CHAPTER 11

Silverlight: A Beginner's Guide

"The noblest pleasure is the joy of understanding."
– Leonardo Da Vinci

In this chapter:

- Silverlight Basics
- Silverlight and the .NET Framework
- Hello Silverlight

This book is about Silverlight solutions. In this chapter we go behind the scenes and see what is meant by Silverlight plug-ins, what the components of Silverlight application are, and the role of the .NET Framework is.

Silverlight is ideally suited for the following kind of applications:

- Advanced Graphics and Animations
- Data Visualization with AJAX technology
- Rich Interactive games, Web gadgets, and Ad Banners
- Advanced Media Applications

This chapter is meant to give you a quick overview, and set a foundation for rest of the chapters. Let's begin.

Silverlight Basics

Silverlight is used to create rich internet applications. It is both a User Interface technology as well as a robust platform for creating web applications. It is integrated with .NET Platform so you can not only create interactive applications, but also enhance your current web applications to give your users a better experience.

Silverlight applications will run in a local machine but to run a Silverlight application on a web server, you need to add the extension .XAP with the MIME type application/x-silverlight in the server supported file types configuration.

Silverlight technology is comprised of four main parts:

- Silverlight Plug-in
- Silverlight Host, the Web Page
- Silverlight Application File (.XAP)
- The Interface language, Extensible Application Markup Language (XAML)

A Silverlight Plug-in is the engine that renders the Silverlight application in the browser, the host is the web page where Silverlight Application is hosted, and the Silverlight Application is the Internet Application which is developed using Microsoft Visual Studio and Expression Blend.

The core of the Silverlight technology is an Extensible Application Markup Language (XAML) which is a declarative programming language used to create rich vector graphics and animations and is used in the Silverlight Application file. Figure 11.1 shows an overview of the Silverlight Technology:

Figure 11.1 *XAML is the core of a Silverlight application.*

In the next few sections we will discuss these in more details.

Silverlight Plug-in

Silverlight is a Plug-in for a web browser. A Plug-in is an extension of an application created to enhance or extend functionality. Examples of other web browser plug-ins are Flash, Java applets, Windows Media Player, and QuickTime. To run a Silverlight application in a browser, users have to download and install the Silverlight plug-in first.

The Silverlight Plug-in is the complete environment where the Silverlight applications run. The Plug-in contains the XAML Parser, Media Pipeline, Presentation Engine, and a .NET programming layer that uses the embedded Common Language Runtime (CLR). The Plug-in also provides a way for Silverlight applications to communicate with the Browser. Figure 11.2 shows the components of Silverlight Plug-in.

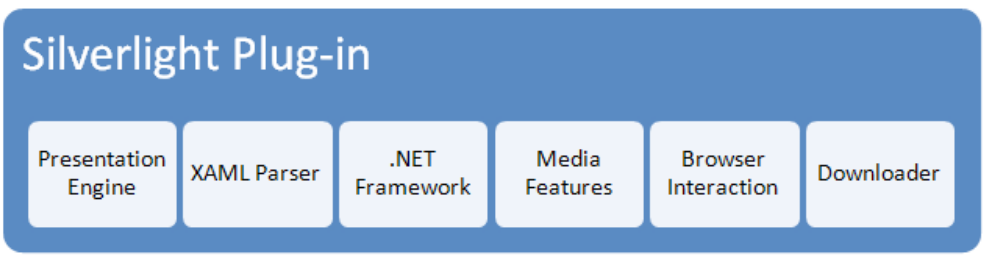

Figure 11.2 *The Plug-in contains the necessary components to render silverlight applications.*

The Silverlight Plug-in consists of the following sub components and features:

- The Presentation Engine with Multi Core support
  - Drawing Vector graphics and Shapes
  - PNG/JPG decoders
  - Image Cache, Text and Glyph cache
  - Animation System
  - 3D rendering engine
- The XAML Parser
- Subset of the .NET Framework
  - Core Common Language Runtime (CLR)
  - The .NET base class libraries
- Media features
  - Media Pipeline

    - Download and streaming
    - Audio/video decoders
- Browser Interaction
    - HTML Bridge
    - Mouse and Keyboard Input and Events
    - Ink Support
- The Downloader

The Silverlight Plug-in resides in the browser and is activated when you navigate to a web page that has the embedded code for Silverlight Application.

Steps for rendering Silverlight on a web page

The Plug-in acts in the following sequence when a user navigates to a Page with Silverlight, as shown in the figure 11.3:

Figure 11.3 *CLR is launched which then extract XAML and other assemblies.*

1. User opens a web page with Silverlight
2. The page has embedded code which refers to a XAML Application (.XAP) file
3. The Silverlight Plug-in is installed in the browser, and then downloads the .XAP file
4. Launch the embedded Common Language Runtime (CLR), the environment which executes and manages the Silverlight application
5. Extract the XAML Files and Assemblies from the XAML Application (.XAP) file
6. Load the .NET Assemblies
7. Create instances of the Silverlight Control that includes:
    1. Creating a User Interface Element Tree
    2. Managing the Layout of the User Interface
    3. Drawing the User Interface

Silverlight application is hosted on a web page using an HTML Object tag. We discuss how this works in the next section "Silverlight Host".

Silverlight Host

Microsoft Visual Studio and Expression Blend both allow you to create Silverlight applications. These applications essentially create a XAML Application file (.XAP) file, referenced by a link in an object tag.

There are three ways you can host a Silverlight Application File in a web page

1. Hosting the Silverlight in any HTML Page using <object> tag
2. Hosting in an ASPX Page using a Silverlight Control
3. Hosting in an ASPX Page using a MediaPlayer Control

Using an <object> Tag is similar to embedding any ActiveX control like a Media Player or a Flash File. Here is a sample code which embeds a Silverlight Object that link to a HelloWorld.xap.

```
<object data="data:application/x-silverlight-2,"
 type="application/x-silverlight-2">
       <param name="source" value="ClientBin/HelloWorld.xap"/>
       <param name="onError" value="onSilverlightError" />
       <param name="background" value="white" />
       <param name="minRuntimeVersion" value="4.0.50826.0" />
       <param name="autoUpgrade" value="true" />
     </object>
```

Visual Studio allows creating web projects that uses Silverlight User Controls in a webform. This user control is eventually converted into an object tag as shown in the earlier example. Silverlight also allows using Media control inside a web form (ASPX).

Silverlight Controls can also be embedded into an existing or new Web application by dragging the user control from the Tool Box to the web page. To use these controls, you will also need a Script-Manager control, the control which manages script files. See figure 11.4.

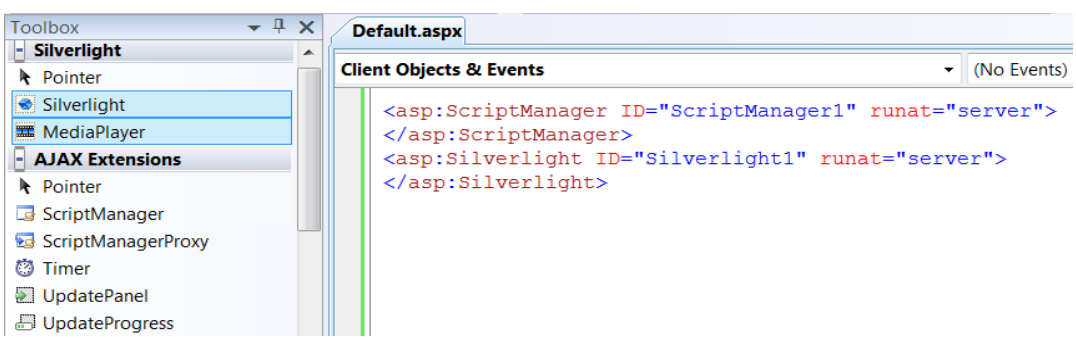

Figure 11.4 *Silverlight and MediaPlayer control are available in the Toolbox.*

Here is an example using a Silverlight Control in a Web Form (ASPX)

```
<asp:Silverlight ID="Xaml1" runat="server" Source="HelloWorld.xap" Minimum-
Version="4.0.50826.0" Width="100%" Height="100%" />
```

Using a MediaPlayer Control in a Web Form (ASPX)

```
<asp:MediaPlayer ID="MediaPlayer1" runat="server" Height="240px"
Width="320px"> </asp:MediaPlayer>
```

Silverlight Application File (.XAP)

A Silverlight Application file is a single compressed .ZIP file whose extension is changed to .XAP (XAML Application). It consists of a number of XAML files, .NET assemblies, and other resources. When you build a Silverlight application, Visual Studio compresses all relevant files into this single file.

The .XAP file is referenced inside the Silverlight Object tag in HTML and is loaded, parsed and executed by the Silverlight Plug-in. The .XAP file contains the actual logic and the interface of the Silverlight application. See the instruction for "Hello World" application at the end of the chapter to follow along as we develop the application.

Notice there are four files- App.xaml, App.xaml.cs, MainPage.xaml, and MainPage.xaml.cs. These are automatically created by Visual Studio. App.xaml and .cs files are used for application level properties and events. MainPage.xaml and .cs file are the files which contain the first visual of the application. Note the InitializeComponent method in the .cs files. This method combines the XAML file with the code.

To better understand the code interaction, we will open the HelloWorld Folder as a website inside Visual Studio. This will show us the hidden folders and files which are not shown in a Silverlight application.

In the Solution Explorer click on the **HelloWorld Solution** and click on **add existing website** and select the **HelloWorld Folder** and click **ok.** See figure 11.5 for an updated explorer view. In the Website view you will find two new folders:

- *Obj* folder, used for intermediate processing
- *Bin* folder, which stores the binaries after the project is compiled

Figure 11.5 shows a website view of the HelloWorld application with details of OBJ Folder(left) and Debug Folder(right).

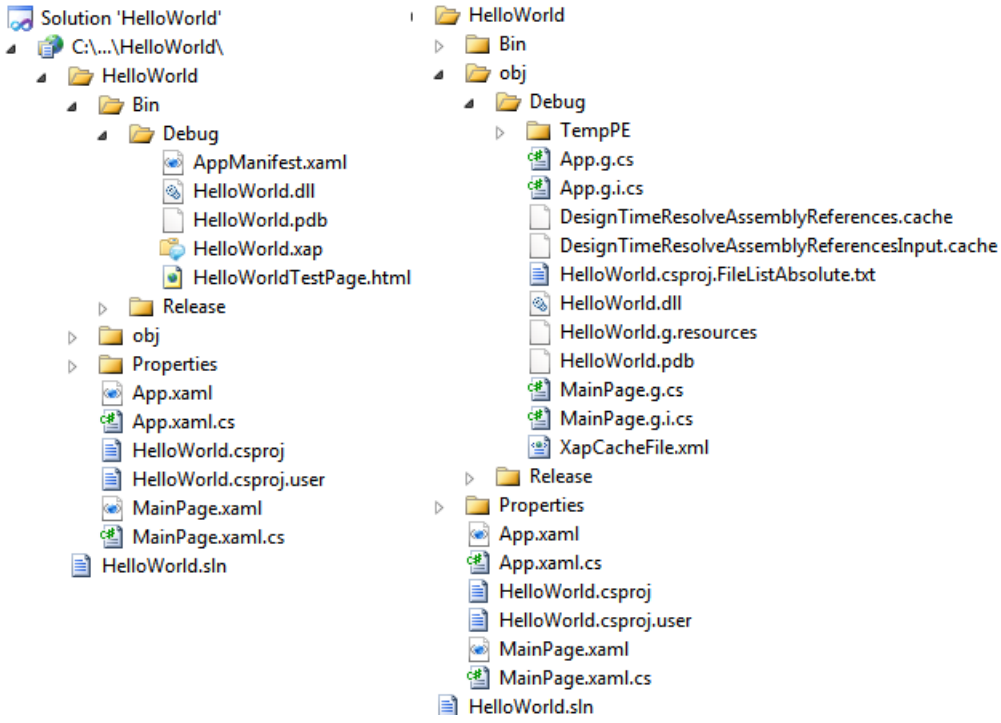

Figure 11.5 *The Obj/Debug Folder contains the intermediate files.*

The Obj/Debug Folder store objects and intermediate files before they are linked together:

- Page.g.cs and App.g.cs are the intermediate files corresponding to each xaml file. The "g" stands for generated. Each XAML file is converted into this partial class file. This file declare the initialize component method, and loads the corresponding xaml file as components.
- HelloWorld.g.resources file is the compressed binary form for all the xaml files which is also sometimes called BAML files
- The HelloWorld.dll, the assembly created by Visual Studio, is also copied to the BIN folder before compressing into the .XAP
- XAPCacheFile is the intermediate file for the Assembly Manifest
- HelloWorld.PDP is the projects related file that contains debugging and project state information
- FilelistAbsolute.txt, as the name suggest stores the absolute path of the files used before compiling them inside the dynamic link library

The Bin/Debug is the folder where the compiled files are stored. The files created as a result of the build process are:

- HelloWorld.XAP: File is the final compressed Silverlight application file which contains the HelloWorld.dll as well as the AppManifest.XAML. You can see this by changing the extension to .zip and opening it.
- TestPage.html: Host file which embeds the HelloWorld.XAP file
- The HelloWorld.dll: Assembly created by Visual Studio that contains the XAML files, required .NET assemblies, and other embedded resources related to the application
- AppManifest.xaml: Application manifest file which stores the Entry Point of the application, describes the names and source of assemblies that the application should bind to at the run time, and also contains metadata for assemblies used by the application.

To deploy your Silverlight application, you upload the TestPage.html and HelloWorld.XAP file to a web server. The HelloWorld.XAP file has all the needed files. HelloWorld Application has the following files in the .XAP:

- AppManifest.xaml
- HelloWorld.dll

A sophisticated Silverlight Application file will have other .NET assemblies, xaml files and other resources like images, and media files. When the Silverlight Plug-in loads a HelloWorld.zap file in a web browser, it reads the AppManifest file first to find out the entry point.

The entry point is the name of the dll which needs to be loaded first and is entered in the field EntryPointAssembly. In AppManifest.XAML, the value is HelloWorld. The manifest file will also have references to other required assemblies as AssemblyPart to be loaded by the application at run time.

```
<Deployment xmlns="http://schemas.microsoft.com/client/2007/de-
ployment" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
EntryPointAssembly="HelloWorld" EntryPointType="HelloWorld.App" Runti-
meVersion="2.0.31005.0">
  <Deployment.Parts>
    <AssemblyPart x:Name="HelloWorld" Source="HelloWorld.dll" />
    <AssemblyPart x:Name="OtherAssembly" Source=" OtherAssembly.dll" />
 </Deployment.Parts>
</Deployment>
```

In our example we have only one assembly, but Deployment.Parts can have list of all assemblies used in the application. Other assemblies will be loaded as required. Now let's look at the HelloWorld.dll. Figure 11.6 shows the HelloWorld.dll inside a .NET Reflector

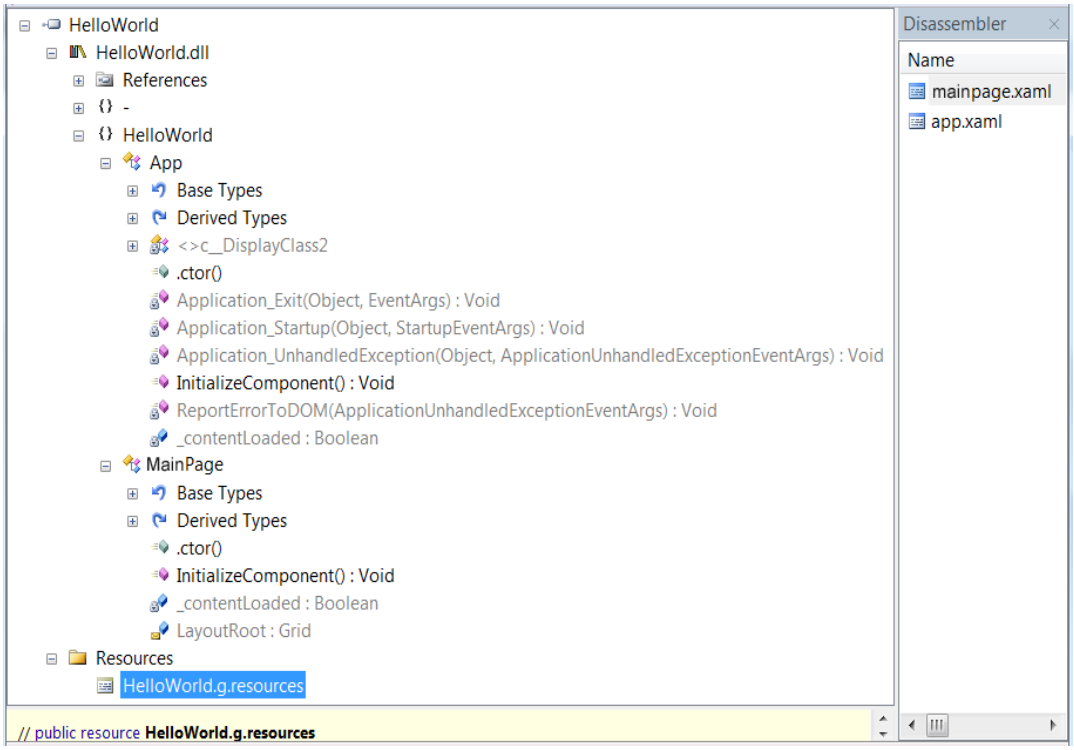

Figure 11.6 *The XAML files are stored in the resource section inside the HelloWorld assembly.*

The figure 11.6 shows the inside view of the HelloWorld.dll. Note that the DLL file has kept the XAML files in the resources and the code behind files becomes the standard methods of the Silverlight application.

The Silverlight plug-in follows the following sequence of events:

1. The XAP file is extracted
2. AppManifest.XAML file is consulted for the entry point assembly
3. The entry point assembly is loaded (in our case HelloWorld.dll)
    1. Application InitializeComponent is called
    2. Initialize component links and loads App.xaml which load application resources
    3. Application Startup method is called
    4. Startup method sets Application's RootVisual to Page object
        1. A new instance of Page is created

2. Page's InitializeComponent is called

5. MainPage.xaml is linked and loaded

6. Silverlight User interface is rendered based on MainPage.xaml

4. Web page displays the content of MainPage.xaml

When you create a new Silverlight application, MainPage.xaml becomes the default visual interface. In our example, "Hello World" text block in the MainPage.xaml is displayed. What makes MainPage.xaml rich is the vector quality of the interface markup language. Try changing the font size to 34, and notice how it scales without loss of quality. Thanks to eXtensible Application Markup Language (XAML), which we are going to discuss in the next section.

Silverlight XAML

Silverlight XAML (eXtensible Application Markup Language) is an interface markup language which has a visual file (MainPage.xaml) and a .NET code behind file (MainPage.xaml.cs) for logic. The visual part is an advanced HTML, and the code behind file brings the robustness of .NET type safe, managed environment. These two parts make XAML simple and a powerful language.

XAML = MainPage.xaml + MainPage.xaml.cs

The visual part is extended from xml and allows you to declaratively define your User Interface. It is straight forward, you have fields, attributes and properties and you get what you declare. MainPage.xaml.cs has application logic and event declarations. Let's see each of these in more details.

Silverlight XAML can be divided into two parts:

- Extensible Markup languages file (Visual Part)
- XAML .NET files (Application Part)

Extensible Markup Language files (Visual Part)

XAML is much more advanced than HTML, it lets you declare tags for objects, shapes and allows URL linking to jpg, png images and other multimedia files. While HTML parsing is done at runtime, XAML markup is compiled.

The object and shapes you declare in XAML are vector images and can scale without loss of quality. It also supports advanced Anti-alias features for sharp image borders and 2-D animations using story board tags. Animations in XAML are also declared-no programming is involved.

XAML Markup files support:

- Vector graphics
- Tag based 2-D animations
- Allow linking images and multimedia files

The second aspect of XAML is extensibility. XAML is based on XML, so apart from the tags available in the default xml namespace defined for objects and shapes, you can create your own namespace and use your own tags. In fact it's the extensibility of the Silverlight XAML file which allows for defining a code behind file.

Let's look at the MainPage.xaml we created in HelloWorld application.

```
<UserControl x:Class="HelloWorld.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"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400">

    <Grid x:Name="LayoutRoot" Background="White">
       <TextBlock x:Name="TextHello" FontSize="34" Margin="80">
        Hello World !</TextBlock>
    </Grid>
</UserControl>
```

Note the following four elements:

1. Root element UserControl
2. The significance of x:Class
3. Mapping (xmlns and xmlns:x) and XAML Namespace
4. The Grid and TextBlock Object

Let's talk about each of these to have a better understanding of the embedded code.

Root Element

UserControl is the root element of the XAML file. Silverlight allow two kinds of XAML templates in a silverlight project, an Application template and a User Control template with corresponding root elements.

A Silverlight project will have these two default files: App.xaml (Application) and MainPage.xaml (UserControl). The HelloWorld program had one application file (App.xaml) which uses one User-Control (MainPage.xaml) for display but you can have multiple templates in the same project. See Figure 13.7 for the class diagram.

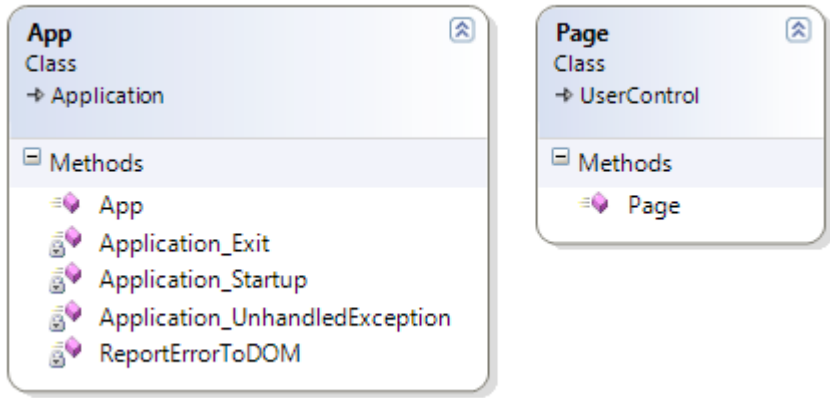

Figure 11.7 *The App class deals with the application as a whole.*

App class deals with the application as a whole and Page class as single interface inside the application. At any point of time the root elements of Silverlight is set to a Page class. By default this is set in the Application Startup event.

```
private void Application_Startup(object sender, StartupEventArgs e){
this.RootVisual = new Page1(); }
```

To change the user interface you will need to change the RootVisual to a different Page.

X:Class

The X: Class attribute allows you to add your namespace in the XAML file. The attribute declares the value of the namespace and the class. By default every MainPage.xaml file is associated with a Page class which is the code behind file. You can use this attribute to add more namespaces. The compiler use this values to generate the intermediate partial class (Page.g.cs) which is later combined with the code behind file. This attribute links the Visual XAML with the code behind file.

```
//  For x:Class="HelloWorld.Page" Namespace, class generated is shown
namespace HelloWorld { class Page : UserControl {} }
```

Namespace is the Assembly name, and since this is an attribute of a UserControl the Page class, it is inherited from the assembly.

xmlns and xmlns:x

In the root element we saw two attribute declarations: xmlns and xmlns:x. These attributes indicate to the XAML processor which XML Namespaces (xmlns) contain the element definitions the markup will reference. Let's go through more details of the XML Namespace.

XML Namespace

XML Namespace is a context associated with a set of XML elements and attribute, and are defined by a unique identifier. When you define an XML Namespace in the root of an XML files, you declare that the objects defined in the file will follow predefined language elements. XML Namespace ensures uniqueness of elements and their attributes Names in an XML file.

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

As we can see, the xmlns identifier is a URI (Uniform Resource Identifier) but is not required to point to an actual resource.

In Silverlight, this is meant for XAML files which are an extended version of XML so it is also called XAML Namespace. XAML Namespace defines many commonly-used features that are necessary for basic functionalities. It also defines a set of elements, attributes and properties which are accessible through the XAML processor.

The Default Namespace (Xmlns)

The xmlns attribute indicates the default XML namespace. All the elements which are specified without a prefix map to the default namespace. This includes all the UI elements like Grid, TextBlock, vector, and Animation objects.

The XML Namespace http://schemas.microsoft.com/winfx/2006/xaml/presentation is the namespace for Windows Presentation Foundation (WPF). The Silverlight technology originated from WPF so the first namespace defaults to the WPF namespace.

These elements are defined in the PresentationFramework.dll

The XAML Language Namespace (xmlns:x)

The second namespace is specific to XAML language and is mapped it. Every element or attribute which needs to use this namespace have to use the prefix "x:". We saw "x:Name" "x:Class" attributes in the XAML example above, these two attributes uses the XAML language namespace. So in the following example, although the TextBlock uses the default namespace, Grid's X:Name attribute is specific to the XAML language namespace.

```
<Grid x:Name="LayoutRoot" Background="White">
<TextBlock Margin="120" FontSize="20">Hello World!</TextBlock>
</Grid>
```

Figure 11.8 shows the two namespaces with common elements

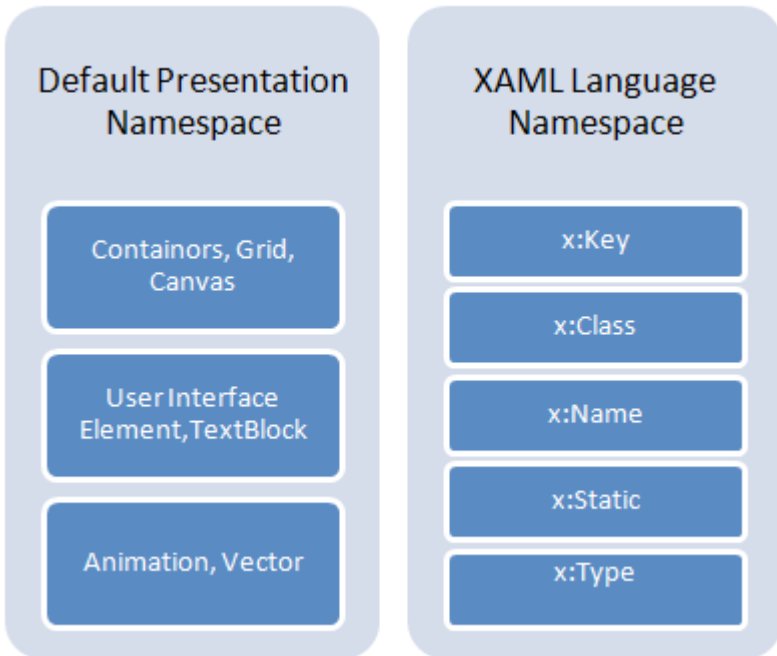

Figure 11.8 *The Namespace contains the element definition.*

The Grid and the TextBlock

A Grid object in XAML is similar to a Table tag and TextBlock and a label tag in HTML. By Default the Grid has one row and one column. We entered "Hello World" with FontSize 24 in a TextBlock inside a Grid. In HTML, this is similar to a label with text inside a borderless table with one row and one column.

The difference is the quality of the text block is better in XAML because it is in vector. The same text in HTML will not scale with that quality.

XAML .NET Files (Application Part)

Silverlight XAML is also a powerful application and treated as a .NET class by Common Language Runtime (CLR). There are three aspects to the application part of XAML:

- MainPage.xaml.cs, .NET code behind file is attached with the XAML file.
- Page.g.cs, a partial class file which is generated for the XAML file (MainPage.xaml)
- A BAML (Binary Application Markup Language) file is created based on the XAML file and stored as a resource (HelloWorld.g.resources) in Assembly.

MainPage.xaml.cs

A .NET code behind file is created for each XAML file. The XAML file uses the X:Class attribute to attach this file. The code behind file contains functions and methods. XAML allows declaring event names in the attributes, but the logic for the events is declared in this code behind file.

Page.baml

Binary Application Markup Language (BAML) file is an intermediate file which becomes a part of the Resources in the Assembly. It is a binary representation of the object hierarchy and properties defined in the source XAML file. It has been pre-tokenized (broken into discrete elements) so at runtime loading of BAML from the resource file is much faster than loading a xaml file. This allows XAML to retain the ease of XML during development without performance overhead at the runtime.

Page.g.cs

During build process, an intermediate partial class file is generated with an extension .g.cs for every xaml file. The Roles of Page.g.cs includes the following:

- It defines a field for every element with an X:name attribute

```
internal System.Windows.Controls.Grid LayoutRoot;
internal System.Windows.Controls.TextBlock TextHello;
```

- It declares the InitializeComponent method which creates the tree of Object by calling LoadComponent() and loads the MainPage.xaml from the Assembly resource

```
System.Windows.Application.LoadComponent(this, new System.Uri("/
HelloWorld;component/MainPage.xaml", System.UriKind.Relative));
```

- The InitializeComponent method also links the elements created with attribute x:Name to their respective names so that you can directly use them. The Field LayoutRoot and TextHello will be set to the value of the grid and textbox that was created for it at that time.

```
this.LayoutRoot = ((System.Windows.Controls.Grid)(this.
FindName("LayoutRoot")));
this.TextHello = ((System.Windows.Controls.TextBlock)(this.
FindName("TextHello")));
```

Page.g.cs file is combined with the code behind file MainPage.xaml.cs at runtime to create a single

class.

Page Class = MainPage.xaml + MainPage.xaml.cs becomes,
Page Class = Page.g.cs + MainPage.xaml.cs + HelloWorld.g.resources

The Build process takes both visual and application part and combines it into a single assembly (HelloWorld.dll). This separation of markup and application in the XAML provides easier development and robustness of .NET. Figure 13.9 summarizes the three aspects of XAML.

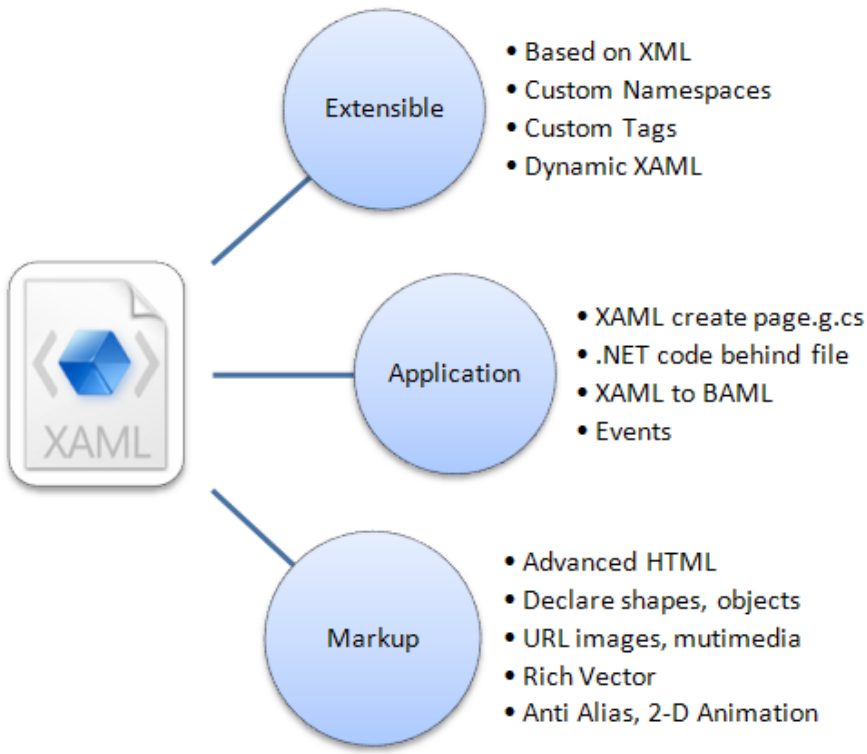

Figure 11.9 *XAML is an Extensible Application Markup Language.*

Silverlight and .NET Framework

The real power of Silverlight comes from .NET Framework. The .NET support in Silverlight ranges from base class library, to advanced internet features like AJAX and LINQ, available in .NET Framework 3.5.All of these features are managed through an embedded Common Language Runtime(CLR) in the Silverlight Plug-in.

The .NET Application support in Silverlight can be divided into two parts:

- Embedded Common Language Runtime (CLR)
- .NET Framework Libraries

The .NET Core Common Language Runtime resides with the Silverlight Plug-in the browser and is installed with Silverlight Plug-in. The Core CLR is responsible for the execution of the Silverlight application in the browser. The .Net Framework Libraries are required for the rendering of the Silverlight application, and all other functionalities.

Understand the Embedded CLR

The .NET Embedded CLR is the engine responsible for managing the .NET part of the Silverlight application. The CLR instantiates the Silverlight Application, loads it in the memory, renders it using the presentation framework libraries and dynamically links it with the networking libraries for the data and internet related features. It manages memory and ensures type safety and security features. The Embedded CLR provides the following features:

- Automatic Memory Management
- Garbage Collection
- Just in Time Compiler
- Managed Exception handling
- Type Safety
- Security Features
- Threading

These features are analogous to the CLR used for .NET applications. We will see more details on these in examples. The next part is the .NET Framework libraries available for Silverlight.

.NET Framework Libraries for Silverlight

The .NET Framework libraries supported by Silverlight are Base Class Libraries, Generics, Collections, Regular Expression, LINQ and Threading.

.NET Framework Libraries

- Base Class library
- Presentation Foundation
- Data framework

- Communication Foundation
- Additional Libraries Specific to Silverlight

Base Class library

The base class library in Silverlight is meant to provide all the basic features of .NET applications, like input/output (IO) operations, text, Regular expressions, collections, and Generics. Here is the list of namespaces supported by base class libraries:

- System for the core object
- System.IO handles input and output
- System.Text support storage and manipulation of text
- System.Text.RegularExpressions for Regular expression support
- System.Resources allows embedding of resource in the assembly
- System.Collections.Generics allows generics in Silverlight
- System.Cryptography enforces Security features
- System.Globalization for multi language support
- System.Reflections for assembly related information

Presentation Foundation

Presentation foundation consists of all the libraries used to render the Silverlight application on the webpage. This include libraries related to Markup, Shapes, Vector, Animations, and libraries related for User Controls and multimedia files. This also includes Windows library for Silverlight applications and deployment. Here is the list of namespaces:

- System.Windows for application and deployment
- System.Windows.Shapes for Ellipse, Line, Path and Polygons
- System.Windows.Documents for Glyphs and linebreaks
- System.Windows.Media supports brush and transformations
- System.Windows.Media.Animation for Animation and storyboards
- System.Windows.Threading supports Dispatcher and DispatcherTimer
- System.Windows.Browser for communicating with the HTML Browser
- System.Windows.Controls Standard controls and UserControls

Data framework

The next set is for .NET libraries related to XML, JSON data and LINQ

- System.XML provides XMLReader and XMLWriter
- System.Runtime.Serialization provides data and XML Serializer
- System.Runtime.Serialization.JSON supports JSON data
- System.Linq for advanced LINQ support
- System.XML.Linq for LINQ to XML conversions

Communication Foundation

Communication foundation relates to data communication in the Web. It is meant to support AJAX technologies, web services and RSS, Atom feed formats.

- System.Net provides Ajax, HTTPWebRequest, WebClient and Sockets
- System.ServiceModel supports web service client and Service reference
- System.ServiceModel.Channel for binding and message
- System.ServiceModel.Syndications used for RSS and Atom feed

Additional Libraries Specific to Silverlight

Silverlight provides additional functionalities for creating rich and interactive applications:

- System.IO.IsolatedStorage
- DeepZoom through MultiScale class in System.Windows.Controls
- System.ComponentModel for Background Worker (Async. Programming)
- HTML managed code interaction through System.Windows.Browser
- XML, JSON and POX (Plain Old XML) data Support through System.XML
- Dynamic Language Runtime which supports IronRuby and IronPython

Figure 11.10 gives an overview of the .NET library available in Microsoft Silverlight.

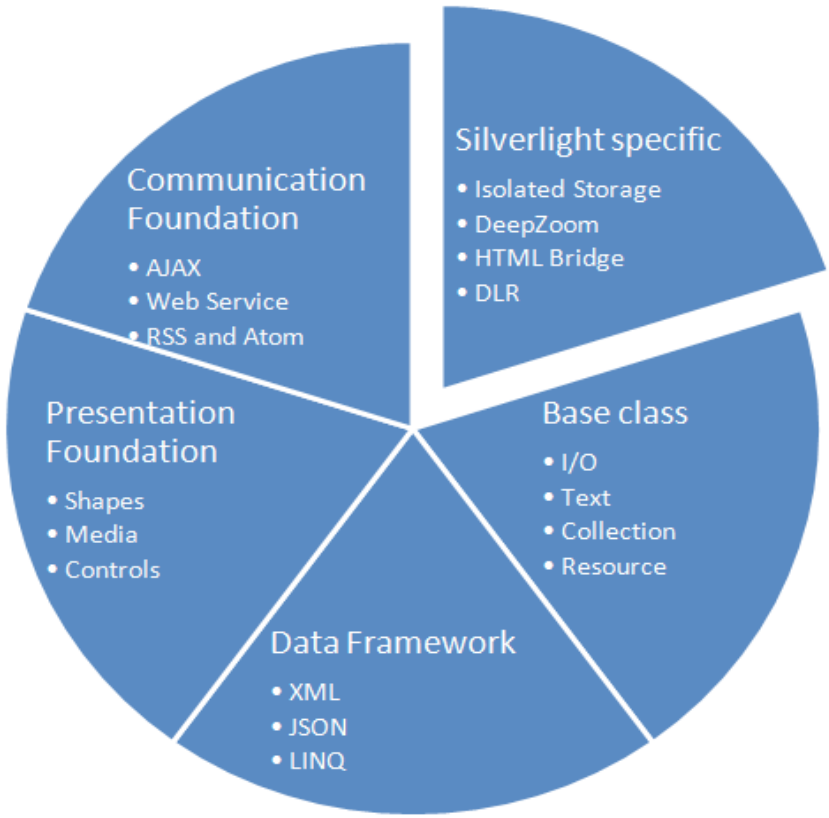

Figure 11.10 *Silverlight comprises the best of the .Net Framework.*

Create Silverlight Application

Let's start with two simple programs.

1. Hello World – Basic application
2. Silverlight Rocks – Event driven Silverlight application with dynamic resources

The first example application has already been discussed in the chapter, and the second example is to give you a taste of event driven Silverlight programming.

Creating the Hello World Application

Problem

How to create a Hello World in Silverlight?

Solution

The Hello World application is normally seen as the first and the simplest program. The program will demonstrate the methods exposed by the App class which is derived from Application class. See figure 11.11.

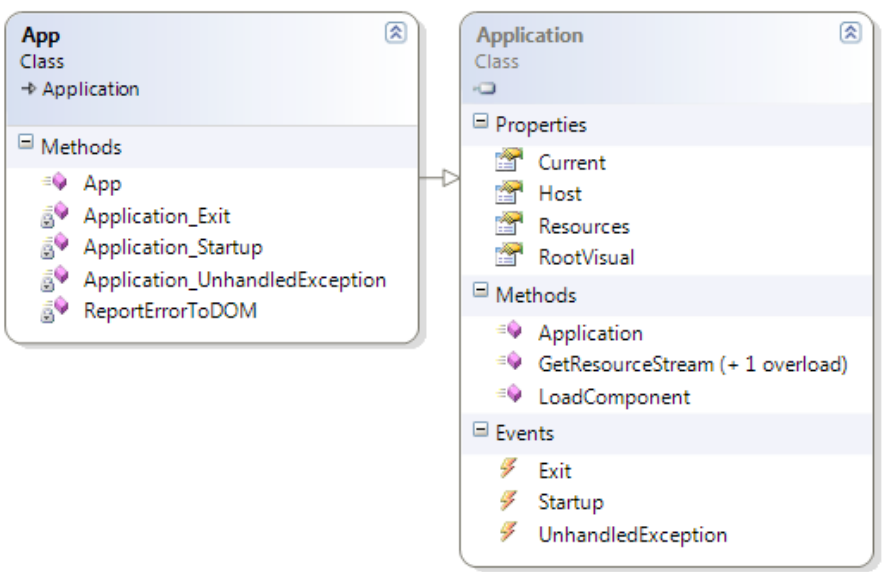

Figure 11.11 *The RootVisual property of the Application class defines the Visual element of the Silverlight application.*

Steps to create the HelloWorld Silverlight Application:

1. Open Visual Studio, click on File, New, Project, and select Silverlight in the Project Type and select Silverlight Application in the template. Enter HelloWorld in the Name and Solution field and click ok.
2. Select automatically generate a test page to host Silverlight and click ok.
3. Open the MainPage.xaml file in the editor add a TextBlock control inside the Grid tag and

enter Hello World as shown below

```
<Grid x:Name="LayoutRoot" Background="AliceBlue">
<TextBlock FontSize="24" Margin="100">Hello World !</TextBlock>
</Grid>
```

4. Press F6 to build Solution and CTRL F5 to run the test page with Silverlight inside. See figure 11.12.

Figure 11.12 *The Hello World Text scales without loss of quality.*

Once you have created the Hello World application, look into the application files inside Visual Studio. The important point here is the MainPage class which is used to set the RootVisual property of the App.cs. This is done in the Application_Startup method. The application has the following four public methods: Startup, Exit, UnhandledException and ReportErrortoDOM. Also note the properties of the base Application class.

XAML

App.xaml

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

MainPage.xaml

```
<UserControl x:Class="HelloWorld.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="400" Height="300">
    <Grid x:Name="LayoutRoot" Background="AliceBlue">
      <TextBlock Margin="120" FontSize="20">Hello World!</TextBlock>
    </Grid>
</UserControl>
```

Code

App.xaml.cs

```
using System;
using System.Windows;
namespace HelloWorld {
public partial class App : Application {
public App() {
 this.Startup += this.Application_Startup;
 this.Exit += this.Application_Exit;
 this.UnhandledException += this.Application_UnhandledException;
 InitializeComponent();
 }
private void Application_Startup(object sender, StartupEventArgs e) {
 this.RootVisual = new Page();
 }
private void Application_Exit(object sender, EventArgs e){}
private void Application_UnhandledException(object sender,
 ApplicationUnhandledExceptionEventArgs e) {
 if (!System.Diagnostics.Debugger.IsAttached)  {
 e.Handled = true;
 Deployment.Current.Dispatcher.BeginInvoke(delegate{ReportErrorToDOM;});
 }
}
private void ReportErrorToDOM(ApplicationUnhandledExceptionEventArgs e) {
 try  {
 string errorMsg = e.ExceptionObject.Message +
 e.ExceptionObject.StackTrace;
 errorMsg = errorMsg.Replace('"', '\'').Replace("\r\n", @"\n");
 System.Windows.Browser.HtmlPage.Window.Eval("throw new Error(\"Unhandled
 Error in Silverlight 2 Application " + errorMsg + "\");");
 }
 catch (Exception){}
 }
}
```

MainPage.xaml.cs

```
using System;
using System.Windows.Controls;
namespace HelloWorld {
public partial class Page : UserControl {
```

```
public Page() { InitializeComponent();}
 }
}
```

Create an Event Driven Application

Problem

How to create a simple silverlight application with events?

Solution

This program is meant to give you the next step for creating Silverlight-based applications. From now on we will concentrate only on the Page class (see figure 11.13)which is the visual element. In this example we will also see how to wire up events in a Silverlight User Interface and understand how to add and use resources.

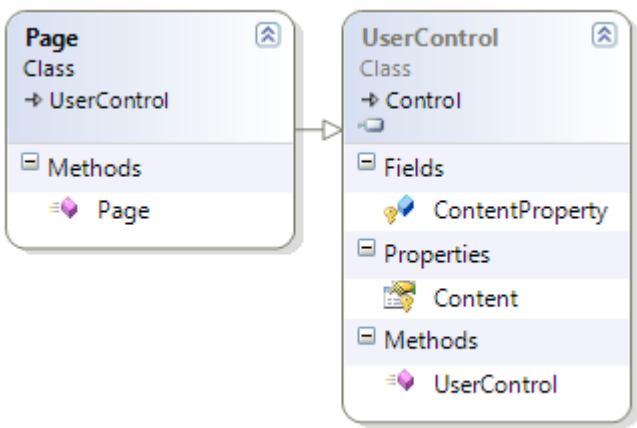

Figure 11.13 *A Page is a UserControl.*

The Silverlight Rocks program displays a smiley at the point where the user clicks on the User Interface (see figure 11.14) and gives an option to delete them individually or delete them all.

The application is meant to show you the event driven programming in Silverlight. We will look into more detail on Reading XAML and XML files in later chapters. There are two different colored Smiley's: One added as a Content (smiley.xaml Build Action= Content) and the other as a Resource (smileypink.xaml Build Action=Resource) in the Application.

The first thing to notice in the MainPage.xaml is the LayoutRoot_MouseLeftButtonUp. Every UIElement in Silverlight exposes the following methods for interaction:

- Loaded
- SizeChanged
- Layout Updated
- BindingValidationError

Along with the following events as shown in figure 11.14

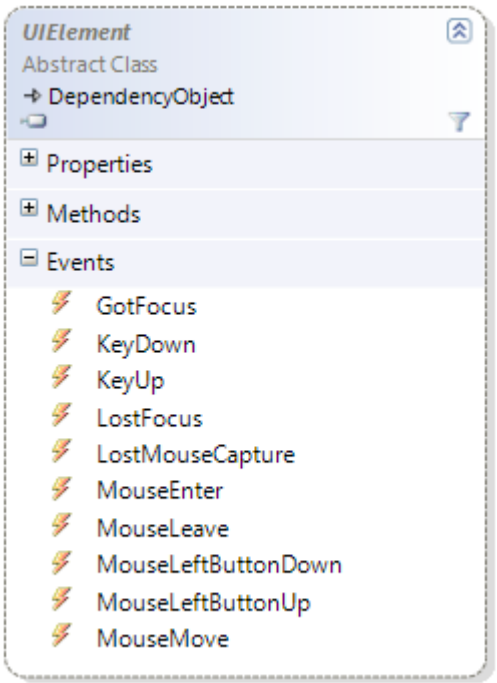

Figure 11.14 *A UIElement comes with a number of Events for input.*

All of these methods can be created dynamically in the code as well as at design time. This means that when the use clicks on the LayoutRoot element (canvas here) the event is fired with the sender object, and MouseButtonEventArgs (see code listing below). The Mouse Event Arguments gives the coordinates is used to position the dynamically created Smiley.

In the Add Smiley method, we randomly select a smiley either from the Content or from the Resource. The difference is Content Smiley is inside the XAP file, the resource Smiley is converted

into binary resource and is stored and retrieved from the Resource part of the HelloSilverlight.dll.

Other items of particular note: LayoutRoot.Children.Add and LayoutRoot.Children.Remove method which gives a standard way to add UIElement in the User Interface, and the JavaScript Alert message which is called using HtmlPage.Window.Alert and HtmlPage.Window.Confirm method.

Figure 11.15 *Silverlight supports alert boxes.*

XAML

```
<UserControl x:Class="HelloSilverlightWorld.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="600" Height="400">
 <Canvas x:Name="LayoutRoot" Background="AliceBlue"\
    MouseLeftButtonUp="LayoutRoot_MouseLeftButtonUp">
   <TextBlock x:Name="Hello" Canvas.Left="200" Canvas.Top="170"
    FontFamily="Arial" FontSize="24" Text="Silverlight Rocks !"
    Foreground="#FF000000"></TextBlock>
   <TextBlock x:Name="Reset" Canvas.Left="250" Canvas.Top="206"
    FontFamily="Arial" FontSize="12" Text="(Click Anywhere)"
    Foreground="#FF000000" ></TextBlock>
 </Canvas>
</UserControl>
```

Code

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Xml;
using System.Windows.Browser;
using System.Windows.Markup;
namespace HelloSilverlightWorld{
public partial class Page : UserControl {
 bool ResetFlag;
 public Page() { InitializeComponent(); }
 private void LayoutRoot_MouseLeftButtonUp(object sender,
   MouseButtonEventArgs e) {
  UIElement s = (UIElement)e.OriginalSource;
  Point MousePosition = e.GetPosition(LayoutRoot);
  AddSmiley(MousePosition.X, MousePosition.Y); }
 private void Smiley_MouseLeftButtonUp(object sender, MouseButtonEventArgs e){
  if (HtmlPage.Window.Confirm("Delete Smiley?")){ RemoveSmiley(sender); } }
 private void Reset_MouseLeftButtonUp(object sender, MouseButtonEventArgs e){
  RemoveAllSmiley();}
 private void AddSmiley(double x, double y){
  XmlReader xmlReader ;
  if ((int)x % 2==0) xmlReader = XmlReader.Create("files/smiley.xaml");
  else xmlReader =
  XmlReader.Create("/HelloSilverlight;component/files/smileypink.xaml");
  xmlReader.MoveToContent();
  UIElement MySmiley = (UIElement)XamlReader.Load(xmlReader.ReadOuterXml());
  MySmiley.SetValue(Canvas.TopProperty, y);
  MySmiley.SetValue(Canvas.LeftProperty, x);
  MySmiley.MouseLeftButtonDown += new
  MouseButtonEventHandler(Smiley_MouseLeftButtonUp);
  LayoutRoot.Children.Add(MySmiley);
  if (!ResetFlag) ResetText(); }
 private void ResetText() {  ResetFlag = true;
  Reset.Text = "(Click To Reset)";
  Reset.MouseLeftButtonDown += new
  MouseButtonEventHandler(Reset_MouseLeftButtonUp); }
 private void RemoveSmiley(Object sender)
  { LayoutRoot.Children.Remove((UIElement)sender); }
 private void RemoveAllSmiley() {
  if (HtmlPage.Window.Confirm("Delete All Smiley?")) {
  int TotalCount = LayoutRoot.Children.Count;
  for (int i = TotalCount - 1; i > 1; i--) {
  LayoutRoot.Children.RemoveAt(i); }
  ResetFlag = false;  Reset.Text = "(Click Anywhere)";
  Reset.MouseLeftButtonDown -= new
  MouseButtonEventHandler(Reset_MouseLeftButtonUp); }
 }
}
```

"Any sufficiently advanced technology is indistinguishable from magic."
– Arthur C. Clarke