not have the facility to animate an object along a pre-defined path.

CUSTOM CONTROLS

Due to the styling and templating capabilities of WPF and Silverlight, the need for custom and user controls is less than in older technologies. However, there are scenarios when the development of such a control becomes necessary; such as when no existing control supports the required functionality needed.

WPF and Silverlight define two types of user creatable controls: user controls and custom controls. User controls can be thought of as composite controls while custom controls are either derived from existing controls or are something completely new and derived from the Control base class itself.

See http://msdn.microsoft.com/en-us/magazine/cc721611 for a great article on creating custom controls within Silverlight.

User Controls

User controls are useful when needing to reuse a complex group of elements in multiple places within an application or across applications. User controls typically derive from the UserControl element and can be declared in XAML as follows:

```
<UserControl
      x:Class="MyProject.MyUserControl"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
>

  <!-- Content -->

</UserControl>
```

VIDEO

There are a few controls to play video resources in WPF and Silverlight. The MediaPlayer class is the major element in playing video from within an application. The MediaPlayer class wraps the functionality of Windows Media Player and therefore supports all the file formats that it does (such as .wmv, .avi, and .mpg). However, the MediaPlayer class is used from procedural code only and has no provision for XAML markup. Additionally, the MediaPlayer class is only available within WPF.

To declaratively use video in WPF or Silverlight, the MediaElement class is a FrameworkElement. Within WPF, the MediaElement wraps the functionality of MediaPlayer, which in turn wraps the Windows Media Player. However, since the MediaPlayer class does not exist in Silverlight the MediaElement is completely separate from any local media player.

The following snippet shows a method for embedding video into a XAML page:

```
    <MediaElement Width="640" Height="480" Source="MyVideo.wmv" />
```

Since the MediaElement is a Framework element, it supports many options to control its look-and-feel. In addition, within WPF only, a MediaElement can be contained within a VisualBrush and be used to "paint" any UIElement's brush properties (i.e. background, 3D surfaces, etc.).

INK AND STYLUS

Digital ink refers to a method of input using a stylus on a touch-sensitive surface. Tablet PCs, pressure-sensitive tablets, touch screens, and other electronic devices have the capability to create digital ink. In addition, the mouse may also be used as a stylus device, though it is generally more difficult to use in that manner.

WPF defines stylus-based input to distinguish between the traditional mouse and one of the touch-based input devices (where the input mechanism could be a digital pen or even a finger).

A special panel called an InkCanvas is the main element used to accept free-form ink input. The InkCanvas is useful for adding annotations to images and documents.

WPF supports ink-based gestures as well found in the System.Windows.Ink.ApplicationGesture enumeration. These gestures allow a user to control a system using only the stylus to perform commands such as Up, Down, Cut, Paste, etc.

Finally, since WPF allows the capture of handwritten text there is a facility for handwriting recognition. The class used to analyze handwriting is InkAnalyzer ; however, it requires adding references to several assemblies found on Tablet PC systems: IAWinFX.dll, IACore.dll, and IALoader.dll. In a Silverlight application, it is possible to get handwriting recognition through a server-side component.

DIFFERENCES BETWEEN SILVERLIGHT AND WPF

Silverlight is a XAML-based technology that runs within the sandbox of browser plug-in. As such, it implements a subset of both the .Net Framework and WPF functionality. Microsoft made the decision to cut features to reduce the download footprint of the Silverlight plug-in. Future versions of Silverlight will attempt to reduce the differences between the two platforms, and still retain a small download footprint. Microsoft will accomplish this feat by only downloading libraries that the Silverlight application actually uses.

ARCHITECTURE

The basic architecture of WPF is analogous to traditional WinForms development, with the addition of the Media Integration Layer (MIL). The MIL is wrapped by two core assemblies, Presentation Framework and Presentation Core. WPF applications are run on a Windows client platform, and, with the exception of browser-hosted applications, are hosted by the system.

Since WPF applications are run on the client, they have access to the full .Net Framework programming stack, including the browser-hosted model.

Silverlight applications, on the other hand, run within the plug-in model of the hosted web browser. The plug-in contains all the core code libraries that Silverlight needs to run. However, in an effort to minimize the download time for installing Silverlight, the entire .Net Framework is not loaded, instead, Microsoft created the Core CLR, a distinct subset of the .Net Framework runtime libraries.

The differences between the two .Net Framework libraries constitute the major hurdles for developers when building applications to run in both architectures. Shared libraries are not capable of being shared between platforms since the core referenced assemblies are different. Library assemblies must be targeted for one platform only; and therefore cannot be shared. However, this whitepaper will discuss various alternate methods for sharing similar code between the two platforms.

Deployment

Obviously, there are significant differences in deploying a Silverlight application compared with deploying a WPF application.

Silverlight applications are hosted within a web server and a web page. To minimize client download size, Silverlight uses an XAP archive. A XAP archive is a zip-compressed archive of the Silverlight application, plus a manifest file describing the content. Visual Studio automatically generates the XAP archive when using the Silverlight project template.

WPF applications can be deployed as a standalone application, ClickOnce application, or a XAML Browser application.

BAML/XAML

BAML is a binary form of the XAML in a WPF application. When compiled and deployed, the text-based XAML is reconstituted into binary form for quicker processing. Technically, the XAML is not compiled, just compacted into an efficient binary format, though one can argue the semantics.

In order to retrieve the XAML of a page or control dynamically, it must be compiled as an Embedded Resource. This will store the actual XAML as an embedded resource within the assembly. It is then possible to retrieve the XAML in code via the resource manager.

C#

```
Stream s =
this.GetType().Assembly.GetManifestResourceStream("MyProject.MyControl.xaml");
```

VISUAL BASIC

```
Dim s As Stream = _
    Me.GetType.Assembly.GetManifestResourceStream("MyProject.MyControl.xaml")
```

XAML

The eXtensible Application Markup Language is a dialect of XML that both WPF and Silverlight use to describe the user interface. Both platforms support a separate code-behind file to contain .Net programming code to manipulate the elements and controls declared within the XAML. The code-behind model is the same one first introduced with .Net at its inception to provide a separation between procedural code and presentation.

XAML Language

Most of the XAML language is similar between WPF and Silverlight. Differences begin appearing with the available UI elements available to each platform.

```
<UserControl x:Class="Sample"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>
  <Grid Background="LightGray">
    <!-- Additional content added here. -->
  </Grid>
</UserControl>
```

Silverlight does require that the root element contain a default “xmlns” namespace declaration, whereas WPF can assume an implicit value.

Additional namespaces added to the root element must point to assemblies that are deployed with the application. WPF applications can point to assemblies within the GAC or deployed elsewhere.

TargetType

Some elements use a TargetType attribute to specify the type of element being referenced. For example, Styles use TargetType to specify the type of element being styled. WPF can specify the type using the {x:Type} markup extension, or by using the short-hand notation that relies on the default TypeConverter for TargetType:

```
<Style x:Key="MyButtonStyle" TargetType="{x:Type Button}">
       …
</Style>
```

Silverlight can only use this short-hand notation to specify the TargetType:

```
<Style x:Key="MyButtonStyle" TargetType="Button">
```

```
      …
</Style>
```

Triggers

WPF supports a trigger mechanism to respond to certain end-user actions, such as moving the mouse over a UI element, or pressing a button. Triggers are usually defined as part of a style, as follows:

```
<Style x:Key="Triggers" TargetType="{x:Type Button}">
   <Style.Triggers>
      <Trigger Property="IsPressed" Value="true">
         <Setter Property="Foreground" Value="Green"/>
      </Trigger>
   </Style.Triggers>
</Style>
```

WPF supports multiple types of triggers, such as MultiTrigger, EventTrigger and DataTrigger.

MultiTrigger's allow more than one condition to be specified before the style is applied. For example,

```
<Style x:Key="Triggers" TargetType="{x:Type Button}">
   <Style.Triggers>
      <MultiTrigger>
         <MultiTrigger.Conditions>
            <Condition Property="IsMouseOver" Value="true" />
            <Condition Property="IsSelected" Value="true" />
         </MultiTrigger.Conditions>

         <Setter Property="Foreground" Value="Purple"/>
      </MultiTrigger>
   </Style.Triggers>
</Style>
```

EventTriggers are triggers that respond to RoutedEvents, and only activate animations, they cannot properties on elements.

```
<EventTrigger RoutedEvent="Button.MouseEnter">
   <EventTrigger.Actions>
      <BeginStoryboard>
         <Storyboard>
            <DoubleAnimation
               Duration="0:0:0.5"
               Storyboard.TargetProperty="Opacity"
               To="0.5" />
         </Storyboard>
      </BeginStoryboard>
   </EventTrigger.Actions>
</EventTrigger>
```

DataTriggers and MultiDataTriggers are triggers that respond to changes in the underlying data bound to an element. An example of a DataTrigger,

```
<Style TargetType="{x:Type ListBoxItem}">
   <Style.Triggers>
      <DataTrigger Binding="{Binding Path=Amount}" Value="0">
         <Setter Property="Foreground" Value="Red"/>
      </DataTrigger>
   </Style.Triggers>
</Style>
```

Since DataTriggers use the Binding syntax, more robust conditions can be specified through the use of binding converters, allowing the application to reduce a range of values to a boolean value. Additionally, the MultiDataTrigger, its syntax mirrors the MultiTrigger element, can be used to further define conditions.

Silverlight, however, does not implement the trigger mechanism, but instead introduces a concept called the Visual State Manager.

Visual State Manager

The Visual State Manager was introduced in Silverlight 2 to simplify control over visual states and transitions. WPF does not currently implement the Visual State Manager, however, the WPF Toolkit (http://wpf.codeplex.com), contains a preview of the VisualStateManager for WPF in the March 2009 release.

Within a ControlTemplate, the Visual State Manager would be used to define any number of visual states and effects.

```
<Canvas x:Name="MyCanvas">
  <vsm:VisualStateManager.VisualStateGroups>
    <vsm:VisualStateGroup x:Name="CommonStates">
      <vsm:VisualStateGroup.Transitions>
        <vsm:VisualTransition To="Normal" GeneratedDuration="0:0:0.2" />
        <vsm:VisualTransition To="MouseOver" GeneratedDuration="0:0:0.5" />
      </vsm:VisualStateGroup.Transitions>
      <vsm:VisualState x:Name="Normal" />
      <vsm:VisualState x:Name="MouseOver">
      <Storyboard>
        <ColorAnimation
          Storyboard.TargetName="MyShape"
          Storyboard.TargetProperty="(Shape.Fill).(SolidColorBrush.Color)"
          To="Yellow"
          Duration="0" />
      </Storyboard>
      </vsm:VisualState>
    </vsm:VisualStateGroup>
  </vsm:VisualStateManager.VisualStateGroups>

  <!-- Add Canvas visual elements -->
```

Within the application code, the state can be changed by a simple call to the VisualStateManager.

C#

```
void MyControl_MouseOver(object sender, MouseEventArgs e)
{
  VisualStateManager.GoToState(this, "MouseOver", true);
}
```

VISUAL BASIC

```
Private Sub MyShape_MouseEnter(ByVal sender As System.Object, _
      ByVal e As System.Windows.Input.MouseEventArgs)

   VisualStateManager.GoToState(Me, "MouseOver", True)
End Sub
```

Note, however, that the VisualStateManager does not completely replace or duplicate the functionality found in the Trigger mechanism. For example, there is no support for anything similar to MultiTrigger or DataTrigger.

Markup Extensions

XAML processors support extensions to the markup language in order to provide additional functionality not found within the default processing. For example, default processing accepts literal values for attributes, but in some scenarios, the attribute should reference a previously constructed object or access a static object.

The syntax for a XAML markup extension in an attribute is:

```
<ElementName Attribute="{MarkupExtension …}">
```

Markup extensions can also be used as a property element, and visually are indistinguishable from the default element syntax. Markup extensions in the x: namespace (the default mapping for XAML namespace) are XAML-defined markups and are not specific to WPF.

WPF has full support for markup extensions. The most notable are:

- StaticResource
- DynamicResource
- Binding
- RelativeSource
- TemplateBinding
- x:Type
- x:Static
- x:Null
- x:Array

WPF also supports the creation of custom markup extensions. Custom markup extensions should be named with the Extension suffix and derive from the MarkupExtension base class.

Silverlight implements limited support for markup extensions, using only x:Null, StaticResource, Binding, and TemplateBinding. Also, since Silverlight doesn't implement a public MarkupExtension class, there is no support for custom markup extensions. The Binding markup extension is the only extension that Silverlight supports that has a corresponding class accessible via code.

Silverlight also doesn't support the alternative syntax of adding the "Extension" suffix like WPF does. So, {x:Null} is valid, but {x:NullExtension} is not.

Markup extensions, in Silverlight, can not specify constructor parameters by name. Therefore {StaticResource myKey} is valid, but {StaticResource ResourceKey=myKey} is not.

USER INTERFACE

Most interaction between a user and the application occurs through the user interface as events. In WPF and Silverlight, most events are Routed Events (see ROUTED EVENTS for more information on the differences in Routed Events between Silverlight and WPF). Each event, regardless of type, defines a delegate for handling purposes. The definition typically only varies the event arguments passed to the handler.

C#

```
public delegate void RoutedEventHandler(
  Object sender,
  RoutedEventArgs e
)
```

VISUAL BASIC

```
Public Delegate Sub RoutedEventHandler ( _
  sender As Object, _
  e As RoutedEventArgs   _
)
```

This definition, in essence, is the base definition for all other UI related events. All routed event arguments derive from the base RoutedEventArgs class. This document will also discuss more specific routed events (such as mouse or keyboard events) in detail, so knowing that there are differences in the base class as well is important.

The table below shows the implementation of the RoutedEventArgs base class and the differences between WPF and Silverlight.

ROUTEDEVENTARGS

Property	WPF	Silverlight
Device	Yes	No

Handled	Yes	Yes
InputSource	Yes	No
OriginalSource	Yes	Yes
RoutedEvent	Yes	No
Source	Yes	No
Timestamp	Yes	No

The Handled property can be set in the event handlers to indicate to other listeners that the event has been successfully handled already. In essence, setting the Handled property to true would let other controls in the event hierarchy (either bubbled up, or tunneled down) know that the event has been successfully handled. That stated, the other controls could potentially ignore the fact and provide additional functionality.

Keyboard

Most UI elements support the KeyDown and KeyUp events. In both Silverlight and WPF, the handler's signature is:

C#

```
public delegate void KeyEventHandler(
   Object sender,
   KeyEventArgs e
)
```

VISUAL BASIC

```
Public Delegate Sub KeyEventHandler ( _
  sender As Object, _
  e As KeyEventArgs _
)
```

However there are some significant differences in between the implementations of KeyEventArgs in the two technologies.

Property	WPF	Silverlight
ImeProcessedKey	Yes	No
IsDown	Yes	No
IsRepeat	Yes	No
IsToggled	Yes	No
IsUp	Yes	No
Key	Yes	Yes
KeyboardDevice	Yes	No
KeyStates	Yes	No
SystemKey	Yes	No
PlatformKeyCode	No	Yes

In addition, the Key enumeration contains different values between Silverlight's version and WPF's. For example, Silverlight contains Key.Alt to represent the Alt key, while WPF's enumeration contains Key.LeftAlt and Key.RightAlt.

WPF also exposed two additional event handlers, PreviewKeyDown and PreviewKeyUp, that Silverlight does not implement.

Non-Windows Platforms

Microsoft released Silverlight plugins for both Windows and Mac OS X operating systems. WPF, on the other hand, is only currently supported on the Windows platform.

There is a drive by the open source community to port Silverlight to Linux. The project, Moonlight, only supports Silverlight 1.0, but the Silverlight 2.0 equivalent is under development. Moonlight is a sub-project under the Mono umbrella (an attempt to port the entire .Net Framework to Linux). Currently, there is no push for porting WPF itself to Linux through the Mono project. See http://www.mono-project.com for more information.

Silverlight supports non-Windows key trapping through the use of the PlatformKeyCode value returned as part of the KeyEventArgs class. However, the common Apple key on Mac platforms is available through the ModifierKeys enumeration in Silverlight only.

Mouse

Both Silverlight and WPF support mouse events to various degrees. The following table documents the different mouse events supported by the two platforms.

Mouse Event	WPF	Silverlight
LostMouseCapture	Yes	Yes
MouseDoubleClick	Yes	No
MouseDown	Yes	Yes
MouseEnter	Yes	Yes
MouseLeave	Yes	Yes
MouseLeftButtonDown	Yes	Yes
MouseLeftButtonUp	Yes	Yes
MouseMove	Yes	Yes
MouseRightButtonDown	Yes	No
MouseRightButtonUp	Yes	No
MouseUp	Yes	No
MouseWheel	Yes	No
PreviewMouseDoubleClick	Yes	No
PreviewMouseDown	Yes	No
PreviewMouseLeftButtonDown	Yes	No
PreviewMouseLeftButtonUp	Yes	No
PreviewMouseMove	Yes	No
PreviewMouseRightButtonDown	Yes	No
PreviewMouseRightButtonUp	Yes	No

PreviewMouseUp	Yes	No
PreviewMouseWheel	Yes	No

The mouse events use several delegates for mouse handling. For those events reacting to a mouse button the following delegate is used:

C#

```
public delegate void MouseButtonEventHandler(
   Object sender,
   MouseButtonEventArgs e
)
```

VISUAL BASIC

```
Public Delegate Sub MouseButtonEventHandler ( _
  sender As Object, _
  e As MouseButtonEventArgs _
)
```

For those events reacting to other mouse events, such as moving, the following delegate is used:

C#

```
public delegate void MouseEventHandler(
   Object sender,
   MouseEventArgs e
)
```

VISUAL BASIC

```
Public Delegate Sub MouseEventHandler ( _
  sender As Object, _
  e As MouseEventArgs _
)
```

The last event handler delegate is only used within WPF and handles mouse wheel events:

C#

```
public delegate void MouseWheelEventHandler(
   Object sender,
   MouseWheelEventArgs e
)
```

VISUAL BASIC

```
Public Delegate Sub MouseWheelEventHandler ( _
  sender As Object, _
  e As MouseWheelEventArgs _
)
```

However, as in the Keyboard event handler, the two Mouse EventArgs classes are implemented differently in Silverlight than in WPF.

The MouseEventArgs implementation:

Property / Method	WPF	Silverlight
GetPosition(IInputElement)	Yes	Yes
LeftButton	Yes	No
MiddleButton	Yes	No
MouseDevice	Yes	No
RightButton	Yes	No
StylusDevice	Yes	Yes
XButton1	Yes	No
XButton2	Yes	No

The MouseButtonEventArgs class adds some additional properties to the MouseEventArgs class. The additional implementation details are:

Property / Method	WPF	Silverlight
ButtonState	Yes	No
ChangedButton	Yes	No
ClickCount	Yes	No

There is no difference between MouseEventArgs and MouseButtonEventArgs within the current release of Silverlight 2. However, due to the separation of the two classes in Silverlight, the assumption that future support for the additional MouseButtonEventArgs properties is likely true.

In order to obtain right mouse button and wheel support in Silverlight, developers need to interact with the browser through JavaScript and pass the mouse data back to Silverlight. Since Silverlight is, at its core, a browser plug-in, it is reliant on the plug-in models of the browser. Most browsers have special handling for the right mouse click and wheel events, and therefore pose a problem for the Silverlight plug-in. However, there are several implementations solving this issue available on the web through the use of browser inter-operation capabilities.

IsEnabled

WPF's UIElement base class implements the IInputElement interface, in which the Boolean property IsEnabled is defined. By setting this property, a developer can control whether all the nested children are enabled or not.

In Silverlight, however, this interface is not implemented by UIElement, and the bubbling functionality of IsEnabled is lost. Possible workarounds are either the brute force method of setting each nested control independently or binding the IsEnabled property to a global setting.

Local File Access

WPF applications have a full range of capabilities in accessing local file resources depending upon the requirements of the application. Silverlight, on the other hand, has very limited access to local resources. Currently, Silverlight needs to acquire permission from the user via the OpenFileDialog to obtain Read access to a local file resource. Silverlight can only write data to isolated storage, but this is subject to change in later implementations of Silverlight.

OpenFileDialog

From a user-interface point of view, the OpenFileDialog in both WPF and Silverlight are virtually identical. Both implementations allow for multiple files to be selected.

The major difference between the two implementations is the handling of the selected file. In WPF, the file can be returned to the application as a path and the application can open it as it needs to (for writing or reading). In Silverlight file is returned to the application as a FileInfo object. The name of the file is known, but none of the path information is available. In addition, the Silverlight application can only obtain a read-only stream to the file (direct access to the file system is forbidden).

A minor difference between the two implementations is the setting of the dialog's title. WPF allows the application to set a title to the dialog window, while Silverlight does not.

SaveFileDialog

The SaveFileDialog allows an application to let the user decide where to store data. WPF can store information to the user's system without any user interaction as well.

The current implementation of Silverlight does not contain a SaveFileDialog class. The only method to persist data through sessions would be to use local IsolatedStorage. Unfortunately, this method would prevent the user from accessing the data outside the Silverlight application.

Silverlight 3 Note
The future release of Silverlight 3 will include an implementation for storing data on the local file system with the user's permission through the use the SaveFileDialog.

Sub-Pixel Rendering

WPF uses a technology called Pixel Snapping to reduce anti-aliasing effects on UI elements. Silverlight uses Layout Rounding to achieve a similar effect.

To turn Pixel Snapping on in WPF (pixel snapping is off by default) set the SnapsToDevicePixels property to true.

```
<Rectangle x:Name="MyShape" SnapsToDevicePixels="True" … />
```

Or in code:

C#

```
this.MyShape.SnapsToDevicePixels = true;
```

VISUAL BASIC

```
Me.MyShape.SnapsToDevicePixels = True
```

In Silverlight, the UseLayoutRounding property is set to true by default. To turn on sub-pixel rendering, set this property to false.

```
<Rectangle x:Name="MyShape" UseLayoutRounding="False" … />
```

Or in code:

C#

```
this.MyShape.UseLayoutRounding = false;
```

VISUAL BASIC

```
Me.MyShape.UseLayoutRounding = False
```

Font Support

As expected, WPF supports all the fonts installed in the Windows operating system. Silverlight, however, natively supports only a handful of fonts. Additional fonts must be embedded within a Silverlight application in order to use them, or dynamically downloaded as needed (see FONTS later in this paper for how to do this).

Supported local fonts for text elements in Silverlight are:

Arial
Arial Black
Comic Sans MS
Courier New
Georgia
Lucinda Grande / Lucida Sans Unicode
Times New Roman
Trebuchet MS
Verdana

Silverlight contains a fallback font called "Portable User Interface." This is the font used if none is specified. However, it is merely an alias to the Lucinda Grande / Lucida Sans font face on Windows platforms; it may be different on other operating systems. According to Microsoft documentation, Lucinda Grande and Lucinda Sans are aliases for the same font and usually named together for compatibility reasons.

Finally, there are several East Asian fonts that can be used if available on the local computer as well as additional ones if the local computer is running Windows or the Mac OS.

Adding additional fonts in Silverlight is a simple as creating adding the required fonts either as TrueType (ttf) files or compressed into a Zip archive as Content Items to the project. Setting the FontSource of a TextBlock control to the resource containing desired fonts will set the appropriate font. The font can then be used as follows, where “Custom Font” is the name of a font stored in the CustomFont.ttf file:

```
<TextBlock FontFamily="customfont.ttf#Custom Font">
      Written in my Custom Font.
</TextBlock>
```

And from a zip archive:

```
<TextBlock FontFamily="customfont.zip#Custom Font">
      Written in my Custom Font.
</TextBlock>
```

In WPF fonts can be added as Content Items to the application, and are separate from the applications binary assemblies. To insure that the fonts are available to the application upon deployment, the CopyToOutputDirectory element should be set to “PreserveNewest.”

Additionally, fonts can be added to a WPF application as Resource Items. The following XAML example shows how to reference a font resource:

```
<TextBlock FontFamily="./resources/#Custom Font">
      Written in my Custom Font.
</TextBlock>
```

Graphics

Both Silverlight and WPF support vector-based drawing elements as well as support for raster images in a variety of formats.

Most graphical functions in WPF are accelerated by offloading the graphics workload to the GPU. Silverlight, however, cannot hand off this processing to the graphics card, and therefore most perform all calculations in software.

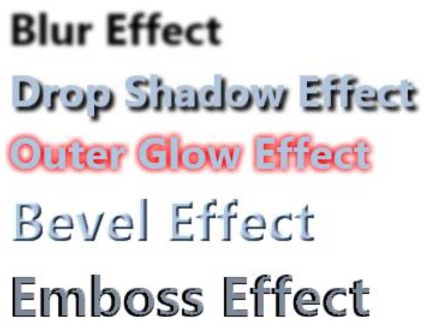
Silverlight 3 Note
Silverlight 3 will support some hardware acceleration through two new parameters on the Silverlight plug-in, EnableGPUAcceleration and EnableCacheVisualization. Additionally, an entire panel can be cached and rendered by the GPU through the use of the new CacheMode property.

2D Graphics

Bitmap Effects

Bitmap effects are simple pixel processing operations performed on WPF content. Since the bitmap effect occurs replaces the Visual object's built-in rendering, bitmap effects are rendered in software instead of accelerated through hardware.

There are five built-in bitmap effects implemented within WPF.



In addition to the default Bitmap Effects, WPF allows the creation of custom effects. However, creating custom effects entails writing an unmanaged COM library containing the effects and a managed-code wrapper for integration into WPF.

Silverlight does not support bitmap effects. In some cases, such as adding drop shadows, there are ways to work around the limitation by using duplicate elements and offsetting them from one another. For the more visually complex effects, such as blurring, there are no easy methods for recreating the effect.

Silverlight 3 Note

The future release of Silverlight 3 will include a form of bitmap effects called pixel-shader effects. The pixel-shader mechanism will perform many of the same effects previously unavailable to Silverlight applications, including writing custom pixel-shaders.

Important Note:

BitmapEffects are actively being discouraged by Microsoft and are being deprecated. Instead, the recommended approach is to use Effects; however, only blur and drop shadow effects have been created. Effects, not coincidentally, are written with the same pixel-shading technology that will be available to Silverlight 3; so shaders written today will be usable with Silverlight 3.

Brushes

Everything drawn on a WPF or Silverlight surface is visible due to brushes. Brushes draw everything from lines to backgrounds to text. Brushes come in several types and allow for great flexibility in their implementation.

There are a variety of brushes available to both Silverlight and WPF.

Brush	WPF	Silverlight
SolidColorBrush	Yes	Yes
LinearGradientBrush	Yes	Yes
RadialGradientBrush	Yes	Yes
ImageBrush	Yes	Yes
DrawingBrush	Yes	No
VisualBrush	Yes	No
TileBrush	Yes	No
VideoBrush	No	Yes

The lack of a TileBrush in Silverlight prevents both the Video and Image brushes from also being tiled.

Kiosks

Kiosks are typically defined as stand-alone devices with a complete user interface. Users are not allowed to access anything other than the kiosk application.

WPF Kiosk

To create a WPF kiosk, the application should be set to fill the entire screen and prevent the user from minimizing or otherwise interacting with any other part of the system.

To force the application's main window into a maximized, borderless form, the following XAML can be used:

```
<Window x:Class="TestWPF.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Kiosk"
    WindowStyle="None"
    WindowState="Maximized"
    Topmost="True">

   <!-- Add Window Content -->

</Window>
```

Additionally, the developers could trap certain key-strokes to prevent users from circumventing the application and accessing other system resources.

Silverlight Kiosk

Kiosks in Silverlight act slightly differently, because Silverlight applications cannot set the full-screen mode during instantiation or in the page's Loaded event handler. Instead, full –screen mode can only be entered upon response of a user-initiated event such as a key-press.

The following code will switch from full-screen to normal mode when the user presses the Escape key (assuming the handler is attached to the page's KeyDown event):

C#

```
void Page_KeyDown(object sender, KeyEventArgs e)
{
   if (e.Key == Key.Escape)
      App.Current.Host.Content.IsFullScreen =
          !App.Current.Host.Content.IsFullScreen;
}
```

VISUAL BASIC

```
Private Sub Page_KeyDown(ByVal sender As Object, ByVal e As KeyEventArgs)
      If e.Key = Key.Escape Then
             App.Current.Host.Content.IsFullScreen =  _
                    Not App.Current.Host.Content.IsFullScreen
      End If
End Sub
```

Note, full-screen mode for Silverlight only forces the Silverlight application into full screen; any surrounding HTML is lost in full-screen mode.

In addition, Silverlight also limits keyboard entry to the following keys: ESCAPE, UP, DOWN, LEFT, RIGHT, PAGE UP, PAGE DOWN, HOME, END, SPACEBAR, TAB, and ENTER. This limitation is by design so that a malicious web site is prevented from mimicking the system's operating system to trick the user.

Object Trees

Visual Base Class

The Visual base class is the class that all WPF visually rendering elements eventually derive from. It supports functionality such as printing and encoding the visual to a bitmap image. The Visual base class itself derives from DependencyObject.

Silverlight does not implement the Visual base class. Instead Silverlight UI elements have several different class hierarchy structures. Controls, such as TextBox and Button, derive from the Control base class, FrameworkElement, UIElement and then from DependencyObject. Other UI Elements, such as Panel or TextBlock, do not inherit from the Control base class, but still have FrameworkElement and UIElement in their base class hierarchy. Note that this list is not the absolute hierarchy for all controls, just some common base classes that developers can rely upon existing somewhere in the chain of base classes.

Even though Silverlight and WPF UI elements ultimately derive from DependencyObject, the class has little to do with UI rendering. Instead the DependencyObject class allows objects to participate in the dependency property system. DependencyObject classes are not necessarily visual elements.

This difference makes writing a shared code base difficult at best since the lowest common UI-related base classes are not compatible programmatically.

VisualTreeHelper

The VisualTreeHelper class allows the developer to examine and work with the visual tree structure. The implementation of this helper class works around the limitation of Silverlight not having a Visual base class.

Both WPF and Silverlight contain this helper class; however, they are implemented differently.

Method	WPF	Silverlight
FindElementsInHostCoordinates	No	Yes
GetBitmapEffect	Yes	No
GetBitmapEffectInput	Yes	No
GetChild	Yes	Yes
GetChildrenCount	Yes	Yes
GetClip	Yes	No
GetContentBounds	Yes	No
GetDesecendantBounds	Yes	No
GetDrawing	Yes	No
GetEdgeMode	Yes	No
GetEffect	Yes	No
GetOffset	Yes	No
GetOpacity	Yes	No
GetOpacityMask	Yes	No
GetParent	Yes	Yes
GetTransform	Yes	No
GetXSnappingGuidelines	Yes	No
GetYSnappingGuidelines	Yes	No
HitTest	Yes	No

The Silverlight method, FindElementsInHostCoordinates, and the WPF method, HitTest, have similar purposes: to find UI elements that touch a specific point. One difference between the two methods is that Silverlight hit tests are performed using the global coordinate system, while WPF uses the local coordinate system of the specified UI element.

To iterate over all the elements under the mouse pointer in a Silverlight application, the following code snippet could be used as a starting point:

C#

```
void Page_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)
{
    IEnumerable<UIElement> elements;
    elements = 
VisualTreeHelper.FindElementsInHostCoordinates(e.GetPosition(null), this);

    foreach(UIElement item in elements)
    {
        // do something.
    }
}
```

VISUAL BASIC

```
Private Sub Page_MouseLeftButtonDown(ByVal sender As Object, _
                                       ByVal e As MouseButtonEventArgs)

   Dim elements As IEnumerable(Of UIElement)

   elements = _
   VisualTreeHelper.FindElementsInHostCoordinates(e.GetPosition(Nothing), Me)

   For Each item As UIElement In elements
       ' do something.
   Next

End Sub
```

The three shared methods allow developers to query the visual tree for a particular UI element and retrieve its children or parent.

CONTROL LIBRARY

The following table outlines the various controls provided to WPF and Silverlight, as well as the location where they can be downloaded if not part of the default installation.

Table taken from http://msdn.microsoft.com/en-us/library/cc903925(VS.95).aspx).

In the case of Silverlight, some controls are available outside the default installation; such as from the Silverlight Toolkit (Toolkit) available at http://www.codeplex.com/silverlight, or the Silverlight SDK (SDK). The Silverlight Toolkit is updated regularly, so checking the web site on a periodic basis is recommended.

Control	WPF	Silverlight
AccessText	Yes	No
AdornedElementPlaceholder	Yes	No
AdornerDecorator	Yes	No
AutoCompleteBox	No	Toolkit
Border	Yes	Yes
BulletChrome	Yes	No
BulletDecorator	Yes	No
Button	Yes	Yes

ButtonChrome	Yes	No
Calendar	Yes	SDK
Canvas	Yes	Yes
CheckBox	Yes	Yes
ClassicBorderDecorator	Yes	No
ComboBox	Yes	Yes
ComboBoxItem	Yes	Yes
ContentControl	Yes	Yes
ContentPresenter	Yes	Yes
ContextMenu	Yes	No
Control	Yes	SDK
DataGrid	Yes	SDK
DatePicker	Yes	SDK
Decorator	Yes	No
DockPanel	Yes	Toolkit
DocumentPageView	Yes	No
DocumentReference	Yes	No
DocumentViewer	Yes	No
Ellipse	Yes	SDK
Expander	Yes	Toolkit
FixedPage	Yes	No
FlowDocumentPageViewer	Yes	No
FlowDocumentReader	Yes	No
FlowDocumentScrollViewer	Yes	No
Frame	Yes	No
FrameworkElement	Yes	No
Glyphs	Yes	No
Grid	Yes	Yes
GridSplitter	Yes	SDK
GridViewColumnHeader	Yes	No
GridViewHeaderRowPresenter	Yes	No
GridViewRowPresenter	Yes	No
GroupBox	Yes	No
GroupItem	Yes	No
HeaderedContentControl	Yes	Toolkit
HeaderedItemsControl	Yes	Toolkit
HyperlinkButton	No	Yes
Image	Yes	Yes
ImplicitStyleManager	Yes	Toolkit
InkCanvas	Yes	No
InkPresenter	Yes	Yes
ItemsControl	Yes	Yes
ItemsPresenter	Yes	Yes
Label	Yes	Toolkit
Line	Yes	Yes
ListBox	Yes	Yes

ListBoxChrome	Yes	No
ListBoxItem	Yes	No
ListView	Yes	No
ListViewItem	Yes	No
MediaElement	Yes	Yes
Menu	Yes	No
MenuItem	Yes	No
MultiScaleImage	No	Yes
NavagationWindow	Yes	No
NumericUpDown	No	Toolkit
Page	Yes	No
PageContent	Yes	No
PageFunction	Yes	No
PasswordBox	Yes	Yes
Path	Yes	Yes
Polygon	Yes	Yes
Polyline	Yes	Yes
Popup	Yes	Yes
ProgressBar	Yes	Yes
RadioButton	Yes	Yes
Rectangle	Yes	Yes
RepeatButton	Yes	Yes
ResizeGrip	Yes	No
Ribbon	Yes	No
RibbonWindow	Yes	No
RichTextBox	Yes	No
ScrollBar	Yes	Yes
ScrollChrome	Yes	No
ScrollContentPresenter	Yes	Yes
ScrollViewer	Yes	Yes
Separator	Yes	No
Slider	Yes	Yes
StackPanel	Yes	Yes
StatusBar	Yes	No
StatusBarItem	Yes	No
SystemDropShadowChrome	Yes	No
TabControl	Yes	SDK
TabItem	Yes	SDK
TabPanel	Yes	No
TextBlock	Yes	Yes
TextBox	Yes	Yes
Thumb	Yes	Yes
TickBar	Yes	No
ToggleButton	Yes	Yes
ToolBar	Yes	No
ToolBarOverflowPanel	Yes	No

ToolBarPanel	Yes	No
ToolBarTray	Yes	No
ToolTip	Yes	Yes
Track	Yes	No
TreeView	Yes	Toolkit
TreeViewItem	Yes	Toolkit
UniformGrid	Yes	No
UserControl	Yes	Yes
ViewBox	Yes	Toolkit
Viewport3D	Yes	No
VirtualizingStackPanel	Yes	No
WebBrowser	Yes	No
Window	Yes	No
WindowsFormsHost	Yes	No
WrapPanel	Yes	Toolkit

Note, that even when Silverlight and WPF share a control, they may be implemented differently. For example, in WPF, a Canvas object contain Left, Right, Top, and Bottom dependency properties. In Silverlight, the Canvas object only exposes the Left and Top dependency properties. The Left and Top properties take precedent over the Right and Bottom properties, but worth noting if porting positioning logic from WPF to Silverlight.

General

Routed Events

There are three types of routed events: direct, bubbling and tunneling. Direct events are those events that are only handled by the element creating the event. Bubbling events are those events that travel upward through the visual tree and can be handled by any parent of the source element. Finally, tunneling events are those events that travel downward through the visual tree and can be handled by any child of the source element.

WPF supports all three types of routed events, while Silverlight only supports direct and bubbling events.

Additionally, WPF supports the creation of custom routed events via the EventManager helper class. Silverlight currently does not support custom routed events.

Data-Binding

Data-binding is WPF's and Silverlight's mechanism of associating data to controls. Changes to the source data are propagated to the associated controls according to the binding rules set on them. Data can be bound to just about any property on any UI element.

The following snippet of XAML is a simple example of data binding.

```
<TextBlock Text="{Binding Person, Path=FirstName}" />
```

Binding Modes

There are four binding mode available in WPF: One-Way, Two-Way, One-Time, and One-Way To Source.

- **One-Way** binding causes data to flow from the data source to the target element. Changes to the data source will update the target.
- **Two-Way** binding causes data to flow from the data source to the target element, and from the target element back to the data source. Changes to the data source are reflected in the target element, and updates to the target element are propagated back to the data source.
- **One-Time** binding cause data to flow from the data source to the target element. However, the binding does not listen to change notifications, and any changes in the data are not updated in the target element.
- **One-Way To Source** binding causes data to flow from the target element to the data source. Changes in the data source are not propagated to the target element, but changes in the target element are flowed back to the data source.

Silverlight does not support the One-Way To Source binding mode.

Additionally, the default binding mode, if none is explicitly set, is different between the two platforms. WPF's default binding mode is dependent upon the dependency property, while Silverlight's default mode is always OneWay.

UpdateSourceTrigger

The UpdateSourceTrigger property of the data binding allows the developer to control when the data binding occurs. There are three types of UpdateSourceTriggers:

- **LostFocus** causes the data-binding when the target element loses focus. For example, tabbing out of a TextBox will cause the data-binding to occur.
- **PropertyChanged** causes the data-binding to occur whenever the data changes. For example, typing each key in TextBox will cause the binding to occur.
- **Explicit** trigger will only cause the binding to occur when the UpdateSource method is called on the binding.

Silverlight does not support Explicit data-binding. In order to force a data-binding refresh, the property changed event must be fired in the data source, or the UI element must be forced to lose focus.

Multi Binding

Data binding maps a single data element to a single property of a UI element. MultiBinding allows binding multiple data elements to a single property of a UI element. MultiBinding, unlike regular binding, requires the use of a ValueConverter, or more specifically, a MultiValueConverter.

To demonstrate the MultiValueConverter, a simple, contrived, example is needed. The XAML below declares a TextBlock with a MultiValueConverter and two bindings:

```
<TextBlock>
   <TextBlock.Text>
      <MultiBinding Converter="{StaticResource NameConverter}">
         <Binding Source="{StaticResource Person}" Path="FirstName" />
         <Binding Source="{StaticResource Person}" Path="LastName" />
      </MultiBinding>
   </TextBlock.Text>
</TextBlock>
```

And the corresponding MultiValueConverter:

C#

```
public class NameConverter : IMultiValueConverter
{
   public object Convert(object[] values, Type targetType, object parameter,
            CultureInfo culture)
   {
      // Assuming first and last names are passed in.
      return string.Concat(values[0], " ", values[1]);
   }

   // ConvertBack method snipped for brevity.
   …
}
```

Visual Basic

```
Public Class NameConverter
   Implements IMultiValueConverter

   Public Function Convert(ByVal values() As Object, _
      ByVal targetType As System.Type, _
      ByVal parameter As Object, _
      ByVal culture As System.Globalization.CultureInfo) As Object _
      Implements System.Windows.Data.IMultiValueConverter.Convert

      ' Assuming first and last names are passed in.
      Return String.Concat(values(0), " ", values(1))

   End Function

   ' ConvertBack method snipped for brevity.
   …
End Class
```

It would be remiss not to mention a simpler solution to this problem in WPF. With Service Pack 1 of WPF 3.5, the StringFormat property has been added to bindings. The above XAML could be rewritten, without the use of a MultiValueConverter, as:

```
<TextBlock>
   <TextBlock.Text>
      <MultiBinding StringFormat="{}{0} {1}">
         <Binding Source="{StaticResource Person}" Path="FirstName" />
         <Binding Source="{StaticResource Person}" Path="LastName" />
      </MultiBinding>
   </TextBlock.Text>
</TextBlock>
```

WPF supports multiple data-binding while Silverlight only allows binding to single value. However, a viable work-around in Silverlight would be to create a custom class to aggregate the multiple data elements into a single property and either parse them in the Value Converter or use the composite value directly.

For example,

C#

```
public class Person : INotifyPropertyChanged
   {
   private string _firstName;
   private string _lastName;

   public event PropertyChangedEventHandler PropertyChanged;

   public string FirstName
   {
      get { return _firstName; }
      set
      {
         _firstName = value;
         NotifyPropertyChanged("FirstName");
         NotifyPropertyChanged("FullName");
      }
   }

   public string LastName
   {
      get { return _lastName; }
      set
      {
         _lastName = value;
         NotifyPropertyChanged("LastName");
         NotifyPropertyChanged("FullName");
      }
   }

   public string FullName
   {
        get { return _firstName + " " + _lastName; }
    }

    private void NotifyPropertyChanged(string propertyName)
    {
        if (PropertyChanged != null)
        {
            try
            {
                PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
            catch (Exception ex)
            {
                // Error handing snipped.
            }
        }
    }
}
```

VISUAL BASIC

```
Public Class Person
    Implements INotifyPropertyChanged

    Private _firstName As String
    Private _lastName As String

    Public Event PropertyChanged As PropertyChangedEventHandler _
        Implements INotifyPropertyChanged.PropertyChanged

    Public Property FirstName() As String
        Get
            Return _firstName
        End Get
        Set(ByVal value As String)
            _firstName = value
            NotifyPropertyChanged("FirstName")
            NotifyPropertyChanged("FullName")
        End Set
    End Property

    Public Property LastName() As String
        Get
            Return _lastName
        End Get
        Set(ByVal value As String)
            _lastName = value
            NotifyPropertyChanged("LastName")
            NotifyPropertyChanged("FullName")
        End Set
    End Property

    Public ReadOnly Property FullName() As String
        Get
            Return _firstName & " " & _lastName
        End Get
    End Property

    Private Sub NotifyPropertyChanged(ByVal propertyName As String)
        Try
        RaiseEvent PropertyChanged(Me, _
            New PropertyChangedEventArgs(propertyName))

        Catch ex As Exception
            ' Omitting error handling for brevity.
        End Try
    End Sub
End Class
```

By binding on FullName, Silverlight will receive a change notification whenever either the FirstName or LastName properties change.

Element Binding

WPF supports binding to other elements within the visual tree. The following is perfectly legal in a WPF application:

```
<TextBox x:Name="Field1" />
<TextBlock Text="{Binding ElementName=Field1, Path=Text}" />
```

Silverlight 3 Note

Element-to-element binding will be supported in Silverlight 3.

Data Providers

While the source object can be any .Net object, WPF provides some specialized data providers to enhancing the binding experience. The XmlDataProvider and the ObjectDataProvider are two such data providers. Silverlight, on the other hand, does not provide an implementation of the ObjectDataProvider.

However, since the source object can be any .Net object, programming around the deficiency is not difficult in Silverlight. The lost functionality, such as binding to a method, can be achieved by wrapping the method in a property and implementing the INotifyPropertyChanged interface judiciously.

Networking

Communications

Silverlight does not have the capability of making a synchronous networking request due to the potential of blocking the main UI thread. Silverlight can only make asynchronous network calls.

WPF has access to the complete networking stack within the .Net Framework and does not have this restriction.

In addition to HTTP communications, Silverlight also can communicate to network resources via socket programming. Socket development provides interfaces for developers who require a more tightly controlled access to the network.

HTTP Communications

Support for HTTP communication opens the door to several options within Silverlight; web services, WCF Services, and REST. Note that Silverlight is still bound to the supported bindings listed below in the BINDINGS section.

SOAP

Silverlight supports SOAP 1.1 over HTTP, and does not support other versions of SOAP, or other web services not compliant with WS-I Basic Profile 1.0.

Silverlight only supports textual XML encoding, but not binary encoding.

Silverlight 3 Note
Silverlight 3 will support binary encoding

WPF can support all SOAP versions and multiple encoding types.

Fault Handling

Silverlight cannot currently natively handle SOAP Fault Exceptions due to web browser limitations. When a fault does occur, an exception is thrown, but does not specify any detail about the fault. In essence, Silverlight can determine a fault happened, but not why it happened.

Silverlight 3 Note
Silverlight 3 will implement a solution to provide more detailed information about SOAP exceptions.

Concurrent Connections

Silverlight is bound to the browsers networking stack and is limited to its concurrent connection settings. The most restrictive of these limits is two concurrent connections, but is dependent on the system's maximum network connections. This setting can be changed in the registry under the following key:

```
HKEY_LOCAL_MACHINE/ (or HKEY_CURRENT_USER)
SOFTWARE/
Microsoft/
Internet Explorer/
MAIN/
FeatureControl/
FEATURE_MAXCONNECTIONSPERSERVER
```

WPF has no such restriction and is free to open as many concurrent connections as needed within system limitations.

Bindings

Network bindings define the methodology an application uses to connect to network resources.

The System.ServiceModel namespace contains several different types of network bindings.

Binding Class	WPF	Silverlight
BasicHttpBinding	Yes	Yes
BasicHttpContextBinding	Yes	No
NetMsmqBinding	Yes	No
NetNamedPipeBinding	Yes	No
NetPeerTcpBinding	Yes	No
NetTcpBinding	Yes	No
NetTcpContextBinding	Yes	No
PollingDuplexHttpBinding	Yes	Yes
WebHttpBinding	Yes	No
WS2007FederationHttpBinding	Yes	No
WS2007HttpBinding	Yes	No
WSDualHttpBinding	Yes	No
WSFederationHttpBinding	Yes	No
WSHttpBinding	Yes	No
WSHttpContextBinding	Yes	No

BasicHttpBinding uses HTTP as the transport for communicating through Windows Communication Foundation (WCF) endpoints to access ASMX-based Web services or other services conforming to the WS-I Basic Profile 1.1.

PollingDuplexHttpBinding is similar to BasicHttpBinding, except that the WCF services are configured for duplex communication with a polling client.

Local Isolated Storage

There are some significant changes in the implementation of local isolated storage between Silverlight and WPF. There are two classes shared between WPF and Silverlight involving isolated storage: IsolatedStorageFile and IsolatedStorageFileStream. Both classes have significantly different implementations, and in addition, the Silverlight implementation marks most of the methods as SecuritySafeCritical or SecurityCritical.

Silverlight also implements the concept of a quota, prohibiting Silverlight applications from consuming excessive amounts of physical disk space. By default, a Silverlight application is allowed to consume up to 1 MB of storage. Applications may increase their quota after receiving approval from the user.

ISOLATEDSTORAGEFILE

Method	WPF	Silverlight
Close()	Yes	Yes
CreateDirectory()	Yes	Yes

CreateFile()	No	Yes
DeleteDirectory()	Yes	Yes
DeleteFile()	Yes	Yes
Dispose()	Yes	Yes
FileExists()	No	Yes
GetDirectoryNames()	Yes	Yes
GetFileNames()	Yes	Yes
GetMachineStoreForApplication()	Yes	No
GetMachineStoreForAssembly()	Yes	No
GetMachineStoreForDomain()	Yes	No
GetStore()	Yes	No
GetUserStoreForApplication()	Yes	Yes
GetUserStoreForAssembly()	Yes	No
GetUserStoreForDomain()	Yes	No
GetUserStoreForSite()	No	Yes
IncreaseQuotaTo	No	Yes
OpenFile()	No	Yes
Remove()	Yes	Yes
Property		
AvailableFreeSpace	No	Yes
CurrentSize	Yes	No
MaximumSize	Yes	No
Quota	No	Yes

IsolatedStorageFileStream

Method	**WPF**	**Silverlight**
BeginRead()	Yes	Yes
BeginWrite()	Yes	Yes
EndRead()	Yes	Yes
EndWrite()	Yes	Yes
Flush()	Yes	Yes
Read()	Yes	Yes
ReadByte()	Yes	Yes
Seek()	Yes	Yes
SetLength()	Yes	Yes
Write()	Yes	Yes
WriteByte()	Yes	Yes
Property		
CanRead	Yes	Yes
CanSeek	Yes	Yes
CanWrite	Yes	Yes
IsAsync	Yes	No
Length	Yes	Yes
Position	Yes	Yes
SafeFileHandle	Yes	No

While virtually identical classes, there are significant differences in the constructors for IsolatedStorageFileStream between the two platforms. WPF provides for eight overloaded constructors, while Silverlight supports only three of those.

Constructor	Silverlight
IsolatedStorageFileStream(String, FileMode)	No
IsolatedStorageFileStream(String, FileMode, FileAccess)	No
IsolatedStorageFileStream(String, FileMode, IsolatedStorageFile)	Yes
IsolatedStorageFileStream(String, FileMode, FileAccess, FileShare)	No
IsolatedStorageFileStream(String, FileMode, FileAccess, IsolatedStorageFile)	Yes
IsolatedStorageFileStream(String, FileMode, FileAccess, FileShare, Int32)	No
IsolatedStorageFileStream(String, FileMode, FileAccess, FileShare, IsolatedStorageFile)	Yes
IsolatedStorageFileStream(String, FileMode, FileAccess, FileShare, Int32, IsolatedStorageFile)	No

Resource Dictionaries

Both Silverlight and WPF support resource dictionaries. Resource dictionaries can exist in multiple places throughout the application; from the application itself to individual controls.

Resources in WPF can be either static or dynamic. Dynamic resources are not available in Silverlight. Dynamic resources are resources that are evaluated each time they are accessed and are discussed in the WPF-only implementation section.

Referencing resources from code also differs slightly between Silverlight and WPF. In both platforms, the resource can be retrieved by its key from the appropriate resource dictionary (either the immediate resource dictionary or the application resource dictionary).

WPF, however, also supports the FindResource() method to search for a resource key wherever it may exist. In addition, the WPF implementation supports IEnumerable to allow iterating over the collection of resources.

Merged Dictionaries

WPF supports external resource files and the ability to merge them into one cohesive resource dictionary.

```
<ResourceDictionary>
   <ResourceDictionary.MergedDictionaries>
      <ResourceDictionary Source="MyResources1.xaml" />
      <ResourceDictionary Source="MyResources2.xaml" />
```

```
    </ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
```

Silverlight, unfortunately, does not currently support merged resource dictionaries. However, it is possible to craft a similar solution to dynamically populate the application resource collection at runtime.

Silverlight 3 Note
The next version of Silverlight will support merged dictionaries in the same way WPF currently does.

Other Differences

In WPF, the FrameworkTemplate base class used for templating, contains a Resources collection property. Silverlight's implementation of the FrameworkTemplate class does not have a Resources property.

Silverlight does not support x:Shared or x:Static attributes. In addition, Silverlight has a smaller range of shareable types than WPF. Shareable objects are the only types of objects that can be stored within a resource dictionary.

Custom Dependency Properties

Coerce-value Callback

Silverlight does not support WPF's implementation of coerce-value callbacks and DependencyObject.CoerceValue. However, since Silverlight supports property-changed callbacks, most of the missing implementation can be added.

Read-Only Properties

Silverlight does not support read-only dependency properties. Unfortunately, there is no fool-proof method for working around this limitation since any external code can call set the property through the use of the SetValue method.

Property Metadata

Overriding metadata is not supported in Silverlight; a dependency property's metadata is set as initially registered to the owner type and cannot change.

Silverlight only supports the PropertyMetadata class, whereas WPF implements several derived classes (such as UIPropertyMetadata and FrameworkPropertyMetadata). The PropertyMetadata class is not sealed, so it is possible to derive from it to implement any missing functionality.

Once constructed, Silverlight's PropertyMetadata is not as useful as the version implemented in WPF due to a lack of readable properties. Additionally, API functions such as DependencyProperty.GetMetadata and DependencyObjectType class are simply missing from Silverlight.

AddOwner

The AddOwner mechanism is particularly useful in WPF when attempting to change the PropertyMetadata of a dependency property defined in an existing class. AddOwner allows the re-registering of a dependency property to a new owner. Silverlight does not support the AddOwner method on the DependencyObject class.

Commanding

Commanding is an input mechanism introduced in WPF to provide a separation between UI elements and the actions taken. Silverlight does not support the concept of commanding as defined in WPF, it only supports the ICommand interface. Some third-party projects, both open source and off-the-shelf libraries and used the interface to provide rudimentary commanding to Silverlight.

.NET FRAMEWORK

Silverlight contains a subset of the .Net framework, called CoreCLR, and there are noticeable differences in implementation of even the most mundane functionality. Producing an exhaustive list of framework differences would be a daunting task.

Take, for example, the simple String object and the Split() method.

String.Split() Overloads	WPF	Silverlight
Split(params char[] separator)	Yes	Yes
Split(char[] separator, int count)	Yes	No
Split(char[] separator, StringSplitOptions options);	Yes	Yes
Split(string[] separator, StringSplitOptions options);	Yes	Yes
Split(char[] separator, int count, StringSplitOptions options);	Yes	No
Split(string[] separator, int count, StringSplitOptions options);	Yes	No

Silverlight only supports half the overloaded methods as WPF. These types of differences can be found throughout the entire .Net Framework.

Collections

Looking at the System.Collections and System.Collections.Generic namespaces, several differences between the implementations of both platforms become apparent.

SYSTEM.COLLECTIONS

Classes	WPF	Silverlight
ArrayList	Yes	No
BitArray	Yes	Yes
HashTable	Yes	No

Queue	Yes	No
SortedList	Yes	No
Stack	Yes	No

Typically, this should not be a problem since most of these collections are available in a generic form. However, use of these classes can be problematic when porting code from WPF to Silverlight.

SYSTEM.COLLECTIONS.GENERIC

Classes	WPF	Silverlight
Dictionary<TKey, TValue>	Yes	Yes
KeyedByTypeCollection<TItem>	Yes	Yes
HashSet<T>	Yes	No
LinkedList<T>	Yes	Yes
List<T>	Yes	Yes
Queue<T>	Yes	Yes
SortedDictionary<TKey, TValue>	Yes	No
SortedList<T>	Yes	No
Stack<T>	Yes	Yes
SynchronizedCollection<T>	Yes	No

Unfortunately, Silverlight doesn't support the highly efficient HashSet<T>; other data structures would need to be used as Silverlight doesn't support the HashTable collection either.

The SortedDictionary and SortedList omissions can easily be worked around using LINQ.

C#

```
var q = from key in MyDictionary.Keys
        orderby key ascending
        select MyDictionary[key];
```

VISUAL BASIC

```
Dim q = From key In MyDictionary.Keys _
        Order By key Ascending _
        Select MyDictionary(key)
```

Cryptography

Like most aspects of Silverlight, cryptography support is a subset of the WPF implementation. Comparing the classes in System.Security.Cryptography namespaces in both platforms would not be especially helpful (there are 107 classes in the namespace for WPF, and only 20 for Silverlight).

Silverlight only supports the AES encryption algorithm. WPF supports the full range of algorithms in the .Net Framework.

Silverlight supports four hashing algorithms: SHA1, SHA1Managed, SHA256, and SHA256Managed. WPF supports all the .Net hashing algorithms.

Threading

Interprocess Synchronization

WPF supports several methods for synchronizing threads between processes; such as the Monitor, Mutex or Semaphore classes. Silverlight, however, only supports the Monitor class for synchronization.

Locking

WPF implements support for the ReaderWriteLock class, enabling locks on resources to support single writers and multiple readers. Silverlight supports locking through the Monitor class, which grants locks to an object for a single thread. The Monitor class can ensure that no other thread is allowed to access a code section except by that of the lock owner. Though a bit coarser than the locks available to WPF, the Monitor should prove to be sufficient for most multi-threaded needs in Silverlight.

WPF Specific Functionality

WPF applications are primarily desktop applications running within the confines of a Windows operating system. As such, WPF applications do not have the restrictions imposed on Silverlight which runs in a tightly controlled sandbox within the confines of a web browser.

Desktop Functionality

Printing

Unlike printing in the WinForms era, printing in WPF is relatively straight-forward. Any visual element can be printed, but the most likely scenario would be printing an entire Window.

The following snippet will print the contents of a UIElement using the OS printer dialog window.

C#

```
PrintDialog dialog = new PrintDialog();

if (dialog.ShowDialog() == true)
{
    dialog.PrintVisual(this.MyGrid, "My Grid");
}
```

Visual Basic

```
Dim dialog As New PrintDialog

If dialog.ShowDialog() = True Then
       dialog.PrintVisual(Me.MyGrid, "My Grid")
End If
```

This basic printing has limitations: if the size of the visual is larger than an actual page, the content will get clipped.

For more complex printing needs, such as multiple page printing, WPF offers the PrintDocument method. Instead of printing a Visual element, the PrintDocument accepts a DocumentPaginator object. The DocumentPaginator is an abstract class, so developers will have to define a class to perform the more complex functionality of multiple-page printing.

XPS Documents

XML Paper Specification (XPS) documents maintain a consistent appearance across all display mediums. WPF has full support for creating, viewing and printing XPS documents through the XpsDocumentWriter and XpsViewer classes in the System.Windows.Xps namespace.

There are two types of documents, FlowDocument and FixedDocument. FlowDocuments can adjust its content to make use of the space within the page.

A simple FlowDocument looks like:

```xml
<FlowDocument>
    <Paragraph>
      Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
      eiusmod tempor incididunt ut labore et dolore magna aliqua.
    </Paragraph>
    <Paragraph>
      Sed ut perspiciatis unde omnis iste natus error sit voluptatem
      accusantium doloremque laudantium, totam rem aperiam, eaque ipsa
      quae ab illo inventore veritatis et quasi architecto beatae vitae
      dicta sunt explicabo.
    </Paragraph>
</FlowDocument>
```

A Document consists of two main TextElements, blocks and inlines. WPF implements five types of block elements:

- **Paragraph**
  Contains a collection inline elements representing the content of a paragraph. By default, any content placed between the start and end tags automatically become the content of a Run inline element and added to the Paragraph's Inline collection.

- **Section**
  Groups block element together without changing the physical structure. This is useful when multiple blocks share a common look, such as a background color.

- **List**
  Supports creating bulleted, numbered, or plain lists. Contains a collection of ListItem elements.

- **Table**
  Supports placing elements into columns and rows.

- **BlockUIContainer**
  Holds any UIElelement to allow embedding of virtually any WPF visual element.

Inline elements mirror virtually all the inline elements within HTML: spans, bold, italic, underline, and hyperlink to name a few.

Displaying Documents

WPF provides three controls for displaying documents.

- **FlowDocumentScrollViewer**
  Displays the document as one continuous block with vertical and/or horizontal scrolling

available.

- **FlowDocumentPageViewer**
  Shows the document in individual pages and supports zooming.

- **FlowDocumentReader**
  Supports the full-range of viewing capabilities, including zooming, and text searching, alternating between scrolling and paging.

Annotations

In addition to displaying documents, WPF implements an annotation system for adding textual notes or highlights to an existing document without changing the original document. The one downside to the annotating service is that WPF does not expose any user interface for adding annotations or highlighting; the developer needs to create their own UI and tie it to the annotation service APIs.

Speech

WPF supports the speech APIs found in the System.Speech namespace, which in turn, are wrappers around the Microsoft SAPI framework, a Win32 library. However, the speech API is not actually WPF technology: no dependency properties, routed events, or other aspects that make WPF what it is. In fact, there is no XAML syntax for speech; instead speech relies on a different XML dialect called Speech Synthesis Markup Language (SSML).

There are two aspects to the speech APIs: synthesis (speaking text) and recognition (listening to commands). The System.Speech namespace provides classes for both concepts, but recognition requires a speech recognition engine to be installed. Fortunately, Windows Vista has one by default (as well as Office XP or later).

Speech Synthesis

The most important class for converting text to speech is the SpeechSynthesizer class. Here is a simple example:

C#

```
SpeechSynthesizer speaker = new SpeechSynthesizer();
speaker.Speak("Text to speech from WPF.");
```

VISUAL BASIC

```
Dim speaker As New SpeechSynthesizer
speaker.Speak("Text to speech from WPF.")
```

Speech Recognition

Speech recognition begins with the SpeechRecognizer class, which exposes the SpeechRecognized event. Spoken words are converted to text and set as part of the event arguments. This method is ideal for converting spoken words into text to be stored in a text box.

Supporting commands requires more work, such as defining a grammar dictionary in the Speech Recognition Grammar Specification (SRGS) XML dialect, or by using the SrgsGrammar helper class. The SrgsGrammar class is not as fully functional as creating a SRGS document.

InterOp

Interoperability in WPF generally refers to interacting with previous technology such as Win32 applications, COM+, and ActiveX controls.

WindowsFormsHost

The WindowsFormsHost control allows a developer to embed a WinForms control directly into a WPF application with little work. However, the WindowsFormsHost control cannot overlap or be overlapped by any WPF content. Each region in an application can only be controlled by one display technology.

Hosting a Windows Forms NumericUpDown control within a WPF application:

```
<WindowsFormsHost>
      <winforms:NumericUpDown x:Name="myUpDownControl" />
 </WindowsFormsHost>
```

ElementHost

ElementHost is the Windows Forms equivalent of WindowsFormsHost, and allows embedding of WPF elements within a WinForms application. Of course, in WinForms, the work must be done in code instead of declarative XAML.

ActiveX Controls

Embedding an ActiveX control in a WPF application uses the same methodology as embedding a Windows Forms control; that is, via the WindowsFormsHost element. However, the reverse, exposing a WPF control as an ActiveX control is not currently supported.

Advanced InterOp

More complex interoperability may be performed in WPF through the knowledge of window handles (HWNDs). While most of WPF does not implement any method for interacting with HWNDs, the HwndHost, a subclass of FrameworkElement, class can host any object exposing an HWND. In addition, WPF defines an HwndSource to expose any visual element as a HWND for consumption in a Win32 application.

XAML

WPF supports some additional XAML elements and attributes than Silverlight. Styles are one example where WPF adds additional functionality. Note, Microsoft is working to implement more and more of WPF within the Silverlight platform.

Styles

Styles in WPF are analogous to HTML's cascading style sheets, in that they allow changing the appearance of any visual element from a single location; essentially abstracting the look of the control from its functionality. While both WPF and Silverlight can use styles, WPF implements additional functionality with regard to styles.

BasedOn Styles

The BasedOn property allows a form of visual inheritance between styles. As the name states, a developer can base one style off of another.

```
<Style
   x:Key="DefaultText"
   TargetType="TextBlock">

      <Setter Property="FontFamily" Value="Segoe UI"/>
      <Setter Property="FontSize" Value="11" />
      <Setter Property="Foreground" Value="Black" />
</Style>

<Style
   x:Key="EmphasizedText"
   TargetType="TextBlock"
   BasedOn="{StaticResource DefaultText}">
      <Setter Property="Foreground" Value="Red" />
      <Setter Property="FontWeight" Value="Bold" />
</Style>
```

Silverlight 3 Note

Based-On styles are supported by Silverlight 3, as well as dynamic styles that allow styles to be changed on elements after they have been set.

Implicit Styles

Implicit styles are styles defined without a key, such as:

```
<Style TargetType="{x:Type TextBlock}">
    <Setter Property="FontFamily" Value="Segoe UI"/>
    <Setter Property="FontSize" Value="15" />
    <Setter Property="Foreground" Value="Black" />
</Style>
```

An implicit style will set the properties of any visual element matching the TargetType. This is an easy method for updating the look-and-feel of an entire application without having to add Style declarations to every control.

In some cases, it might be necessary to remove an implicit style on a visual element. To negate an implicit style, simply set the Style property of the element to {x:Null}.

USER INTERFACE

3D Graphics

WPF supports hardware accelerated 3D functionality within the WPF graphics system. All 3D content is viewed through a Viewport3D container, and can be completely declared from within XAML.

WPF supports many 3D features including cameras, models, meshes, materials, lights, transformations and animation.

Silverlight 3 Note
Silverlight 3 gains some 3D features, such as perspective projections in 3D for scaling and rotational effects. In addition, Silverlight 3 will support hardware acceleration for many effects for which it previously used software rendering.

However, it will not support the full range of 3D capabilities found in WPF.

Themes

Themes are a set of visual styles and control templates available to WPF applications and are not necessarily application-specific. WPF installs several OS-specific themes such as “Windows Classic,” “Luna”, and “Aero” to name a few. In addition, additional themes may be found for free and for sale on the internet. One free source can be found at http://wpf.codeplex.com, containing nine professional themes available in both WPF and Silverlight platforms (separately through the Silverlight Toolkit).

In practicality, themes are defined as resource dictionaries containing implicit styles and control templates for as many elements as necessary to define the theme. In most cases, virtually every visual control would be styled in the theme.

Themes can be created and placed within the themes subfolder off the root of the project. Theme resource dictionaries also have a distinct naming convention in the form of ThemeName.ThemeColor.xaml. The fallback theme is always named themes\generic.xaml.
To add statically add a theme to a project; simply include the following XAML in the application’s XAML file:

```
<Application.Resources>
```

```
   <ResourceDictionary>
      <ResourceDictionary.MergedDictionaries>
         <ResourceDictionary Source="themes\custom.blue.xaml"/>
      </ResourceDictionary.MergedDictionaries>
   </ResourceDictionary>
</Application.Resources>
```

Skins

Skins are almost identical to themes, except that skins are geared to a specific application while themes are system oriented. Different applications can share system-wide themes, while an individual application may support additional skins.

Applications are usually designed to be skinned by end-users and therefore provide a skinning API for designers. The only major difference is that skin resource files typically name styles explicitly, whereas themes normally use implicit naming. However, this is not a hard and fast rule.

Distinguishing between themes and skins is a nebulous area at best. The current line of thinking is that themes refer to the look and feel of an application to match the operating system, and skins refer to changing an applications appearance to suit the user.

Image Manipulation

WPF implements common imaging capabilities found with the Microsoft GDI+ library, as well overcoming some of the short-comings of that aging library. WPF supports both managed and unmanaged image manipulation libraries. However, this document will focus on the managed library found within the Systme.Windows.Media and System.Windows.Media.Imaging namespaces.

WPF Image Formats

WPF supports the following bitmap (raster) image types:

- BMP
- JPEG
- PNG
- TIFF
- Windows Media Photo
- GIF
- Icon

Both encoders (writers) and decoder (readers) are available for each of the bitmap image types. The basic building block of WPF imaging is the BitmapSource class. It can represent not only a single image, but multiple frames in GIF or TIFF images (both formats support multiple frames).

The following snippet shows to save a Visual element, named “MyGrid” in this example, to a PNG image file sized at 800x600 pixels at the default 96 dots per inch:

C#

```
string fileName = "test.png";
RenderTargetBitmap target = new RenderTargetBitmap(800, 600, 96, 96,
PixelFormats.Default);

target.Render(this.MyGrid);
BitmapEncoder encoder = new PngBitmapEncoder();
encoder.Frames.Add(BitmapFrame.Create(target));

// Save image to disk
FileStream fs = File.Open(fileName, FileMode.Create);
encoder.Save(fs);
fs.Close();
```

VISUAL BASIC

```
Dim fileName As String = "test.png"
Dim target As New RenderTargetBitmap(800, 600, 96, 96, PixelFormats.Default)

target.Render(Me.MyGrid)
Dim encoder As New PngBitmapEncoder()
encoder.Frames.Add(BitmapFrame.Create(target))

' Save image to disk
Dim fs As FileStream
fs = File.Open(fileName, FileMode.Create)
encoder.Save(fs)
fs.Close()
```

Silverlight 3 Note
A WriteableBitmap class has been added to Silverlight 3 that implements an API for creating bitmaps programmatically. With the new SaveFileDialog class, creating and storing images for later retrieval will now be possible.

Page-Based Navigation

Traditional client development centers on windows and forms and WPF has no issues working in that paradigm. However, the internet brought the web page, and navigational-based web applications. WPF attempts to bridge the divide by bringing the page-based navigation metaphor to the desktop.

Applications using page-based navigation typically start with the NavigationWindow element, instead of a Window element. Individual screens are defined through the Page element. In addition, WPF supports a Frame element that may also contain Pages for navigation purposes, but embedded within a screen instead of being the top-level window.

The majority of paging interaction occurs through the NavigationService instance exposed by the Page element, or through a static method on the NavigationService class. The Navigate() method is overloaded to accept instances of Pages as well as URIs, relative or external.

Silverlight 3 Note
Silverlight 3 will contain a new NavigationFramework that will function similarly to the page-based navigation implementation in WPF, as well as take advantage of the browser's Back and Next buttons.

Journal

The journal is an aspect of the page-based navigation system to provide history of pages visited. NavigationWindows implement a journal by default, while Frames need to request one through the JournalOwnership property.

Pages can opt out of being stored in the journal by setting the RemoveFromJournal property to true.

Input Gestures

While the term gestures may suggest some fancy high-tech method for analyzing mouse movements, the reality is much simpler. Input gestures are a facility for defining a sequence of key-strokes, such as Alt-C or Ctrl-Alt-F2 or mouse button clicks such as Alt-Left Click.

Input gestures work in conjunction with WPF commands. Commands can be associated with one or more input gestures.

Input gestures can be declared in XAML as well as in code. For example, to attach the F1 key to the help command or CTRL-ALT-F2 sequence to a print command, a developer could code in XAML:

```xml
<Window.InputBindings>
  <!-- Single Key Binding -->
  <KeyBinding Command="Help" Key="F1" />

  <!-- Multi-Key Binding -->
  <KeyBinding Command="Print" Gesture="Control+Alt+F2" />

</Window.InputBindings>
```

Mouse gestures are defined in the same manner:

```xml
<Window.InputBindings>

  <MouseBinding Command="CustomCommand" Gesture="Alt+LeftClick" />

</Window.InputBindings>
```

GENERAL FUNCTIONALITY

Freezable Objects

A freezable object in WPF is an object deriving from the Freezable base class. It is special in that when frozen, the object can no longer be modified. Once frozen, the object can then be shared across threads and processes; non-frozen objects cannot be shared in this manner.
Brushes, Transforms, and Geometry objects are all freezable objects; in fact the majority of freezable objects in WPF are contained within the graphics libraries. When frozen, performance will improve due to the way these objects have been implemented. A large portion of the graphics sub-system internally implements unmanaged code, requiring the system to monitor these objects for changes. When frozen, the system no longer needs to monitor the objects for changes and can make optimizations based on this fact.

Once frozen, a freezable object cannot be unfrozen; however, it can be cloned. In addition, not all freezable objects can be frozen and therefore the base class exposes a CanFreeze property.

Finally, freezing an object that contains freezable objects will also freeze all the children. For example, freezing an object deriving from Geometry will also freeze all the Figures and Segments contained within it.

Data Providers

WPF implements a data provider model for data binding and ships with two concrete data provider classes: ObjectDataProvider and XmlDataProvider. In addition, it is possible to create custom data providers by deriving from System.Data.DataSourceProvider.

XmlDataProvider

The XmlDataProvider can provide access to an embedded XML data structure or a complete XML file. An XML data structure can be embedded into an applications resource dictionary as follows:

```
<Window.Resources>
  <XmlDataProvider x:Key="MyData" XPath="Teams">
    <x:XData>
        <Teams xmlns="">
            <Team Name="NY Yankees">
                <Players>
                    <Player>
                        <Name>Derek Jeter</Name>
                        <Position>Shortstop</Position>
                    </Player>
                    <Player>
                        <Name>Mariano Rivera</Name>
                        <Position>Closer</Position>
                    </Player>
                </Players>
            </Team>
        </Teams>
    </x:XData>
  </XmlDataProvider>
</Window.Resources>
```

To load data from an external XML data file,

```
<Window.Resources>
  <XmlDataProvider x:Key="MyData" XPath="Teams" Source="Baseball.xml">
</Window.Resources>
```

To bind to this Xml snippet, use standard binding syntax with the XPath set appropriately.

```
<TextBlock
      Text="{Binding Source={StaticResource MyData}, XPath=Team/@Name}" />
```

ObjectDataProvider

The XmlDataProvider may not be suitable for all purposes, especially when data may be already stored in custom objects. The ObjectDataProvider wraps any .Net object as a data source. Though most .Net objects can already be set as a data source, the ObjectDataProvider provides additional functionality; such as exposing methods as a data source, allowing parameterized construction of the object, and supporting additional asynchronous binding.

```
<Window.Resources>

  <!-- Adding an object as a Resource. -->
  <local:BaseballTeams x:Key="Teams" />

  <!-- Wrapping object inside an ObjectDataProvider -->
  <ObjectDataProvider
      x:Key="MyData" ObjectType="{x:Type local:BaseballTeams}" />

  <!-- Wrapping object inside an ObjectDataProvider w/ constructor -->
  <ObjectDataProvider
      x:Key="Yankees" ObjectType="{x:Type local:BaseballTeams}">
      <ObjectDataProvider.ConstructorParameters>
            <sys:String>NYY</sys:String>
      </ObjectDataProvider.ConstructorParameters>
  </ObjectDataProvider>
```

Validation

WPF provides a framework for adding validation rules to data bindings. The Binding class contains a collection of ValidationRules in which one or more objects deriving from ValidationRule can be loaded. Each rule is checked in turn, therefore, all the rules must pass or the validation fails. Additionally, the validation engine can also be configured to handle exceptions that may arise in the data source itself by using the ExceptionValidationRule.

To create a custom validation rule, simply derive from ValidationRule and add any custom logic to the Validate() method. Then, add the rule to the ValidationRules collection either in code or declaratively in XAML. For example,

C#

```
public class EvenValidationRule : ValidationRule
{
    public override ValidationResult Validate(object value, CultureInfo
cultureInfo)
    {
        // Error checking omitted for brevity.
        int number = int.Parse(value.ToString());

        if (number % 2 != 0)
        {
            return new ValidationResult(false, "Number must be even.");
        }

        return new ValidationResult(true, null);
    }
}
```

VISUAL BASIC

```
Public Class EvenValidationRule
    Inherits ValidationRule

    Public Overrides Function Validate(ByVal value As Object, _
             ByVal cultureInfo As CultureInfo) As ValidationResult

        ' Error checking omitted for brevity

        Dim number As Integer = Integer.Parse(value.ToString())

        If number Mod 2 <> 0 Then
            Return New ValidationResult(False, "Number must be even.")
        End If

        Return New ValidationResult(True, Nothing)
    End Function
End Class
```

```
<TextBox>
    <TextBox.Text>
        <Binding Source="{StaticResource MyData}" XPath="Team/@Score">
            <Binding.ValidationRules>
                <local:EvenValidationRule />
            </Binding.ValidationRules>
        </Binding>
    </TextBox.Text>
</TextBox>
```

By default, when an element fails its validation, WPF outlines the element with a thin red border. However, by providing an ErrorTemplate, a designer can change the look-and-feel of fields failing validation.

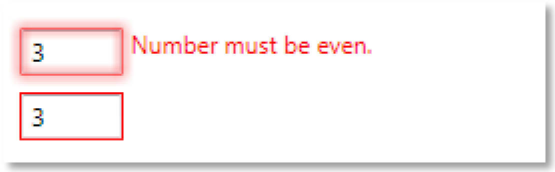
The following example shows how to create a validation template and apply it to a TextBox element.

```
<ControlTemplate x:Key="TextBoxErrorTemplate">
    <DockPanel>
        <Border BorderBrush="Red" BorderThickness="0.5">
            <Border.BitmapEffect>
                <OuterGlowBitmapEffect GlowColor="Red" GlowSize="5"/>
            </Border.BitmapEffect>

            <AdornedElementPlaceholder x:Name="ph" />
        </Border>
        <TextBlock
            Margin="4,0"
            Foreground="Red"
            Text="{Binding ElementName=ph,
               Path=AdornedElement.(Validation.Errors)[0].ErrorContent}" />
    </DockPanel>
</ControlTemplate>

<TextBox Validation.ErrorTemplate="{StaticResource TextBoxErrorTemplate}">
    <TextBox.Text>
        <Binding Source="{StaticResource MyData}" XPath="Team/@Score">
            <Binding.ValidationRules>
                <local:EvenValidationRule />
            </Binding.ValidationRules>
        </Binding>
    </TextBox.Text>
</TextBox>
```

The image below shows the results of this snippet in an error condition, and, as a reference, the default look is show at the bottom.



Silverlight 3 Note

New features for supporting data validation will be available in Silverlight 3 through the use of built-in validation controls and an exception system to notify controls of invalid data.

Dynamic Resources

Dynamic resources differ from static resources in that the framework will evaluate the dynamic resource every time it is used, whereas a static resource is evaluated once and cached.

The major reason for using dynamic resources is for situations where the value of the resource is not known until runtime; such as system resources, or other user-settable resources. Other reasons include situations that require adjusting the contents of a resource during the lifetime of the application.

Dynamic resources syntax is identical to static resources; other than specifying the DynamicResource markup extension instead of the StaticResource extension.

For more information on resources, see http://msdn.microsoft.com/en-us/library/ms750613.aspx.

Code Access Security

WPF supports the .Net Framework's implementation of code access security (CAS). This security mechanism limits access to system resources and functions. A full discussion of code access security is beyond the scope of this paper.

In essence, CAS defines permissions and permission groups that represent a right to access particular system resources. Code can then request permissions in which it needs to run, what it would like to have, and what permissions it should never be granted.

All WPF applications, except web browser applications (WBAs), require FullTrust permissions in order to run. Web browser applications, however, are run with partial trust; typically using one of the zone-based security sandboxes defined within Internet Explorer (such as the Internet or Intranet zones).

Silverlight Specific Functionality

Some functionality is available only to the Silverlight platform solely due to the fact that it is a web-based technology, hosted within a browser.

Web Functionality

Browser InterOp

A Silverlight application does not necessarily need to consume its entire hosted web page, and may exist as a smaller region within a rich HTML page. In the case where the Silverlight applet is a portion of a larger web page, it may become necessary for the applet to communicate with the page itself.

It is possible for Silverlight to call JavaScript methods, have Javascript call Silverlight methods, and have Silverlight interact with the HTML document object model (DOM).

To call a JavaScript method from Silverlight:

C#

```
using System.Windows.Browser;

// call a method
HtmlPage.Window.Eval("customMethod();");

// use invoke syntax
HtmlPage.Window.Invoke("customMethod");

// use invoke syntax with parameters
HtmlPage.Window.Invoke("customMethod", "param1", 12);

// use invoke syntax and get return value.
string name = (string) HtmlPage.Window.Invoke("getName");
```

Visual Basic

```
Imports System.Windows.Browser

' call a method
HtmlPage.Window.Eval("customMethod()")

' use invoke syntax
HtmlPage.Window.Invoke("customMethod")

' use invoke syntax with parameters
HtmlPage.Window.Invoke("customMethod()", "param1", 12)
```

```
' use invoke syntax and get return value.
Dim name As String = CType(HtmlPage.Window.Invoke("getName"), String)
```

To call a Silverlight method from JavaScript, the Silverlight class and name must be attributed correctly as ScriptableType and ScriptableMember respectively, as follows:

C#

```
[ScriptableType]
public partial class Page : UserControl
{
    public Page()
    {
        InitializeComponent();
        this.Loaded += new System.Windows.RoutedEventHandler(Page_Loaded);
    }

    void Page_Loaded(object sender, System.Windows.RoutedEventArgs e)
    {
        HtmlPage.RegisterScriptableObject("SilverlightMethods", this);
    }

    [ScriptableMember]
    public string GetUserName()
    {
        // assume implementation of UserName property.
        return this.UserName;
    }
}
```

VISUAL BASIC

```
<ScriptableType()> _
Partial Public Class Page
    Inherits UserControl

    Public Sub New()
        InitializeComponent()

        AddHandler Me.Loaded, AddressOf Page_Loaded
    End Sub

    Private Sub Page_Loaded(ByVal sender As Object, ByVal e As EventArgs)
        HtmlPage.RegisterScriptableObject("SilverlightMethods", Me)
    End Sub

    <ScriptableMember()> _
    Public Function GetUserName() As String
        ' assume implementation of UserName property.
        Return Me.UserName
    End Function
End Class
```

Then, within the JavaScript of the hosting page, add the following code:

```
<script type="text/JavaScript">
   function onPluginLoaded()
   {
      var sl = $get("Xaml1"); // Use ID associated with plugin.
      var sm = sl.content.SilverlightMethods;
      var name = sm.GetUserName(); // Case sensitive.

      // Do something...
   }
</script>
```

All HTML interoperability occurs through the static HtmlPage class; from registering scriptable methods to interacting with the Html document or window elements.

MEDIA

Silverlight supports rich media services delivered over the web. The Silverlight client supports several methods of receiving streaming media over the HTTP protocol.

Traditional Streaming

Silverlight can play media streamed to the client using a technique that sends data to the client at the bit rate of the encoded media, and only sends enough to fill the client buffer. Silverlight's default buffer length is 5 seconds. This methodology reduces bandwidth consumption since if the client is paused or cancelled, the only wasted bandwidth would be from the data stored in the buffer.

Progressive Download

Silverlight also implements the progressive download delivery method. Unlike smooth-streaming, the progressive download method streams the entire media file to the client. The client, however, can begin playing the media without it being completely downloaded. If the client pauses or cancels the player, the server continues to transmit the entire media file to the browser's cache.

Smooth Streaming

Silverlight also supports a relative new adaptive streaming technology dubbed Smooth Streaming. Smooth Streaming is a hybrid technology, combining traditional streaming with progressive downloading; transmitting a series of small progressive downloads instead of a single large one.

The Smooth Streaming technology is adaptive in that the client can request pieces of data of various sizes to optimize bandwidth consumption. In addition, the source media can be stored at different bit rates, and the client can seamlessly choose the best bit rate to use based on any number of factors, including network conditions and CPU usage.

The major advantages of Smooth Streaming are quick startup, no buffering, and constant adaptation to the environment. This insures a consistent, and more importantly, a smooth playback of media.

Timeline Markers

A timeline marker is metadata stored directly within the media file to provide the capability to skip to specific positions or provide cues for scripting. Within Silverlight, these timeline markers generate a MarkerReached event within the MediaElement class. The event can be handled in code to provide any number of enriching experiences, such as media pop-ups.

For example, the following XAML might be used to show media pop-ups as certain markers are reached within the media stream.

```
<StackPanel Orientation="Horizontal">
    <MediaElement x:Name="movieClip"
       Width="640"
       Height="480"
       MarkerReached="OnMarkerReached"
       Source="movieClip.wmv" />

    <TextBlock x:Name="popup"
       Width="200"
       Height="100"
       Visibility="Collapsed"
       Text="" />
</StackPanel>
```

The procedural code to display the popup could be implemented as follows:

C#

```
public void OnMarkerReached(object sender, TimelineMarkerRoutedEventArgs e)
{
    popup.Text = e.Marker.Text;
    popup.Visibility = Visibility.Visible;
}
```

Visual Basic

```
Private Sub OnMarkerReached(ByVal sender As System.Object, _
             ByVal e As TimelineMarkerRoutedEventArgs)

   Me.popup.Text = e.Marker.Text
   Me.popup.Visibility = Visibility.Visible
```

```
End Sub
```

The code to hide the popup after a set amount of time passes will be left as an exercise to the reader.

Silverlight's MediaElement class also populates a Markers collection containing all the timeline markers appearing in the media file.

Deep Zoom

Deep Zoom is a technology for viewing extremely high resolution images through Silverlight. The technology allows for quicker downloads of a large image by only displaying the portion being viewed. Deep Zoom also pre-processes an image into many separate resolution images to allow for speedier zooming and panning.

A separate pre-processing application, the Deep Zoom composer, is used by a designer to partition an image into a composition of smaller tiles or pieces at a variety of different resolutions. Deep Zoom calls this an image pyramid.

At run-time the Deep Zoom engine seamlessly stitches the pieces together depending upon how the image is being viewed.

For example, an image with a resolution of 10,000x10,000 pixels may first be displayed in a canvas sized at 600x600 pixels. The image downloaded to the browser would not be the entire image, but a lower resolution version of it. Zooming into the image would cause higher resolution pieces to be downloaded to the browser, but at no time would the entire 10,000x10,000 image be transferred.

Code Reuse Strategies

Sharing code between Silverlight and WPF can be a daunting task, as seen by the myriad differences between the two platforms. However, with careful planning, developers can share a significant amount of code between the two technologies. Microsoft is committed to bringing Silverlight and WPF closer together with each release.

Many of the differences in programming between Silverlight and WPF have been covered in the body of this whitepaper. However, this section compiles a list of some of the top techniques or strategies you can use to promote re-use of the code between the two platforms.

User Controls

Custom user controls are one way to begin sharing code between WPF and Silverlight applications. Controls built completely in code would work the most seamlessly, provided that any specific Silverlight or WPF code is cordoned off with the use of compiler directives. There may be some issues in sharing XAML directly due to differences in the xmlns declarations.

Partial Classes

Version 2.0 of the .Net Framework implemented a feature known as partial classes. Essentially, a partial class tells the .Net compiler that additional implementation exists in a separate file; thus, allowing a developer to organize their classes across multiple files.

When developing for Silverlight and WPF, partial classes could be used to organize a class into three files: the common code, the WPF-only code, and the Silverlight-only code. The WPF project, of course, would include the common code partial class as well as the WPF-only partial class; likewise, the Silverlight project would contain the common code and the Silverlight-only partial classes.

Generally speaking, both the WPF and Silverlight partial classes should contain implementations of the same methods and properties. This will insure that the main code base will function correctly without having to know which class implementation it is working with.

Extension Methods

New to version 3.0 of the .Net Framework are extension methods. Extension methods are static methods contained within a static class and have a particular method signature. These methods essentially, add functionality to existing value types or classes.

When developing for both Silverlight and WPF, developers could create two separate extension classes: one for WPF and one for Silverlight. Each project would contain their respective extension classes containing platform specific instructions.

Unfortunately, the implementation of an extension method is different between Visual Basic and C#. For example, to add a method called, GetPhysicalSize() to a Label, the following extension method can be used.

C#

```
public static class ExtensionMethods
{
    public static Size GetPhysicalSize(this Label label)
    {
       label.Measure(new Size(double.PositiveInfinity,
                                  double.PositiveInfinity));

       return new Size(label.ActualWidth, label.ActualHeight);
    }
}
```

VISUAL BASIC

```
Public Module ExtensionMethods

    <Extension()> _
    Public Function GetPhysicalSize(ByVal l As Label) As Size
        l.Measure(New Size(Double.PositiveInfinity, Double.PositiveInfinity))
        Return New Size(l.ActualWidth, l.ActualHeight)
    End Function

End Module
```

COMPILER DIRECTIVES

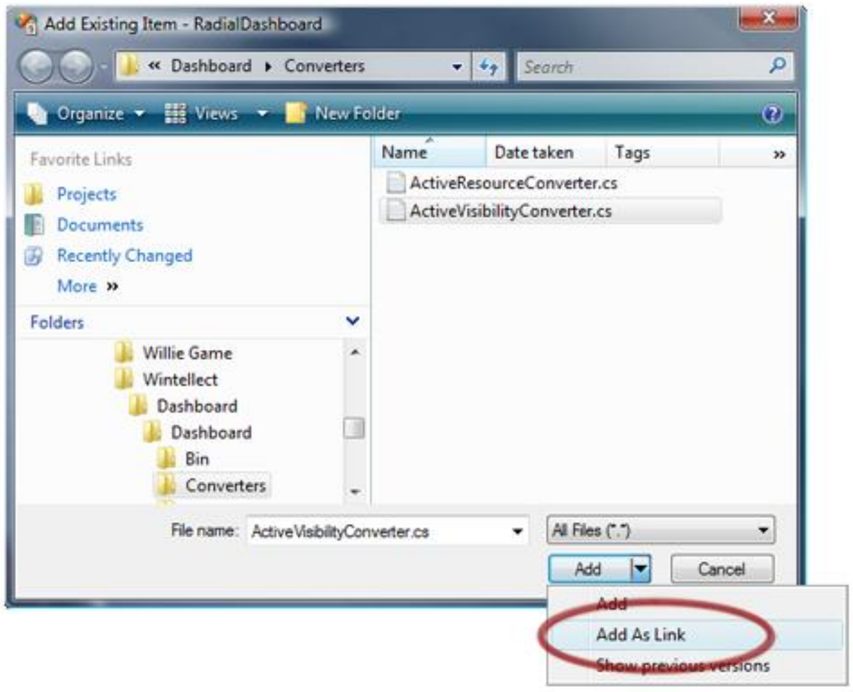
Compiler directives are embedded commands to instruct the compiler how to compile a certain section of code, also known as conditional compilation. At compile-time the conditional statements are evaluated and the proper code fragments are incorporated into the assembly.

Silverlight defines the boolean SILVERLIGHT compiler directive. This directive can be used to segment code that performs differently between the two platforms.

```
#if SILVERLIGHT
         // Perform Silverlight-only tasks.
#else
         // Perform WPF-only tasks.
#endif
```

LINKED FILES

Silverlight projects implement a separate CLR than WPF projects. Therefore, it is not possible to share sub-projects, such as a shared library, across applications. One solution to reuse code in both platforms would be to link files through Visual Studio into new sub-projects for each application.



CODING SCENARIOS

There are several scenarios when coding for both platforms that require some forethought in order to reduce code redundancy. While this whitepaper outlined many of the differences and similarities between the two platforms, the following section details some of the more common development areas where the differences between the two platforms become problematic when trying to share code.

Element-to-Element Binding

WPF supports element-to-element binding to facilitate designing the UI when controls rely on values from other controls. However, since Silverlight 2 does not support element-to-element binding, developers should concentrate on creating an intermediate class to store the values needed by all the controls and then bind the controls to that class. Developers familiar with design patterns will recognize this methodology as a form of the model-view-presenter (MVP) pattern.

For example, assume that a CheckBox control is responsible for enabling a group of controls. In WPF, simple element-to-element binding can set the IsEnabled property of the parent control. However, since Silverlight supports neither element-to-element binding nor the self-propagating IsEnabled property, an intermediate class could be used to store this setting.

The custom class could be constructed as follows:

C#

```
public class CustomSettings : INotifyPropertyChanged
{
    private bool _isEnabled = false;

    public bool IsEnabled
    {
        get { return _isEnabled; }
        set
        {
            _isEnabled = value;
            NotifyPropertyChanged("IsEnabled");
        }
    }


    // Snip remainder of implementation...
```

VISUAL BASIC

```
Public Class CustomSettings
   Implements INotifyPropertyChanged

   Private _isEnabled As Boolean = False

   Public Property IsEnabled() As Boolean
      Get
         Return _isEnabled
      End Get
      Set(ByVal value As Boolean)
         _isEnabled = value
         NotifyPropertyChanged("IsEnabled")
      End Set
   End Property

   ' Snip remainder of implementation
```

The sample XAML for the Silverlight Page might appear as follows:

```
<!-- Snip page declaration context -->

<UserControl.Resources>
   <local:CustomSettings x:Key="MySettings" />
</UserControl.Resources>

<!-- Snip page design -->

<StackPanel DataContext="{StaticResource MySettings}">
    <CheckBox Content="Is Enabled?"
      IsChecked="{Binding IsEnabled, Mode=TwoWay}" />
    <Button x:Name="btnClick1" Content="Click Me 1" Width="150" Height="29"
      IsEnabled="{Binding IsEnabled}" />
    <Button x:Name="btnClick2" Content="Click Me 2" Width="150" Height="29"
      IsEnabled="{Binding IsEnabled}" />
</StackPanel>
```

Silverlight 3 Note

The next version of Silverlight will support element-to-element binding. However, it may be desirable to separate the data model from the view. Several design patterns, including a new variation of the MVP pattern called Model-View-View Model (MVVM), recommend that approach.

Multi-Data Binding

Since Silverlight does not support multi-data binding, developers needing cross-platform support should build their application to the lowest common denominator. Building a separate class to aggregate the multiple values would mimic the WPF functionality. Both Silverlight and WPF can react to classes supporting INotifyPropertyChanged and parse the multiple values correctly in a ValueConverter.

Taking the Person class from earlier where multi-data binding is needed for when either the first name or last name changes:

C#

```
public class Person : INotifyPropertyChanged
{
   // Implementation snipped for brevity.

   public string FullName
   {
      get { return _firstName + " " + _lastName; }
   }

}
```

VISUAL BASIC

```
Public Class Person
    Implements INotifyPropertyChanged

    ' Implementation snipped for brevity.

    Public ReadOnly Property FullName() As String
        Get
            Return _firstName & " " & _lastName
        End Get
    End Property
End Class
```

A corresponding ValueConverter to return the first and last names concatenated into a string in the form of “lastname, firstname”:

C#

```
public class NameConverter : IValueConverter
{
   public object Convert(object value, Type targetType, object parameter,
            CultureInfo culture)
   {
      // Error checking omitted...

      string[] names = ((string) value).Split(' ');


      return string.Concat(names[1], ", ", names[0]);
   }

   // ConvertBack method snipped for brevity.

}
```

VISUAL BASIC

```
Public Class NameConverter
   Implements IValueConverter

   Public Function Convert(ByVal value As Object, _
      ByVal targetType As System.Type, _
      ByVal parameter As Object, _
      ByVal culture As CultureInfo) _
      As Object Implements IValueConverter.Convert

      ' Error checking omitted...
      Dim names() As String = CType(value, String).Split(" ")

      Return String.Concat(names(1), ", ", names(0))

    End Function

   ' ConvertBack method snipped for brevity.

End Class
```

Triggers / Visual State Manager

Adding UI effects to controls can be one of the most troublesome areas to reduce code redundancy. WPF supports various triggers to obtain visual effects while Silverlight implements a Visual State Manager. WPF only supports the Visual State Manager in the WPF Toolkit, and at the time of this writing, is only in beta. Using the VSM would yield the most mode reuse between the two platforms, assuming the WPF Toolkit has been installed.

See the earlier section on the Visual State Manager for an example code snippet.

Fonts

Silverlight has the capability of obtaining fonts in many different ways: from embedding as a resource within the assembly, or externally loaded via an asynchronous web request. WPF can use any font resource available on the Windows system without these extra steps. Since adding fonts to Silverlight applications can increase the size of the application's footprint, it is recommended to judiciously embed fonts and instead download them asynchronously when needed.

Asynchronously loading a font follows the same methodology as loading any asset asynchronously. In the case of a font, however, one needs to set the FontSource property to the incoming stream, and set the FontFamily to choose the font from the source.

C#

```
private void LoadFont()
{
    WebClient wc = new WebClient();
    wc.OpenReadCompleted += new
OpenReadCompletedEventHandler(wc_OpenReadCompleted);
    wc.OpenReadAsync(new Uri("/Assets/myfont.ttf", UriKind.Relative));
}

void wc_OpenReadCompleted(object sender, OpenReadCompletedEventArgs e)
{
    this.txtMessage.FontSource = new FontSource(e.Result);
    this.txtMessage.FontFamily = new FontFamily("Custom Font");
}
```

VISUAL BASIC

```
Private Sub LoadFont()
   Dim wc As New WebClient
   AddHandler wc.OpenReadCompleted, AddressOf wc_OpenReadCompleted
   wc.OpenReadAsync(New Uri("/Assets/myfont.ttf", UriKind.Relative))
End Sub

Private Sub wc_OpenReadCompleted(ByVal sender As Object, _
   ByVal e As OpenReadCompletedEventArgs)

   Dim fs As FontSource = New FontSource(e.Result)

   Me.txtMessage.FontSource = fs
   Me.txtMessage.FontFamily = New FontFamily("Custom Font")
End Sub
```

SUMMARY

WPF and Silverlight provide a new paradigm for application development using the same core technology of declarative XAML and the well-known code-behind model for procedural code. Silverlight, generally considered to be a subset of WPF, has been moving closer in implementation with each successive release. However, since Silverlight runs as an application within the plug-in model of a browser, it will always be hampered in some ways that will require innovative thinking to work around: whether in code or in application design. Future releases of both Silverlight and WPF have been making advancements to close the distance in the differences between the two platforms.

Almost as proof, with the upcoming release of Silverlight 3, many differences are either outright removed, or mitigated. In addition, Silverlight 3 will sport some new and interesting features (see http://silverlight.net/getstarted/silverlight3/default.aspx for more information).