Microsoft Silverlight Tutorial

Introduction to Microsoft Silverlight: A Practical Tutorial for Legacy Applications

Microsoft Silverlight was once a dominant plug-in technology for building rich internet applications (RIAs) with multimedia, graphics, and interactivity. Although its mainstream support ended in 2021 and modern web development has moved to HTML5, many enterprises still maintain Silverlight-based intranet applications, educational content, or media players. Understanding how to work with Silverlight remains valuable for developers tasked with maintaining or migrating such systems. This **Microsoft Silverlight Tutorial** offers a comprehensive, step-by-step guide covering core concepts, development environment setup, XAML basics, C# integration, and common use cases. By the

end, you will have a solid foundational knowledge to create or modify a Silverlight application.

What Is Microsoft Silverlight?

Silverlight is a browser plug-in that enabled rich client-side functionality similar to Adobe Flash. It supported vector graphics, animations, audio/video playback, and data binding. It uses the .NET Framework runtime through a subset called Silverlight Runtime. Developers could write code in C# or Visual Basic, design user interfaces with XAML (eXtensible Application Markup Language), and integrate media or Deep Zoom images. While Microsoft no longer encourages new Silverlight development, many legacy systems still rely on it.

Setting Up Your Silverlight Development Environment

Required Tools

- **Visual Studio 2019 (or earlier)** – Visual Studio 2019 includes Silverlight project templates. For newer versions

(2022), you may need to install the Silverlight extension manually.

- **Silverlight SDK** – Download from Microsoft archives (Silverlight 5.1 SDK).
- **Silverlight Developer Runtime** – Required to run Silverlight applications locally.
- **Web Browser with Silverlight Plug-in** – Internet Explorer 11 still supports it. Edge no longer supports NPAPI plug-ins, so use IE Mode or a dedicated test environment.

Installation Steps

1. Install Visual Studio 2019 (Community edition works).
2. Install the Silverlight SDK (run the installer, ensure "Silverlight Developer Runtime" is included).
3. Open Visual Studio and verify that "Silverlight Application" template appears under New Project > Visual C# > Silverlight.
4. If templates are missing, go to Tools > Extensions and Updates and search for "Silverlight Tools for Visual Studio 2019".

Your First Silverlight Application: A Step-by-Step Tutorial

Creating the Project

Launch Visual Studio, choose **File > New > Project**. Select **Silverlight Application**. Name your project, e.g., *MyFirstSilverlightApp*. In the dialog, choose "Silverlight 5" as target version. You can also opt to host the Silverlight application in an ASP.NET Web Application project, which is useful for testing.

Understanding the Project Structure

A typical Silverlight solution contains two projects:

- **SilverlightClient** – the actual Silverlight XAP file (compiled output). Contains `MainPage.xaml` and `App.xaml`.
- **SilverlightWeb** (if created) – an ASP.NET web project that hosts the test page with the `<object>` tag

embedding the Silverlight plug-in.

The `MainPage.xaml` file is where you design the UI using XAML. The code-behind file `MainPage.xaml.cs` contains event handlers and logic in C#.

Designing the User Interface with XAML

XAML is a declarative language for creating UI elements. Let's build a simple UI with a button and a text block.

```
<UserControl
x:Class="MyFirstSilverlightApp.MainPage"
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="White">
        <TextBlock
x:Name="MessageText"
                   Text="Hello,
Silverlight!"
                   FontSize="24"
HorizontalAlignment="Center"
```



```
VerticalAlignment="Center" />
        <Button Content="Click
Me"
                Width="120"
                Height="40"
HorizontalAlignment="Center"
VerticalAlignment="Bottom"
                Margin="0,0,0,30"
Click="Button_Click" />
    </Grid>
</UserControl>
```

Notice the `Click` event attribute points to an event handler in the code-behind.

Adding C# Logic

Open `MainPage.xaml.cs` and implement the button click event:

```
private void Button_Click(object
sender, RoutedEventArgs e)
{
    MessageText.Text = "You
clicked the button!";
}
```

This simple interaction demonstrates event handling and UI updates. Silverlight uses a subset of the .NET Framework; many

standard classes like System.Windows.Controls are available.

Working with Media: Playing Audio and Video

One of Silverlight's strong points is media playback. To add a media element:

```
<MediaElement
x:Name="MediaPlayer"
Source="myvideo.mp4"
              AutoPlay="False"
              Width="320"
Height="240" />
```

You can control playback via C#: `MediaPlayer.Play()`, `Pause()`, `Stop()`. Silverlight supports adaptive streaming (Smooth Streaming) and DRM (PlayReady). For legacy projects, this is still relevant.

Data Binding and MVVM in Silverlight

Silverlight supports the Model-View-ViewModel pattern, which is crucial for

maintainable applications. Data binding in XAML uses `{Binding Path}` syntax.

Simple Data Binding Example

Define a data source (a class) in C#:

```
public class Person :
INotifyPropertyChanged
{
    private string _name;
    public string Name
    {
        get { return _name; }
        set { _name = value;
OnPropertyChanged("Name"); }
    }
    // implement
INotifyPropertyChanged...
}
```

In XAML:

```
<TextBlock Text="{Binding Name}"
/>
```

Set DataContext in code-behind: `this.DataContext = new Person { Name = "Alice" };`

For two-way binding, use `Mode=TwoWay`.

This pattern is essential for forms and master-detail views.

Navigation and Page Management

Silverlight provides a `Frame` control for navigation similar to web pages. You can use `NavigationPage` or manually load UserControls. This is useful for multi-view applications.

Example Navigation Setup

```
<Frame x:Name="ContentFrame"
Source="/Views/HomePage.xaml" />
```

```
Add hyperlinks to navigate:
ContentFrame.Navigate(new
Uri("/Views/AboutPage.xaml",
UriKind.Relative));
```

Deployment and Hosting Silverlight Applications

The compiled Silverlight project produces a `.xap` file (a ZIP archive containing assemblies, manifests, and resources). You

need to host it on a web server and include the appropriate <object> tag in an HTML page. Example:

```
<object data="data:application/x-silverlight-2,"
        type="application/x-silverlight-2"
        width="100%"
height="100%">
    <param name="source"
value="ClientBin/MyApp.xap"/>
    <param name="onError"
value="onSilverlightError"/>
    <param name="background"
value="white"/>
    <a
href="http://go.microsoft.com/fwlink/?LinkID=149156" style="text-decoration: none;">
        <img
src="http://go.microsoft.com/fwlink/?LinkId=108181"
             alt="Get Microsoft
Silverlight" style="border-style:
none"/>
    </a>
</object>
```

Note: Modern browsers no longer support

the Silverlight plug-in. For internal deployment, you may need to configure Internet Explorer 11 or use Edge with IE mode. This is a critical limitation to consider.

Common Challenges and Workarounds

- **Compatibility**: Silverlight only works on Windows (and briefly Mac). No mobile support.
- **Security**: Outdated plug-in poses risks; use only in isolated intranets.
- **Debugging**: Visual Studio's Silverlight debugger can be unstable. Use trace output and breakpoints carefully.
- **Performance**: Large XAP files can cause slow load times. Use application library caching or split into multiple XAPs.
- **Migration**: Plan to replace with HTML5/JavaScript or Blazor if possible.

Optimizing Silverlight Applications for Legacy Maintenance

When working with an existing Silverlight codebase, follow these best practices:

- Keep XAML files organized with consistent naming conventions.
- Use asynchronous WCF or REST calls with `WebClient` or `HttpClient` (Silverlight's `HttpClient` is a subset).
- Implement logging (e.g., `System.Diagnostics.Trace`) to capture errors without relying on the UI.
- Document any dependencies on specific Silverlight features (e.g., Deep Zoom, out-of-browser mode).
- Plan incremental migration: gradually replace Silverlight modules with Angular/React components, using JavaScript interop or iFrame embedding.

Conclusion

Microsoft Silverlight may be a legacy technology, but understanding it remains important for developers supporting existing enterprise systems. This tutorial has walked you through setting up a development environment, creating a basic XAML UI, adding C# logic, handling media, implementing data binding, and deploying the application. While the industry has moved to modern web standards, the concepts you learn here—XAML, event-driven programming, and the .NET Framework—provide a solid foundation that transfers to other Microsoft technologies like WPF and UWP. For any Silverlight maintenance or migration project, use this guide as a starting point. Always keep security and compatibility in mind, and prioritize a gradual migration to a modern platform.

