

Introducing “Silverlight”!

by Mark Smith

If you've been watching the technical news or attended MIX '07, then you've probably been hearing about Silverlight (f.k.a. WPF/E). The idea behind Silverlight is simple -- provide a cross-browser, cross-platform technology for 2D vector graphics, animations and DRM high-definition media. Currently, Microsoft is targeting the Mac using either Firefox or Safari and Windows using IE6/IE7 or Firefox.

There are actually two different versions of Silverlight currently available. The first, 1.0 (beta), utilizes the same XAML tags as WPF to describe the visual appearance and animations for content. It uses Javascript to control the behavior of those elements by allowing Javascript functions to locate and manipulate properties of the elements created in XAML. This release targets web developers as it uses many of the technologies they are familiar with and use in their development today.

The second version, 1.1 (alpha) was announced at MIX '07 and it adds a CLR engine to the implementation - allowing the code behind to be done in C# or VB.NET instead of Javascript. This will radically change the playing field, as it will open up rich-client web applications to a wide audience of .NET developers.

The first step in using either version of Silverlight is to download the appropriate browser plug-in. You can get these from the Silverlight website <http://www.silverlight.net>. Both 1.0 and 1.1 versions have Mac and Windows implementations available. If you want to use Visual Studio .NET to do your development, then you should download the Orcas Beta release and the project template for Silverlight.

Next, you will create an .HTML host file. This file provides the starting point for the browser and instantiates the browser plug-in. It includes the required Javascript files and create a containing HTML element for the plug-in (typically a `DIV` tag). As an example, let's create a bare-bones HTML file for Silverlight 1.1:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

```

<head>
  <title>Silverlight 1.1 example</title>
  <script type="text/javascript"
    src="Silverlight.js"></script>
  <script type="text/javascript"
    src="createSilverlight.js"></script>
</head>

<body>
  <div id="SilverlightControlHost" >
    <script type="text/javascript">
      createSilverlight();
    </script>
  </div>
</body>
</html>

```

Here, you can see that two Javascript files are included - **Silverlight.js** (supplied with the SDK) and **createSilverlight.js** which is specific to this example and holds the **createSilverlight** method being invoked by this HTML host file.

The **createSilverlight** method looks something like:

```

function createSilverlight()
{
  Sys.Silverlight.createObjectEx(
  {
    source: "test.xaml",
    parentElement: document.getElementById(
      "SilverlightControlHost"),
    id: "SilverlightControl",
    properties:
    {
      width: "640",
      height: "480",
      version: "0.95",
      background: "#00000000",
      isWindowless: true,
      enableHtmlAccess: true
    },
    events: {}
  });
}

```

This code calls an existing function **Sys.Silverlight.createObjectEx**

that is part of the **silverlight.js** file. The function will identify the running browser and either use an `<object>` or an `<embed>` tag to load the browser plug-in for Silverlight. The passed properties allow the plug-in to be controlled – setting the width and height, background color and whether it allows HTML elements to occupy the same space as the plug-in. The function also identifies the starting .XAML file or element containing the XAML data to utilize.

The final required file is the .XAML file which contains the visual information to display – this can be generated by hand, or through a design tool such as Expression Blend or Expression Designer.

For this example, we'll use the following very simple declaration, which creates an ellipse and some text. We'll also throw in an animation to move the ellipse up and down.

```
<Canvas x:Name="rootCanvas"
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Canvas.Triggers>
    <EventTrigger RoutedEvent="Canvas.Loaded">
      <EventTrigger.Actions>
        <BeginStoryboard>
          <Storyboard>
            <DoubleAnimation
              Storyboard.TargetName="ellipse"
              Storyboard.TargetProperty="(Canvas.Top)"
              To="300" Duration="0:0:2" AutoReverse="true"
              RepeatBehavior="Forever" />
          </Storyboard>
        </BeginStoryboard>
      </EventTrigger.Actions>
    </EventTrigger>
  </Canvas.Triggers>
  <Ellipse x:Name="ellipse" Fill="Blue"
    Width="50" Height="50" />
  <TextBlock x:Name="text" Text="Silverlight is cool"
    Canvas.Left="15" Canvas.Top="15" FontSize="36pt"
    FontFamily="Arial" Foreground="DarkRed" />
</Canvas>
```

While that's interesting, where Silverlight *really* gets cool is in adding cross-platform media and code-based behavior. First, codecs are

supplied with Silverlight to allow it to decode DRM-protected HD content in the .WMV and .WMA format and play it in windowed or full screen mode. Media is added to the .XAML through the `<MediaElement />` tag. For example, we could replace the above ellipse with:

```
<MediaElement Source="someVideo.wmv" Width="100"
Height="100" Volume="0.8" Canvas.Top="10" Canvas.Left="10"
x:Name="ellipse" />
```

This would require a video on the web server called "someVideo.wmv". Since it has the proper name ("ellipse") the video would now be moving up and down instead of the ellipse.

A second main feature of Silverlight is the ability to extend it through code. Silverlight 1.0 supports Javascript support – where additional .js files are included by the root HTML page and they have coded functions that handle events from the elements in XAML. The events are wired up using the syntax "javascript:function". For example, we could handle the `Canvas.Loaded` event through the following syntax:

```
<Canvas x:Name="rootCanvas" ...
        Loaded="javascript:Page_Loaded" />
```

The `Page_Loaded` function would be located in a .js file and be passed two parameters – `sender` and `eventArgs`:

```
function Page_Loaded(sender, eventArgs) {
    ...
}
```

With Silverlight 1.1, you can utilize a portable CLR implementation and write your code behind using C# or VB.NET. This follows the desktop paradigm of WPF by supporting an `x:Code` tag on the root canvas which identifies the code behind file to execute. For example, we could add a `Canvas.Loaded` handler in C# by changing our root canvas tag to include:

```
<Canvas x:Name="rootCanvas" ...
        Loaded="Page_Loaded"
        x:Class="Test.Page;assembly=bin/Test.dll">
```

This identifies the Test.dll assembly in the bin directory on the web server and the plug-in will download the assembly and wire up any identified event handlers such as the Loaded event above. The code behind file needs to be compiled and is typically named xxx.xaml.cs where “xxx” is the name of the XAML file it has the code behind for. In the above example, our code behind file will be test.xaml.cs and it will declare a class “Page” in C#.

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;

namespace Test
{
    public partial class Page : Canvas
    {
        public void Page_Loaded(object o, EventArgs e)
        {
            // Required to initialize variables
            InitializeComponent();

            Canvas canvas = o as Canvas;
            TextBlock tb = (TextBlock)
                canvas.FindName("text");

            tb.Text = DateTime.Now.ToString();
        }
    }
}
```

Notice that it is a partial class – the other half of the class is auto-generated by the XAML file when the project is compiled. The Page_Load handler takes the sender, casts it to the canvas and then uses that to locate the TextBlock, which is then set to the current DateTime. The fantastically cool part of this is that this Silverlight page can be executed on the Mac platform as well as Windows – and presumably Linux will follow shortly! Using this 1.1 version will allow you to transition your existing Microsoft .NET skills to build rich web-based applications without requiring you to relearn a whole new platform.

We’ve barely scratched the surface of Silverlight here – if you would

like to learn more, DevelopMentor has an upcoming Silverlight class as well as coverage in our 5-day Essential WPF class which will get you up to speed on both the desktop technology and Silverlight! Check out our website at www.develop.com for more information.