Chapter 1

Introducing Silverlight

Objectives:

- Understand the WPF / Silverlight relationship
- Understand the scope of the Silverlight platform
- Examine the Silverlight runtime and base class libraries
- Learn the role of the Application and UserControl classes
- Learn the syntax and semantics of XAML
Chapter Overview

When initially released, Silverlight was positioned as a web-centric .NET technology, which leverages a number of programming features first introduced with Windows Presentation Foundation (WPF). Essentially, Silverlight applications are similar in concept to an Adobe Flash plug-in. However unlike a Flash applet, Silverlight applications are created using managed code (C#, F#, VB, etc.).

As Silverlight has matured, numerous features have been added to the API, which move applications from the browser to other locations. With the release of Silverlight 3.0, it became possible to run a Silverlight application on the user’s desktop. As well, the Windows Phone 7 platform uses a dialect of Silverlight as the core programming model.

In a nutshell, Silverlight has quickly become a key .NET programming API which can be used for a variety of applications (web, desktop, mobile).

This chapter gives an overview of Silverlight, including the overarching programming model and the Application and UserControl classes. Furthermore, this chapter will examine the role of XAML, the XML based grammar typically to describe the UI of a Silverlight application in a declarative manner.

The information presented here will serve as a foundation for the remainder of the class.
Understanding the WPF / Silverlight Relationship

- Silverlight is a .NET programming API, which has its roots in web development.
  - Using Silverlight, you are able to create and embed highly interactive and media-rich content into your web applications.
  - This “media-rich web content” is termed a Silverlight application.
- Typically, a Silverlight application is part of a larger web page.
  - This web page does *not* need to be created using ASP.NET. Any web development framework can embed a Silverlight application.
  - As you will see later in the class, a web page can embed a Silverlight application using little more than an object tag and some client side JavaScript.
  - It is possible to configure a Silverlight application to consume the entire client area of the browser (as opposed to being part of a larger web page).
- Ironically, the best way to understand the nature of Silverlight is to examine a .NET desktop programming API; WPF.
  - .NET 3.0 introduced the Windows Presentation Foundation (WPF) API.
  - WPF is a supercharged GUI framework for building rich desktop applications.
  - WPF integrates vector graphics, animations, a rich data binding engine, video / audio services (among other services) into a single unified programming model.
  - WPF also radically simplifies the way we customize controls using the content, style and template models.
Introducing Silverlight

- Using WPF, you are able to build extremely rich next-gen desktop user interfaces.
  - Consider the following WPF sample program.
  - Although not viewable from the printed page, when you click on any of the stylized buttons on the bottom of the window, a 3D-rendered motorcycle spins into view.
  - Also notice the transformed text blocks and dynamic shadowing.
  - While such an application could be built without WPF, doing so would require a large amount of low-level code and would consume quite a bit of time.

- You can find many more samples of WPF applications on the official web site, www.windowsclient.net.
  - As counterintuitive as it may seem, the more you know about WPF, the more you know about Silverlight (and vice-versa).
  - For example, both WPF and Silverlight applications are developed using Visual Studio and Expression Blend.
  - More importantly, Silverlight and WPF share a very similar programming model (XAML, control content, templates, styles, etc...)
The Role of Silverlight

- Silverlight is a web-centric, .NET programming framework, based on several WPF technologies.
  - However, unlike WPF, Silverlight allows you to build highly interactive *web plug-ins*.
  - The client machine hosting the Silverlight application does *not* require a full installation of the .NET Framework.
  - Silverlight provides its own “miniaturized” version of the CLR and .NET base class libraries.
- The major goal of Silverlight is to bring the same quality of a WPF desktop UI (e.g., highly interactive, graphically rich, etc.) to the web.
  - Consider Silverlight to be Microsoft’s answer to Adobe Flash.
- Silverlight is not limited to Microsoft OSs / browsers.
  - Silverlight applications can run on the Windows and Mac OS X operating systems.
  - Furthermore, Silverlight applications can be hosted by multiple browsers (IE, Safari, Opera, and Firefox).
- Regardless of the OS / browser, the machine must download and install the free Silverlight runtime / library stack.
  - You will come to know the nature of the Silverlight runtime later in this chapter.
  - For now, understand that if an end user navigates to a web site requiring this plug-in, they will need to install the runtime if it is not currently installed.
  - After this point, your Silverlight application will load into the browser and be hosted by the related runtime.
The official Silverlight web site ([www.silverlight.net](http://www.silverlight.net)) provides links to numerous downloads:

- The Silverlight runtime for Windows and Mac OS X.
- The Silverlight 4.0 developer tools for Visual Studio 2010 (which includes the SDK, Silverlight 4.0 project templates, the F# Silverlight runtime, etc.)
- Sample applications, white papers, video tutorials, etc.
- Simply click on the Get Started link at the top of the home page.
Silverlight applications can be developed on, and hosted by; Linux based OSs using the Moonlight API.

- Moonlight is an open source implementation of Silverlight, created by the Mono/Novell team.
- In fact, Moonlight was built in collaboration with Microsoft.
The Role of XAML

- XAML is a key aspect of building Silverlight (and WPF) applications.
  - XAML is an XML-based grammar which can be used to describe a tree of .NET objects (UI-based or otherwise).
  - The XAML file data is typically embedded into a WPF/Silverlight application as a binary resource, termed ‘BAML’ (the Binary Application Markup Language).
  - At runtime, BAML is used to hydrate objects into memory.
  - You will learn about the XAML ↔ BAML relationship in more detail in your first lab.
- XAML provides a clear separation of concerns between GUI layout and an application’s functionality.
  - For example, a Silverlight UserControl is typically composed of a XAML file to describe its look and feel, in addition to a related code file (C#, VB, F#, etc.) to describe its behavior (event handlers, custom methods, constructors, etc.).
  - As you will see later in this class, we can further provide a clear separation of concerns using the Model-View-ViewModel (MVVM) design pattern.
- XAML also allows for a separation of concerns at the tool level.
  - A graphical artist can use dedicated tools (such as Microsoft Expression Blend or Expression Design) to generate the XAML.
  - Programmers can use Visual Studio to add the coding functionality.
  - Expression Blend and Visual Studio share an identical solution/project format; the same project can be opened into either tool at any time.
  - This allows different skill sets (artists and coders) to cooperatively build a single Silverlight (or WPF) application, using appropriate tools.
• You will learn more about XAML, Expression Blend and Visual Studio’s Silverlight support over the remainder of this class.
  • For the time being however, consider the following screen shot which shows the XAML editor of Expression Blend.
  • While you could directly enter the required markup, Blend will generate a majority of the XAML on your behalf.
  • Visual Studio 2010 can also be used to help automate XAML generation; however Blend still has much greater support for UI construction.
Silverlight’s Functionality through the Versions

• The 1.0 release of Silverlight was nothing like what we have today.
  • Silverlight 1.0 did not support .NET programming languages, and did not ship with a full library of reusable code.
  • While 1.0 did support XAML, this version of the framework did not include a set of standard controls.
  • Thus, 1.0 developers were required to roll their own functionality, which included a healthy amount of JavaScript code.
  • This course will not address Silverlight 1.0, as it is little more than a technology footnote.
• Silverlight 2.0 was the first version of the API which shipped as a .NET aware API.
  • 2.0 did away with JavaScript, in place of .NET languages such as C# and VB.
  • Just as important, Silverlight 2.0’s runtime is a micro version of the .NET CLR which runs within the hosting browser.
  • Just like the .NET CLR, the Silverlight runtime supports garbage collection, dynamic loading of libraries, the Common Type System, and so forth.
  • Version 2.0 also shipped with a micro version of the .NET base class libraries. Here you will find support for threading, feature rich controls, XML manipulation, LINQ, WCF support and other key .NET APIs.
  • Most developers consider Silverlight 2.0 the first “real” version of the programming model.
Silverlight 3.0 is backwards compatible with 2.0. However this release of the API did add a number of new features, including:

- OOB (out-of-browser) support. This allows you to deploy a Silverlight application to the Windows or Mac OS desktop, where it runs as a typical windows application.
- Richer support for themes and styles.
- Full support for object resources (equivalent to WPF support).
- Perspective 3D graphics.
- Improvements to the data binding engine (e.g., control to control data binding).

Silverlight 4.0 offers us more of a good thing. Building on the 2.0-3.0 core, version 4.0 adds support for various features, including:

- Printing support services.
- COM interop via late binding (IDispatch based access).
- Microphone and Web cam support.

At the time of this writing, Silverlight 5 is in beta development.

- Consult the following URL for information regarding the new features proposed by Silverlight 5.

  http://www.silverlight.net/learn/overview/what's-new-in-silverlight-5
The Role of the Silverlight Runtime and Base Class Libraries

- The Silverlight runtime is essentially a ‘micro’ version of the full .NET CLR.
  - This runtime is hosted within the client side browser (or the host OS, for OOB).
  - This runtime will load and manage your Silverlight application, as well as the required Silverlight base class libraries.
  - This runtime will JIT CIL code and manage threads, exceptions and garbage collection.
  - Simply put, your current .NET programming skills apply directly to Silverlight development.
- In addition to a ‘micro-CLR’, Silverlight supports a subset of the .NET base class libraries.
  - When a requesting browser loads a web page using a Silverlight plug-in, any required assemblies are downloaded to the user’s machine, if they are currently not installed.
  - Like any other .NET program, you can build your own code custom libraries for use within your Silverlight programs.
- Many of the Silverlight libraries are identically named to their .NET counterparts (for example, mscorlib.dll, System.dll, System.Xml.dll, etc.)
  - However in every case, the Silverlight equivalents are lean-and-mean versions which support a *subset* of the functionality found in their .NET counterparts.
- For example, .NET and Silverlight each provide a System.Threading namespace within the mscorlib.dll assembly.
  - However, the Silverlight implementation contains fewer types than that of .NET proper.
  - Given this, be careful when incorporating .NET code libraries into a Silverlight application! Testing will be necessary.
• A Silverlight application is deployed from a web server as an ‘XAP’ file format.
  • The XAP file contains the compiled Silverlight application, any embedded resources, and any required external assemblies.
  • The Silverlight runtime downloads the XAP package and executes its contents.
  • As you will see in your lab, a XAP file is really just a ZIP file which contains a .NET *.dll and a deployment manifest.

• The following diagram illustrates the relationship between the client-side and server-side resources:
  • As expected, HTTP is used to request the initial web page containing the Silverlight application.
  • Once loaded by the browser, the Silverlight application is able to make remote calls (to the original web server or any arbitrary endpoint) using WCF.
  • The entire web page does *not* need to be refreshed when the Silverlight application changes state, as it is running on the client computer.
Core Silverlight Assemblies and Namespaces

- Like any .NET technology, Silverlight is represented by several assemblies.
  - The Visual Studio 2010 Silverlight project templates automatically set references to the core assemblies.
  - Like any other .NET application, you are able to reference additional assemblies via the Add Reference dialog.
- Remember, the Silverlight libraries are *not* identical to the .NET libraries, regardless of the fact they have the same physical names.
  - When you build a Silverlight project, you are referencing the “Silverlight base class libraries” which are completely independent from the .NET base class libraries.
• The following table lists the basic functionality found within the core Silverlight libraries.

• You will make use of many of these libraries during your lab work.

<table>
<thead>
<tr>
<th>Silverlight Assembly</th>
<th>Meaning in Life</th>
</tr>
</thead>
<tbody>
<tr>
<td>mscorlib.dll</td>
<td>This is a ‘mini-version’ of the .NET mscorlib.dll assembly. Here you will find the core based types and namespaces.</td>
</tr>
<tr>
<td>System.dll</td>
<td>A mini-version of the .NET System.dll assembly.</td>
</tr>
<tr>
<td>System.Core.dll</td>
<td>This library defines the core LINQ programming model.</td>
</tr>
<tr>
<td>System.Xml.dll</td>
<td>Provides types to manipulate XML documents.</td>
</tr>
<tr>
<td>System.Net.dll</td>
<td>Provides types which allow you to interact with HTTP request / response and sockets.</td>
</tr>
<tr>
<td>System.Windows.dll</td>
<td>This core Silverlight library contains types for animation, data binding, resource management and graphical rendering. It is a slimmed down version of the WPF equivalent.</td>
</tr>
<tr>
<td>System.Windows.Browser.dll</td>
<td>Provides types which allow your Silverlight plug-in to communicate with the hosting browser.</td>
</tr>
</tbody>
</table>
• The following table lists some (but not all) of the core Silverlight-centric namespaces you will encounter within the Silverlight core libraries.
  
  • Notice that these namespaces begin with System.Windows, which can be confusing, as you are not building a desktop Windows application!
  
  • Recall that Silverlight is indeed based on numerous WPF technologies, and therefore shares similar naming conventions.

<table>
<thead>
<tr>
<th>Silverlight Namespace</th>
<th>Meaning in Life</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.Windows</td>
<td>Here you will find core types such as Application, as well as types to support styles, templates and data binding.</td>
</tr>
<tr>
<td>System.Windows.Controls</td>
<td>Here you will find the core controls and layout managers, including the UserControl class, which all Silverlight plug-ins extend.</td>
</tr>
<tr>
<td>System.Windows.Data</td>
<td>Types to facilitate data binding operations.</td>
</tr>
<tr>
<td>System.Windows.Markup</td>
<td>This namespace defines a number of types that allow XAML markup and the equivalent binary format, BAML, to be parsed.</td>
</tr>
<tr>
<td>System.Windows.Media</td>
<td>Within these namespaces you will find types to work with animations, graphical rendering, text rendering, and other multimedia primitives.</td>
</tr>
<tr>
<td>System.Windows.Shapes</td>
<td>This namespace defines interactive geometric shapes (Rectangle, Polygon, etc.) used by various aspects of the Silverlight framework.</td>
</tr>
</tbody>
</table>

• In addition to the Silverlight-centric namespaces, you will be happy to see a number of commonly used .NET namespaces are also supported, including:
  
  
  • System.IO, System.IO.IsolatedStorage: Provides safe access to the client side file system.
  
  
  • System.Security.*: Numerous namespaces for securing a Silverlight web plug-in.
  
  • System.ServiceModel.*: Numerous namespaces for WCF functionality.
  
  • System.Xml, System.Xml.Linq: Provides an XML based programming model.
The Silverlight API Documentation

- To be sure, it is not possible to cover *every* possible aspect of Silverlight in this course.
  - Once the class is completed, you will be armed with knowledge of the Silverlight platform, XAML and various development tools.
  - When you are back at work, you will need to apply what you have learned to your specific business needs.
  - More importantly, you will need to dive deeper into the specific aspects of Silverlight which are most interesting.

- Full documentation of the Silverlight libraries can be found online at the MSDN website.
  - A full list of each supported namespace can be found under the .NET Framework Class Library for Silverlight section of the Silverlight documentation.
  - Simply navigate to www.msdn.com and then search .NET Framework Class Library for Silverlight.
  - Here you will find full descriptions of each namespace in the Silverlight API.
The Role of the Application Class

- Silverlight applications will always contain a class which extends `System.Windows.Application`.
  - This class represents a running instance of a Silverlight application.
  - It encapsulates a number of core services such as routing messages, trapping unhandled exceptions, defining common application data, and more.
  - By convention, this class is named `App`, and is represented via the App.xaml and App.xaml.cs / App.xaml.vb files.
  - When you create a new Silverlight application using Expression Blend or Visual Studio 2010, you will receive an Application derived class automatically.
• The Application class defines a number of key services.
  • Here is a partial list of important members (properties, methods and events).
  • Consult the Silverlight documentation for full details.

<table>
<thead>
<tr>
<th>Member of Application Class</th>
<th>Meaning in Life</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td>This static property provides access to the global application object. This allows a Silverlight control to access the application, which is very helpful in that the app object tends to define core functionality for all owned objects (resources, and so on).</td>
</tr>
<tr>
<td>Host</td>
<td>Gets information about the host of this Silverlight application.</td>
</tr>
<tr>
<td>Resources</td>
<td>Allows you to package up application level resources used within the Silverlight application.</td>
</tr>
<tr>
<td>InstallState</td>
<td>Gets the current out-of-browser installation state of the application.</td>
</tr>
<tr>
<td>IsRunningOutOfBrowser</td>
<td>Gets a value that indicates whether the application was launched from the out-of-browser state.</td>
</tr>
<tr>
<td>RootVisual</td>
<td>Specifies the UserControl which will be displayed when the Silverlight application is loaded by the browser.</td>
</tr>
<tr>
<td>CheckAndDownloadUpdateAsync()</td>
<td>This method launches an asynchronous process to check for and download an updated version of the application.</td>
</tr>
<tr>
<td>Install()</td>
<td>Attempts to install the application so that it can run outside the browser.</td>
</tr>
<tr>
<td>Exit, Startup, InstallStateChanged, UnhandledException</td>
<td>The Application class exposes a number of events which fire during the lifetime of your Silverlight application.</td>
</tr>
</tbody>
</table>
- The XAML description of your application object is quite sparse upon start-up.
  - Note the **x:Class** XAML token, which is set to the name of the related class in your code file.
  - As well, you will find an empty resource container.
  - You'll learn about Silverlight resource management later in this class.

```xml
<Application xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             x:Class="SimpleSilverlightApplication.App">

  <!-- Application.Resources -->

</Application>
```
The constructor of the related code file will handle the Startup, Exit, and UnhandledException events.

- As well, the constructor makes a call to `InitializeComponent()`.
- As illustrated in your next lab, the compiler will generate a partial class file which contains the `InitializeComponent()` method.
- The `RootVisual` property is set to a new MainPage object, which is the name of the UserControl derived class (described next).

```csharp
// C# (VB code would be similar)
public partial class App : Application
{
    public App()
    {
        // VB events would be hooked-up via the Handles keyword.
        this.Startup += this.Application_Startup;
        this.Exit += this.Application_Exit;
        this.UnhandledException += this.Application_UnhandledException;

        InitializeComponent();
    }

    private void Application_Startup(object sender, StartupEventArgs e)
    {
        // Set the UserControl to display upon startup.
        this.RootVisual = new MainPage();

        // Preform additional start up logic here.
    }

    private void Application_Exit(object sender, EventArgs e)
    {
        // Preform any clean up here.
    }

    private void Application_UnhandledException(object sender, ApplicationUnhandledExceptionEventArgs e)
    {
        // Deal with unhandled exceptions here.
        // You can obtain the System.Exception object via the event args.
    }

    // Preform additional end up logic here.
}
```
The Role of the UserControl Type

- A Silverlight application will also define a class extending UserControl.
  - This represents the main UI and functionality of the Silverlight application.
  - Visual Studio names this class **MainPage**, and is the same class which is set to the **RootVisual** property with the application Startup handler.

- The initial XAML definition of the MainPage class is quite simple.
  - Again, the **x:Class** attribute is used to connect this markup to a related class definition.
  - Note that a **<Grid>** is used as the default layout manager, however you can change this to a different container if need be.
  - More details on layout managers can be found in Chapter 2.

```xml
<UserControl x:Class="SimpleSilverlightApplication.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">
  <Grid x:Name="LayoutRoot" Background="White">
  </Grid>
</UserControl>
```
• The initial code file is also straightforward:

  • Notice the MainPage is-a UserControl.

  • Similar to the Application derived type, InitializeComponent() is within a partial class file.

```csharp
// C# (VB code would be similar)
public partial class MainPage : UserControl
{
    public MainPage()
    {
        InitializeComponent();
    }
}
```
Understanding the Silverlight Control Content Model

- The UserControl parent class allows derived types to host ‘content’.
  - Simply put, “content” is which is displayed in the interior of a control.
  - The Silverlight content model demands that a UserControl specifies a single piece of content.
  - Most often, the ‘single piece of content’ will be a layout manager containing all the UI required elements, vector graphics, etc.

- The Content property of UserControl can be used to set the content to display in code.
  - In XAML, the first sub element of the <UserControl> scope will be used to implicitly set the Content property.
  - In addition to the Content property of UserControl, be aware that a majority of Silverlight controls extend the ContentControl parent class.
    - This class also defines a Content property for a similar purpose.
    - For example, the Button class extends ContentControl, and therefore can participate in the Silverlight content model.
• By way of an example, a Button could maintain an inner StackPanel as ‘content’.
  
  • The **StackPanel** contains an **Ellipse** and **TextBlock**.
  
  • You will examine the details of the content model in a later chapter. For now, here is a simple example in XAML.

```xml
<!-- The Content of this UserControl is the following Button -->
<Button Height = "150" Width = "120">
  <!-- This button has a StackPanel as content. -->
  <StackPanel>
    <Ellipse Fill = "Orange" Height = "75" Width = "75"/>
    <TextBlock Text = "OK!" FontSize = "20" HorizontalAlignment = "Center" />
  </StackPanel>
</Button>
```

• While the previous example is extremely simple, it does illustrate how easy it is to customize Silverlight controls.
  
  • Rather than needing to subclass from the Button class, override numerous virtual methods and author a good amount of procedural code, we are able to reshape the entire of a control via the content model.
  
  • As you will see in later chapters, Silverlight supports additional ways to customize controls using styles and templates.
The Syntax of XAML

- Although it is true that Silverlight development tools will often generate XAML on your behalf, it is important to understand the underlying syntax.
  - This will allow you to tweak the generated XAML if the need should arise.
  - Also, XAML can be generated, loaded, parsed, and manipulated at runtime.
  - To understand how to do so, a working knowledge of XAML’s syntax is useful.
- It is important to point out that XAML is a general-purpose markup language and is not limited to Silverlight.
  - As noted earlier in this chapter, XAML is also within WPF applications.
  - XAML can also be found within the Windows Workflow Foundation API (WF), where it can be used to build custom activities and workflows via markup.
  - XAML is also used by the XML Paper Specification (XPS), a Microsoft technology for e-paper.
- Although WPF, WF, XPS and Silverlight all use XAML, be aware that the underlying schemas are not identical!
  - Therefore, don’t be too surprised if you attempted to copy a WPF XAML description into a Silverlight project, and need to modify the markup.
  - As a good rule of thumb, consider Silverlight XAML to be a subset of WPF XAML.
The role of XML Namespaces

- The root element of a XAML document typically defines two key XML namespaces that map to core Silverlight .NET namespaces and definitions of XAML specific tokens.
  - [http://schemas.microsoft.com/winfx/2006/xaml/presentation](http://schemas.microsoft.com/winfx/2006/xaml/presentation) maps a number of Silverlight namespaces for use by the current *.xaml file:
  - [http://schemas.microsoft.com/winfx/2006/xaml](http://schemas.microsoft.com/winfx/2006/xaml) is used to include XAML specific tokens, as well as a subset of types within the System.Windows.Markup namespace.
- Here would be a `<UserControl>` that defines these two XML namespaces.
  - Given the cascading nature of XML, all sub-elements of the `<UserControl>` have access to the same information.

```xml
<UserControl x:Class = "SimpleSilverlightApplication.MainPage"
    xmlns = "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x = "http://schemas.microsoft.com/winfx/2006/xaml"
    ...
    >
    <!-- Add content here! -->
</UserControl>
```
The first xmlns attribute is the primary namespace as it has not been qualified with a tag prefix.

- Notice that the second xmlns attribute has been given the ‘x’ prefix. This is simply to avoid ambiguity with the other XML namespace definitions.
- Like any XML prefix, the actual name is irrelevant. There is nothing “magical” about the letter x.
- Thus, the following <UserControl> definition is also permissible although more verbose.

```xml
...>
</UserControl>
```

- Beyond the two key XML namespaces, XAML makes it possible to define custom xmlns values that map to additional assemblies / namespaces.
  - This can be helpful when your markup needs to refer to types defined in external code libraries.
  - This is very common when your Silverlight application needs to access custom controls packaged up in .NET class libraries.
  - The clr-namespace token is used to do this very thing.
- Here is an example that makes the types in the System namespace of mscorlib.dll available within the current <UserControl>.
  - If you are mapping to a namespace in the current Silverlight project, the assembly qualifier is optional as seen in the second XML namespace listing.

```xml
<UserControl x:Class = "SimpleSilverlightApplication.Page"
xmlns = "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x = "http://schemas.microsoft.com/winfx/2006/xaml"
... xmlns:SystemCorLib = "clr-namespace:System;assembly=mscorlib"
xmlns:MyTypes = "clr-namespace:SomeNamespaceInThisAssembly"
>
</UserControl>
```
XAML-Specific Tokens

- Strictly speaking, XML-based grammars do not support true keywords.
  - However, XML parsers can be programmed to look for special tokens that will be treated in a special, customized manner.
  - The same is true for XAML as it defines a number of tokens that can be regarded as ‘keywords’ in a general sense.
  - Many of these are used to control how the XAML markup is processed at compile time.

- Because these XAML-specific tokens are part of the XAML namespace, they should be qualified with the x: prefix (x:Name, x:Class, and so forth).

- You will see additional XAML tokens beyond the following over the course of this class; however here is a rundown of common options.

<table>
<thead>
<tr>
<th>Token from the XAML XML Namespace</th>
<th>Meaning in Life</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Used to establish the name of the class used in conjunction with the markup.</td>
</tr>
<tr>
<td>Key</td>
<td>Allows you to establish a key value for a XAML item that will be placed into a resource dictionary.</td>
</tr>
<tr>
<td>Name</td>
<td>This allows you to specify the generated C# / VB name of a given XAML element.</td>
</tr>
<tr>
<td>Null</td>
<td>Represents a null reference.</td>
</tr>
<tr>
<td>ClassModifier</td>
<td>Allows you to control the visibility of the compiler generated member or class.</td>
</tr>
<tr>
<td>FieldModifier</td>
<td></td>
</tr>
<tr>
<td>StaticResource</td>
<td>Allows you to make reference to a resource declared elsewhere.</td>
</tr>
<tr>
<td>Binding</td>
<td>Used to establish a data binding operation in markup.</td>
</tr>
</tbody>
</table>
Controlling Element Visibility

- The x:Class token is a commonly used XAML attribute.
  - A UserControl or Application can specify an **x:Class** attribute in its opening definition and is used to define the name of the class type in the code files.

```xml
<UsrControl x:Class = "SimpleSilverlightApplication.MainPage"
xmlns = "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x = "http://schemas.microsoft.com/winfx/2006/xaml">
</UsrControl>
```

- The **x:ClassModifier** / **x:FieldModifier** attributes allow you to control the visibility of a member in the related code file.
  - A XAML file is often paired with a C# or VB code file where you author event handlers, helper methods, and so forth.
  - An additional compile-time generated code file (*.g.cs / *.g.vb) will be used to contain the XAML => code object mapping, control declarations, and more.
  - You will examine this compiler generated file during your lab time.

- If you do not make use of the **x:ClassModifier** / **x:FieldModifier** attributes, the item will be defined using the default visibility of the .NET language.
  - In most cases, you will not need to change these defaults.
  - As a result, you will not frequently need to make use of the **x:ClassModifier** or **x:FieldModifier** tokens.
• As a simple example however, consider the following use of the x:ClassModifier and x:FieldModifier attributes.
  • This will be used to declare a code file containing an internal class with a public Button.
  • The VB code would be similar, using the Friend keyword rather than the C#-specific internal keyword.

<!-- This class will now be internal. If using a code file, the partial class must also be defined as internal. -->
<UserControl x:Class = "SimpleSilverlightApplication.MainPage"
    xmlns = "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x = "http://schemas.microsoft.com/winfx/2006/xaml">
  <Button x:Name = "myButton" Content="OK" x:FieldModifier = "public"/>
</UserControl>

// C# code (VB code would be similar)
internal partial class MainPage : UserControl
{
    ...
}
Naming Elements

- When you are declaring an item in XAML, you typically will want to assign a value to the x:Name attribute.
  - This becomes the name of the member variable in the code file. You can use this member variable to refer to the item in your code file.
  - Because the need to name a UI element is so common, you can omit the x: prefix when assigning the Name attribute, when referring to any Silverlight control which extends FrameworkElement.
  - If the item does *not* extend FrameworkElement, you *must* use x:Name rather than Name.
  - Since Button is-a FrameworkElement, the previous Button could be defined as so:

```xml
<-- No x: prefix on the Name attribute...still OK! -->
<Button Name = "myButton" Content="OK" x:FieldModifier = "public"/>
```
XAML Property-element Syntax

- Within the scope of an opening element, you will be able to set values to the properties and events of the type.
  - For example, the following `<Button>` sets the **Content**, **Height** and **Width** properties and handles the **Click** event.
  - The name assigned to the Click event will map to a method in your code file.

```xml
<Button Name = "myButton" Height = "100" Width = "100"
      Click = "myButton_Click" Content = "OK" />
```

- Here, Content, Height, Width, and Click were assigned values that could be captured as a simple string.
  - Behind the scenes, these strings are transformed into specific data types using built in type converters (example, the string "100" becomes a double data type).
  - However, many properties of classes do not operate on data that represents simple string values.
  - For example, some properties require full-blown objects (complex brushes, pens, and so on).

- XAML property-element syntax allows you to assign complex object values to a property.
  - This syntax defines a subscope scope representing a property of the defining type.
  - Within this scope, you can describe the object to be used for the property assignment.
  - The format follows the following template:

```xml
<DefiningType>
  <DefiningType.PropertyOnDefiningType>
    <!-- data used to set property -->
  </DefiningType.PropertyOnDefiningType>
</DefiningType>
```
• Consider the following syntax, which sets the Background property of a Button to a LinearGradientBrush type using property-element syntax.

• The associated image shows you how this button would look when rendered by a XAML parser.

```xml
<Button x:Name="myButton" Height="100" Width="100" Content="Click Me!">
  <Button.Background>
    <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
      <GradientStop Color="Blue" Offset="0" />
      <GradientStop Color="Yellow" Offset="0.25" />
      <GradientStop Color="Green" Offset="0.75" />
      <GradientStop Color="Pink" Offset="0.50" />
    </LinearGradientBrush>
  </Button.Background>
</Button>
```

• By way of another example, here we are setting a drop shadow effect for a Button using property-element syntax.

• Take a moment to ponder how simple it is to apply a visual effect.

• Silverlight supports numerous visual effects (swirl, blur, sharpen, etc.)

```xml
<Button Content="Nice!" Margin="32,26,0,0" Height="137" VerticalAlignment="Top" HorizontalAlignment="Left" Width="262">
  <Button.Effect>
    <DropShadowEffect BlurRadius="37" ShadowDepth="20"/>
  </Button.Effect>
</Button>
```
XAML Attached-Property Syntax

- XAML also makes use of a concept termed ‘attached property’ syntax.
  - One use of attached properties is to make it possible for a sub-element to assign a property value on a parent element.
  - This is quite common when working with various control layout managers such as the Grid or Canvas.
  - Generally speaking, the template looks like so

```xml
<ParentType>
  <ChildType ParentType.PropertyName = "Value">
  </ChildType>
</ParentType>
```
Here is some attached property syntax which positions a set of controls in a Grid layout manager.

- Notice that the Grid is divided into a set or rows and columns.
- Each control declares where it wishes to be positioned via attached property syntax.
- Chapter 2 will examine Silverlight layout managers in more detail.

```xml
<UserControl ... >

<Grid Background="White" Width="450" Height="200" ShowGridLines="True">
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition Height="*" />
    <RowDefinition Height="Auto" />
  </Grid.RowDefinitions>
  <TextBlock Grid.Column="0" Grid.Row="0" Text="Testing, 1, 2, 3!"/>
  <TextBlock Grid.Column="0" Grid.Row="1" Text="Rectangle:"
    <Rectangle Grid.Column="1" Grid.Row="1" Fill="Purple" Width="250" Height="50"/>
  <TextBlock Grid.Row="3" Grid.Column="0" Text="Another test"/>
  <TextBlock Grid.Row="3" Grid.Column="1" Text="Final test"/>
</Grid>
</UserControl>
```
The Role of XAML Markup Extensions

- XAML supports various “markup extensions”.
  - In a nutshell, markup extensions are used to set property values which are not known until runtime.
  - For example, data binding operations require the use of markup extensions.
  - You will also make use of markup extensions when setting property values to named object resources (such as styles or templates).

- A XAML markup extension is encased within curly brackets at the time you assign a property value.
  - Here is a general template to follow. You will see specific examples as we progress through the course.
  - For the time being, simply remember that this syntax allows you to set a XAML property to a value not known until runtime.

```xml
<DefiningType DefiningTypeProperty = ""{x:MarkUpExtension Value}" />
</DefiningType>
```

- That wraps up your introduction to XAML and the core Silverlight programming model.
  - In your first lab, you will learn how to manipulate XAML at runtime, which can be quite powerful.
  - As well, you will learn the nuts-and-bolts of building Silverlight applications using Visual Studio 2010.
  - Other labs will begin to introduce Silverlight development using Expression Blend.

Lab Exercise: Introducing the Silverlight Programming Model
Chapter Summary

- Silverlight is a web-centric technology used to created rich interactive web plug-ins.

- Silverlight is based on several WPF technologies, including XAML, controls, animations, graphics and the use of code-files.

- Silverlight is OS and browser independent.
  - Windows and Mac OS X are supported by Microsoft.
  - IE, Firefox, Opera and Safari browsers are currently supported.
  - The Moonlight API enables Silverlight development and hosting on Linux based operating systems.
  - Silverlight is the core development model for Windows Phone 7.

- A Silverlight application in is represented at minimum by an Application derived class, and a UserControl to represent the UI.

- XAML is a key aspect of Silverlight development.
  - XAML allows you to represent the UI of an application.
  - XAML provides for a separation of concerns.
  - Tools such as Visual Studio 2010 and Expression Blend can be used to generate a majority of the required markup.