INTRODUCING MICROSOFT SILVERLIGHT

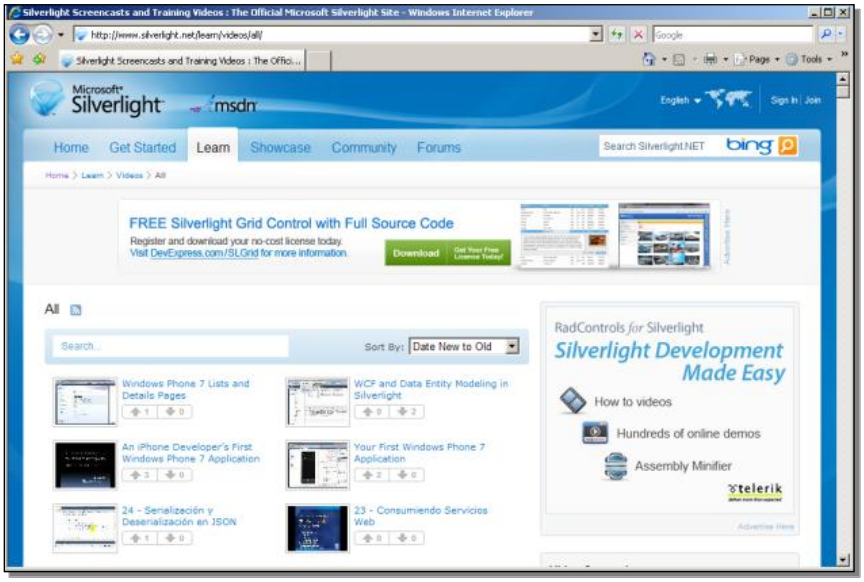
John Sadd
Fellow and OpenEdge Evangelist
Document Version 1.0
December 2010

DISCLAIMER

Certain portions of this document contain information about Progress Software Corporation's plans for future product development and overall business strategies. Such information is proprietary and confidential to Progress Software Corporation and may be used by you solely in accordance with the terms and conditions specified in the PSDN Online (http://www.psdn.com) Terms of Use (http://psdn.progress.com/terms/index.ssp). Progress Software Corporation reserves the right, in its sole discretion, to modify or abandon without notice any of the plans described herein pertaining to future development and/or business development strategies. Any reference to third party software and/or features is intended for illustration purposes only. Progress Software Corporation does not endorse or sponsor such third parties or software.

This is one of a series of presentations on different technology choices you can make to create a modern user interface for your OpenEdge application. This one introduces Silverlight, a major UI technology from Microsoft. And perhaps the most evident characteristic of Silverlight is the fact that it is a Microsoft technology. You can take a look at the website at www.silverlight.net. Here you get a flavor for the wealth of supporting materials for Silverlight, which reflects its status as a major Microsoft initiative.

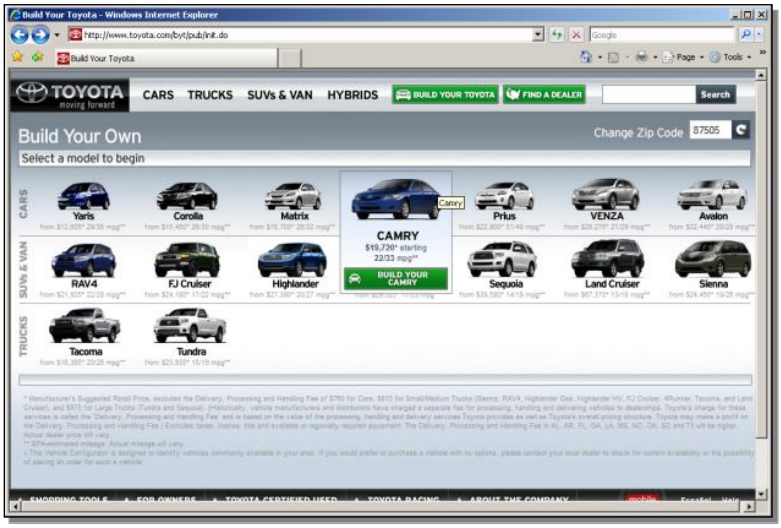
For instance, there is a link to a large collection of video tutorials somewhat like the one this paper is based on.



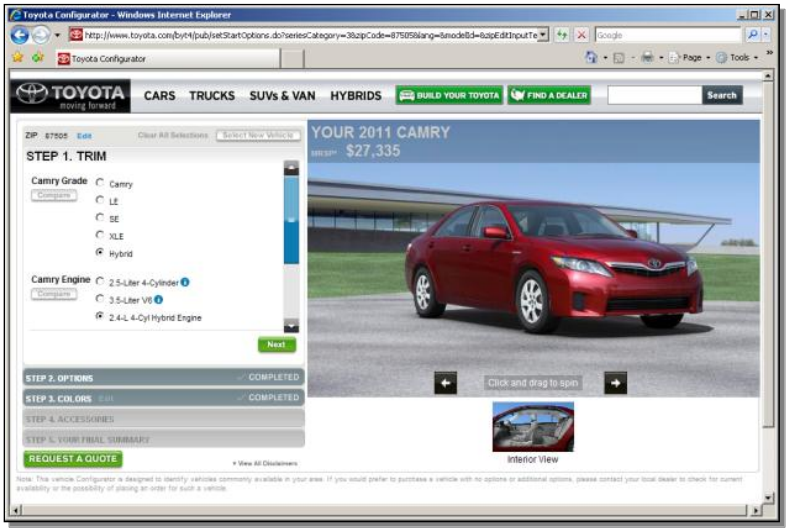
And there are hundreds of them, as well as extensive documentation, technical papers, user support groups, and so forth. So this is the first obvious message about Silverlight: it's a major technology with the power of Microsoft and Microsoft's user community behind it.

You can build all kinds of applications using Silverlight for the front-end, but there's a serious focus on graphical design, animation and media in a lot of Silverlight applications and tutorials. Here at the website, for instance, there are a number of sample applications you can run. One is a Microsoft application called Visit Solution City, which integrates video into the app, a much-promoted strength of Silverlight as a user interface platform. Again, this kind of thing may not be an important requirement for your business application:

The next screenshots show another Silverlight front-end to a type of consumer application you may have seen. This is one of those where you can build and price your own virtual car. And again, you can see the extensive use of graphics and animation here as part of the user interface.

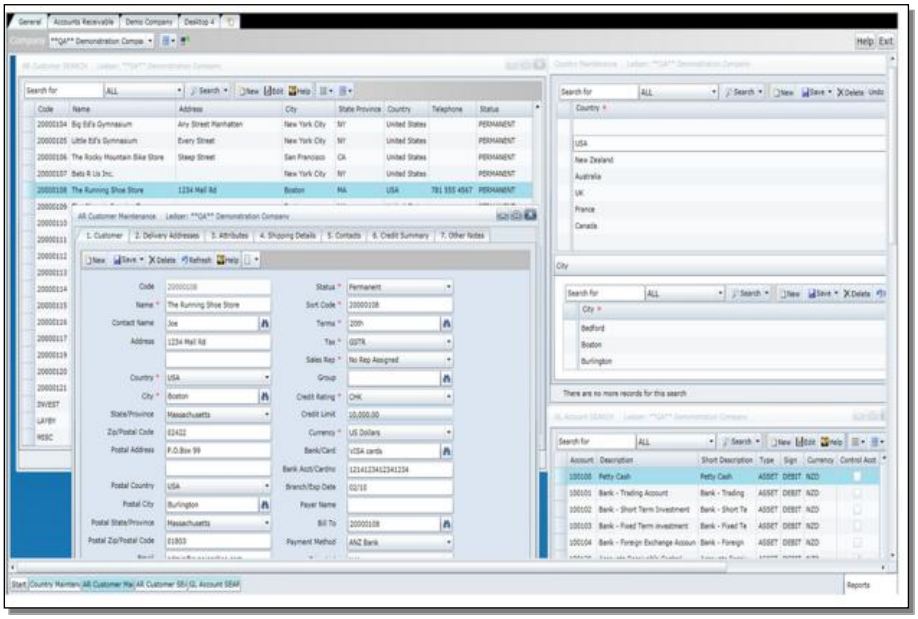


After you've selected a type of car, you can spin it around to see it from different angles.



And you can select a different color, and see the car in that color. This is a typical example of a highly design-oriented and graphical application. But Silverlight is a powerful platform for developing more standard business applications as well, with controls like grids that rival the look and feel of true desktop controls like those you can get with a GUI for .NET application.

For example, below is a screen shot from an actual OpenEdge application built by one of our application partners in OpenEdge 10.2B using Silverlight, and using WebSpeed to communicate with ABL business logic on the backend. WebSpeed and Web services are both options for an OpenEdge application in 10.2B, as are the native RIA services that Microsoft supports. Other sessions and papers in this series discuss some of these options.

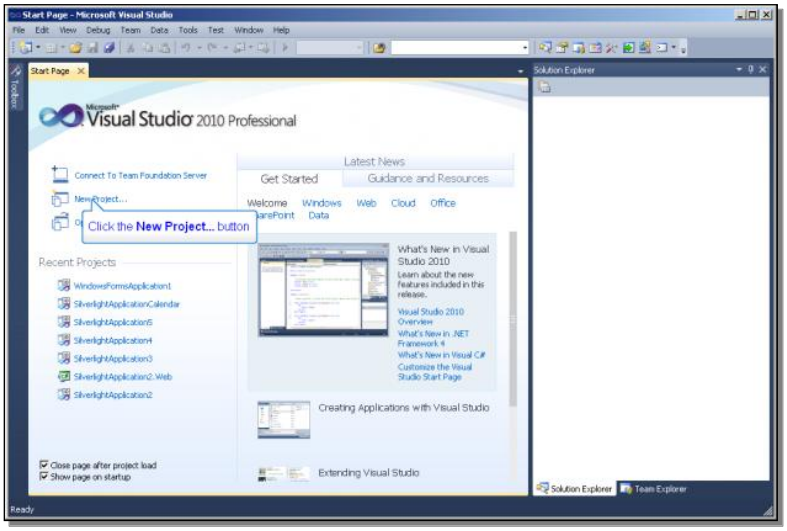


So apart from the Microsoft support, how does Progress Software position Silverlight as a UI choice?

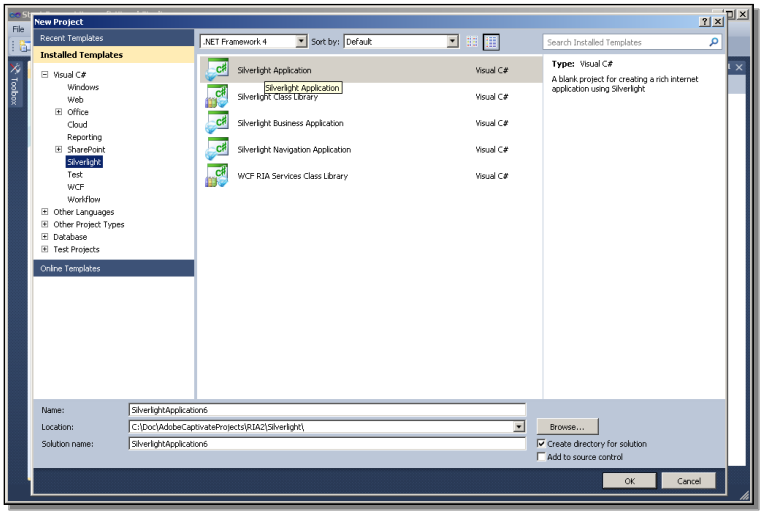
If an open-source library based on AJAX like ExtJS, with no client-side footprint at all, represents one end of the spectrum, and a true desktop application platform like OpenEdge GUI for .NET with Infragistics controls installed on the client machine represents the other, then Silverlight can be seen as an intermediary between them. It's a browser plug-in, so it's not zero footprint, but it requires only a quick and easy plug-in install the first time the client machine runs a Silverlight application. It's browser-based rather than running natively on the desktop, and there may be some limitations on the platforms it can run on, but it also provides user interface controls and behavior that rival the desktop app and are more powerful than the AJAX solutions.

So let's take a look at a bit of what's involved in building an application with Silverlight. First of all, because this is a Microsoft product, you're going to be doing development in Visual Studio. So if this is an environment that is familiar to your development staff, that can certainly be a factor in your choice.

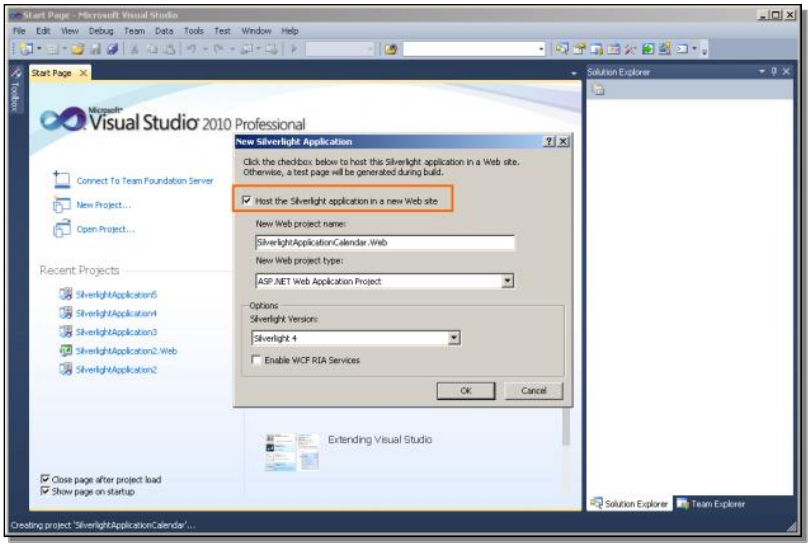
You start by creating a **New Project** in Visual Studio:



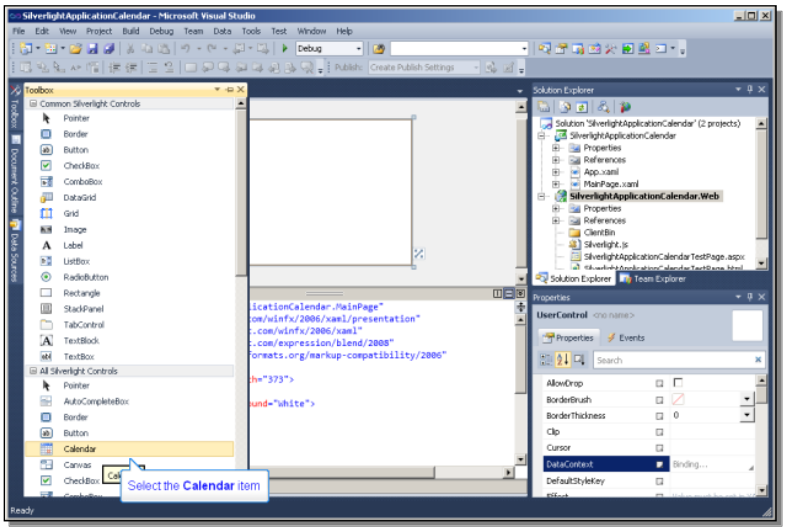
From a set of standard templates you can use as a starting point, select **Silverlight**, and then a basic **Silverlight Application**:



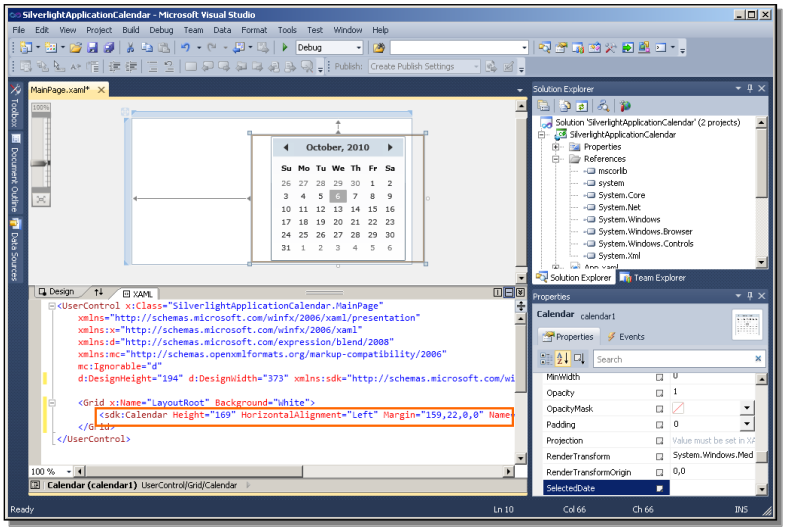
Another series of presentations walks you through building a simple Silverlight Business Application, which of course is likely to be more relevant to your needs, but this simple example serves my purposes in introducing the components of the Silverlight development environment. In this little demo all I do is put a calendar control into the user interface. Visual Studio makes it easy to create a test environment for your application, by creating a separate web project that's parallel to your Silverlight project. That's what this next page of the wizard is doing:



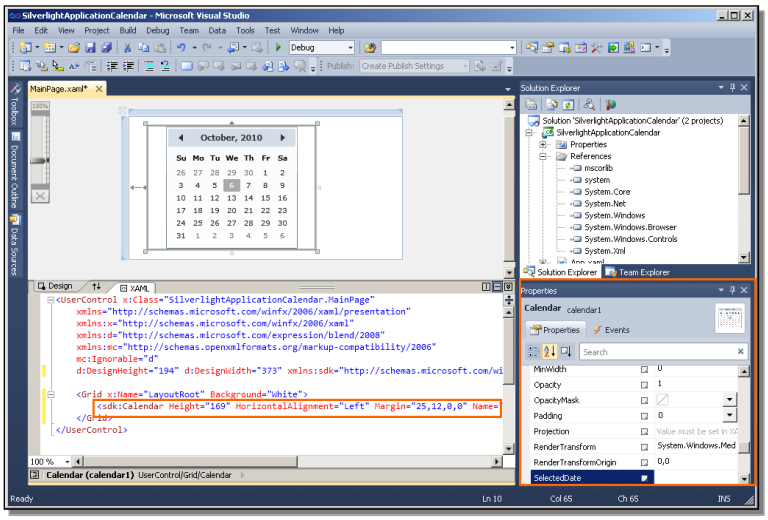
Visual Studio looks a lot like OpenEdge Architect with Visual Designer opened. There's a design canvas at the top, and user interface definition code at the bottom. The code that defines the user interface is a Microsoft-specific extended XML that they call XAML, pronounced "zammel", for Extended Application Markup Language. Generally you can expect that almost all of your user interface definition will be generated for you as you lay out your controls in Visual Studio, or in another tool you'll see in a moment, but this is one of the parts of the development platform that you need to become familiar with. Just as in Architect's Visual Designer, there's a Toolbox with all the controls that Silverlight supports. You can see some of them here, including a Calendar control:



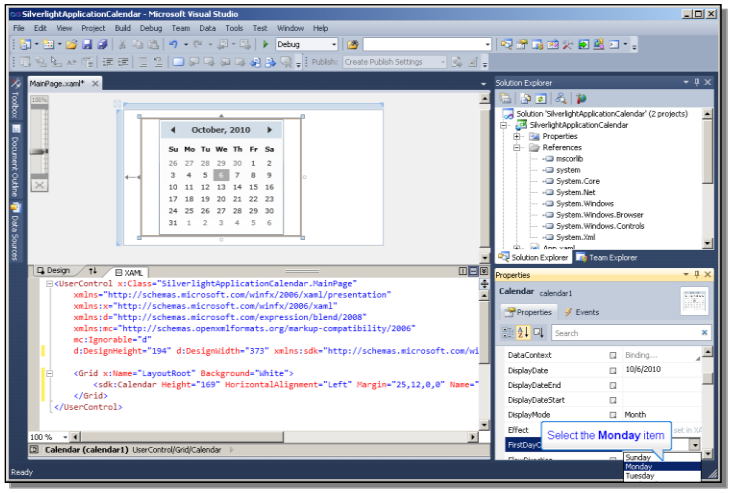
If you drag it onto the design canvas, you can see the control in the WYSIWYG design canvas, and at the same time the XAML statement to define the Calendar is added to the code in the display pane below.



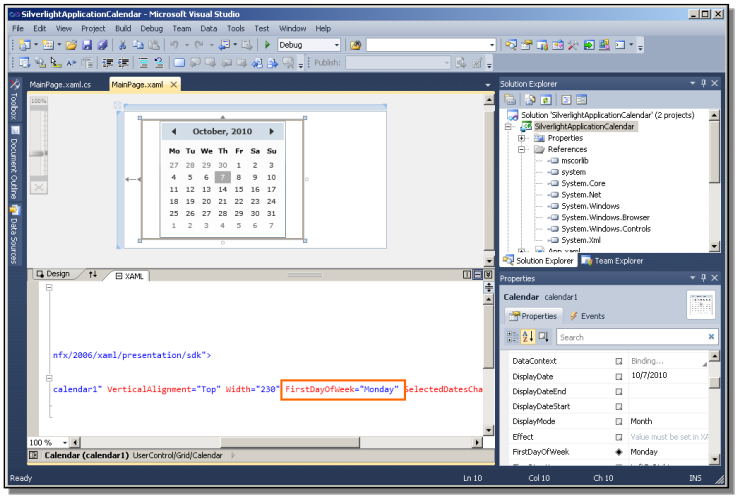
If you move the calendar in the design window, you can see that the value of the margin property in the code that defines where it is within its container is adjusted to reflect the change.



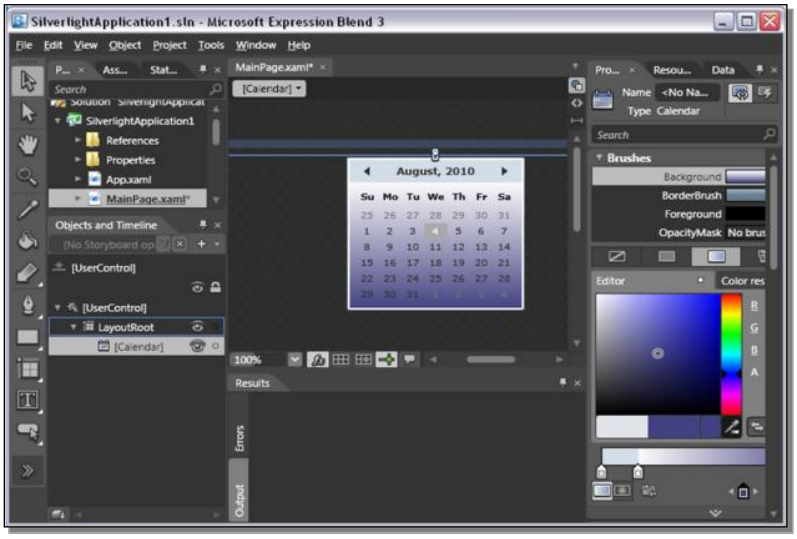
There's also a properties list, as you would expect. You can make a change here, for instance to change the day of the week that's displayed first on each line of the calendar:



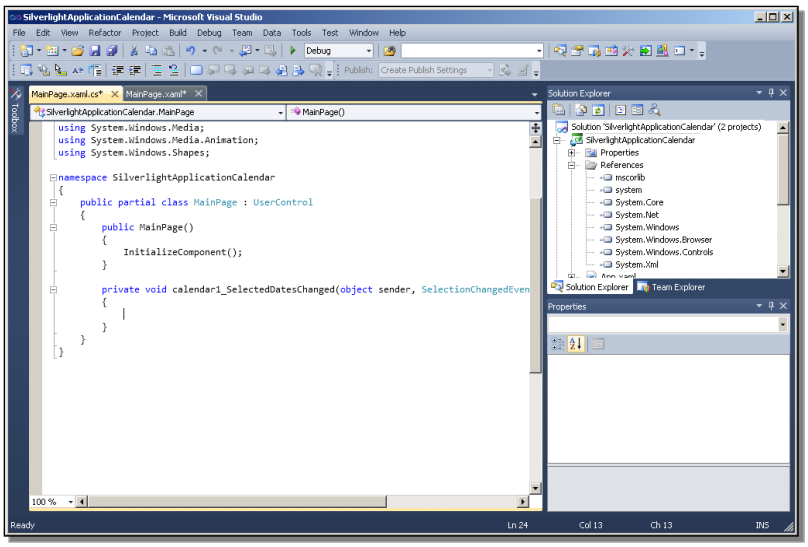
That's immediately reflected on the design canvas. Looking at the properties for the calendar in the XAML, the property setting has been added there as well:



This illustrates that you can expect that most of the XAML your Silverlight application uses will be generated for you. It's worth mentioning that because of the emphasis with Silverlight on a sophisticated user interface and graphical design, Microsoft supports another tool called Expression Blend just for that purpose. It's intended for graphical designers who are building the user interface independently of the software developers who are responsible for coding user interface control logic and business logic. That's part of what we can call a separation of concerns between UI and logic that is an important part of the Silverlight platform. This is a screenshot from Expression Blend:



So beyond the declarative XAML user interface definition, what does the user interface *logic* look like? If you double-click on the calendar control, you can see the skeleton for an event handler in C#:

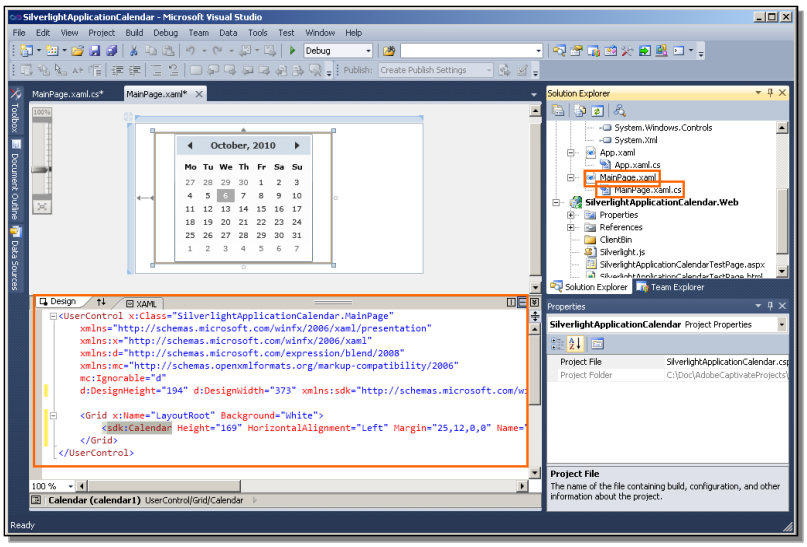


This illustrates another important consideration. If you choose to work with Silverlight, then you'll be doing all the coding beyond the UI definition in a Microsoft language, C# or perhaps VB.NET. So the expertise of your development staff or the people you would expect to hire to do the client-side development is a factor there too.

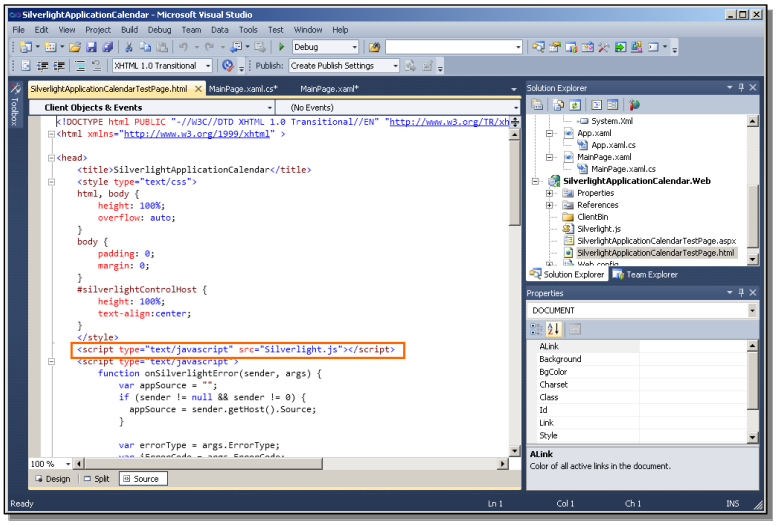
Just to review quickly some of the various code elements that get generated for you when you create a user interface like this:

The file **app.xaml** contains resource definitions that apply to the entire application, across its pages. And there's an accompanying C# file that has generated supporting code for the application.

MainPage.xaml is the UI definition code for the one page containing the calendar control, and **Mainpage.xaml.cs** is the C# code file that was generated from double-clicking on the calendar control, where you would define event handlers and any other UI logic for the page:

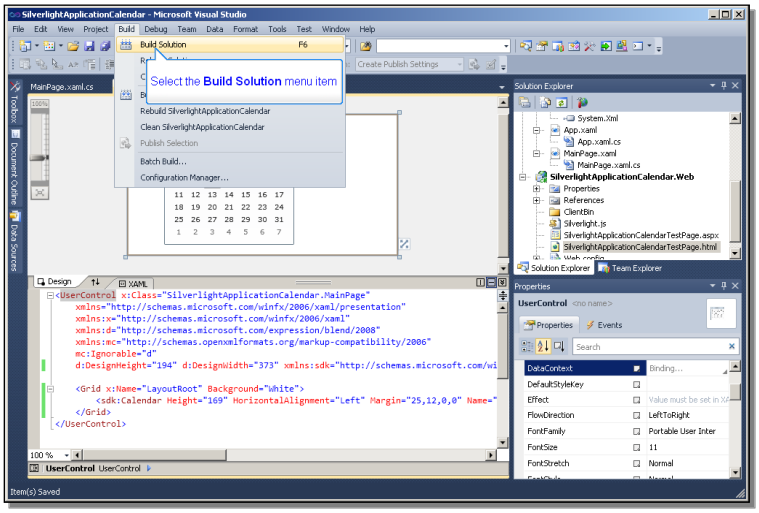


Down below in the Solution Explorer are the files that support the web project where you can test out an application. The principal one is the project **TestPage** html file. You can take a quick look at that just to see one thing. Here is the line of code that invokes another file called **Silverlight.js** :

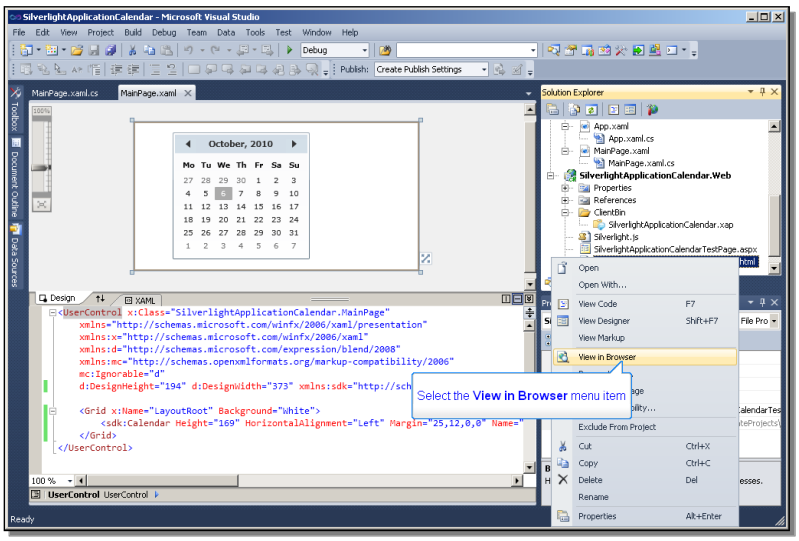


This is the key to the intermediate nature of Silverlight as a UI solution. Silverlight.js is a small JavaScript file that checks to see if the Silverlight plug-in is installed locally, and if not, prompts the user to allow it to be downloaded. This is the step that all Silverlight applications go through to make sure the required plug-in is there. The plug-in is only about four megabytes and just takes a few seconds to download and install, but it's the essental part of the platform that differentiates it from zero-footprint browser-based solutions.

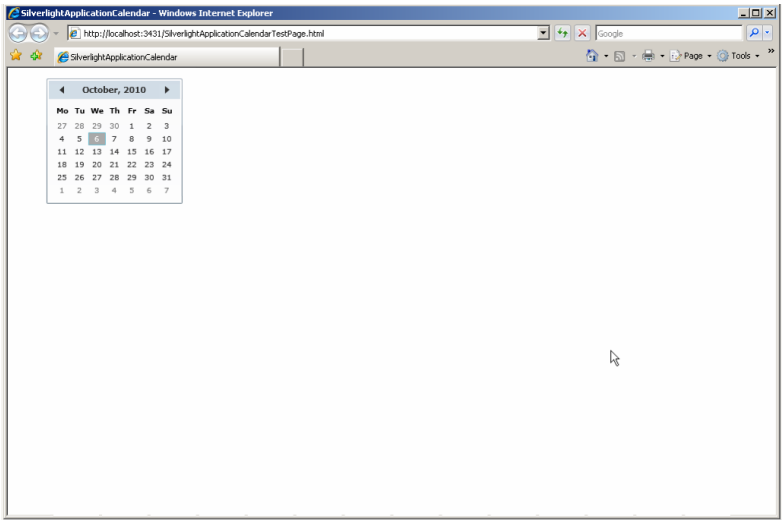
You can save everything that has been generated so far, and then build the solution.



This not only compiles the XAML and C# files, but generates a new file called **MainPage.g.cs** that connects the XAML and C# together to operate as a single unit. Finally, you can see what happens when you run the solution. This time, instead of opening the html code, select **View in Browser** to run it:



And here in the browser is a page with the calendar control.



In summary, Silverlight represents an intermediate type of solution between strictly zero-footprint browser-based libraries that can run on virtually any client platform, and heavyweight desktop solutions that require a client application and supporting controls to be installed on the desktop. It has the support of Microsoft behind it, but also requires that you have a depth of expertise in Microsoft languages and tools to be able to work with it. For that commitment, you get a very powerful user interface platform with a wide array of controls and support for advanced graphical capabilities such as animation and integration of video into your interface.

Other presentations in this series walk you through a simple but more realistic example of a Silverlight Business Application that is likely closer to what your expectations will be for a modern user interface for your ABL application.