DZone Refcardz

CONTENTS INCLUDE:

Silverlight 2

By Chad A. Campbell

ABOUT SILVERLIGHT

Silverlight is Microsoft's offering for designing, developing, and delivering rich interactive applications over the internet. These applications can run on all major platforms and in all major browsers. In this refcard we'll cover the **System.Windows.Browser** namespace in a whirlwind tour that includes: calling managed code from JavaScript, using managed code to control the HTML DOM, and accessing the user's browser window. But first, you will see how to actually create an instance of the Silverlight plug-in.

THE SILVERLIGHT PLUG-IN

Silverlight is a browser-based plug-in that runs within a web page. This plug-in can be integrated with any web technology including ASP, ASP.NET, JSP, and PHP. In order to create an instance of this plug-in, you must use an HTML **object** tag or the JavaScript utility file that is included with the Silverlight SDK. This utility file includes a JavaScript function called **createObjectEx** which creates the **object** element for you. An example of creating a Silverlight plug-in with the **createObjectEx** function is shown here.

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <title>My Silverlight Project</title>
    <script type="text/javascript" src="Silverlight.js"></script>
</head>

<body>
    <!-- Create the Silverlight control inside
    of "mySilverlightHost" -->
    <div id="mySilverlightHost">
        <script type="text/javascript">
            Silverlight.createObjectEx({
            source: "ClientBin/MySilverlightApp.xap",
            parentElement: document.getElementById
            ("mySilverlightHost"),
            id: "mySilverlightControl",
            properties: {
                width: "100%",
                height: "100%",
                version: "2.0"
            },
            events: {}
        });
    </script>
  </div>
</body>
</html>
```

1. Reference the Silverlight.js utility file that is part of the Silverlight SDK.
2. The HTML element that hosts the Silverlight plug-in.
3. The createObjectEx function creates an instance of the Silverlight plug-in.

As this snippet shows, the **createObjectEx** function is basically composed of three sections. The first section defines the mandatory elements of a plug-in instance. The parameters that define these elements are shown in the following table:

Parameter	Description
id	The unique identifier associated with a Silverlight plug-in. This parameter is required.
parentElement	The HTML DOM element that the Silverlight plug-in is hosted within. This parameter is required.
source	The Silverlight application to load into the plug-in. This parameter is required.

In addition to the required elements, you can customize the plug-in instance. These customizations are defined within the nested **properties** object. This object was shown in the previous code snippet. However, only some of the options were shown. All of the possible options are shown in the following table.

Property	Description
background	Represents the color of the rectangular region where the Silverlight plug-in should be.
enableHtmlAccess	Determines whether or not the Silverlight application can access the HTML DOM.
frameRate	The maximum number of frames to render per second.
Height	Defines the height of the rectangle that holds the Silverlight application.
isWindowless	Determines whether the plug-in displays as a windowless plug-in.
splashScreenSource	The location of the XAML file that is used while a Silverlight application is loading.
Version	Specifies the version of Silverlight that the application requires.
Width	Defines the width of the rectangle that holds the Silverlight application.

www.dzone.com

Silverlight 2

The Silverlight Plug-in, continued

In addition to defining the general look of the plug-in, it can also be configured to respond to certain events. These events are listed here:

Event	Description
onError	Handles an exception that has not been caught within the Silverlight application.
onLoad	Triggered when the plug-in is created and all of the Silverlight content has been loaded.
onSourceDownloadProgressChanged	Fires when .05% or more of the Silverlight application has downloaded. This can be used to update the splash screen referenced by the splashScreenSource parameter.
onSourceDownloadComplete	Triggered when the Silverlight application has been fully downloaded.

The events, properties, and core items of the createObjectEx function define how to create an instance of the Silverlight plug-in. This instance may host a Silverlight application which includes some managed code methods. Significantly, you may call these managed code items from JavaScript.

CALLING MANAGED CODE FROM JAVASCRIPT

Silverlight empowers you to expose managed code to the scripting world. When you consider Silverlight's rich networking and data frameworks, the value in this becomes apparent. For instance, you may choose to use the powerful LINQ capabilities to query data or sort a result set. Alternatively, you may need to retrieve data from an RSS feed. Or perhaps you need to communicate with a POX, ATOM, REST, SOAP, or WS-* service. Regardless of your data or communication task, Silverlight allows you to use the power of the .NET Framework to accomplish it. Harnessing this power from JavaScript involves four simple steps:

1. Mark the method, property, or event with the **ScriptableMember** attribute.
2. Tag the class that holds the item from step 1 with the **ScriptableType** attribute.
3. Create the bridge between the class instance and the HTML DOM with the **RegisterScriptableObject** class.
4. Call the item from step 1 from JavaScript.

These four steps outline the process of exposing managed code to the scripting world. To demonstrate how this looks with actual code, we will call a C# method that returns the current date and time. This date and time will then be printed within an alert prompt.

JavaScript from Hosting Web Page

```
function CallManagedCode()
{
    var mySilverlight = document.getElementById("mySilverlight");
    var currentTime = mySilverlight.content.bridge.GetDateTime();
    alert(currentTime);
}
```

1. A Silverlight application is hosting in a Silverlight plugin. This plugin is added to the HTML DOM as an OBJECT element. This line retrieves the Silverlight plugin from the HTML DOM.

2. The content property gives us access to a Silverlight application from the HTML DOM.

3. The "bridge" item is actually set by us. We set this from managed code. This item gives us access to ScriptableMember items in ScriptableType elements.

Class Hosted within Silverlight Application

```
[ScriptableType]
public class MyClass
{
    public MyClass()
    {
        HtmlPage.RegisterScriptableObject("bridge", this);
    }

    [ScriptableMember]
    public string GetDateTime()
    {
        DateTime currentTime = DateTime.Now;
        return currentTime.ToString();
    }
}
```

4. The RegisterScriptableObject method exposes a class instance to the scripting world. This object will be exposed using the handle you pass as the first parameter. In this case, the handle is "bridge".

This example demonstrates how to call managed code from JavaScript. This approach lets us use the .NET Framework in Silverlight to perform tasks that are traditionally difficult or annoying in JavaScript. For instance, consider the task of parsing XML in JavaScript. This task is quite cumbersome with ECMA JavaScript. However, with the powerful .NET Framework provided by Silverlight, this task is a cinch. You can use the feature we just learned about to either enhance, or replace, some current functionality written in JavaScript. However, in order to truly replace that functionality, you will probably need to know how to manage the HTML DOM from managed code.

USING MANAGED CODE TO MANAGE THE HTML DOM

Silverlight gives us the flexibility to manage the HTML DOM from managed code. This means you can alter HTML elements using a language such as C# or Visual Basic. These languages offer features such as compile-time type-checking and richer IntelliSense support through Microsoft Visual Studio. Either way, when you consider coupling this with the ability to call managed code from JavaScript, you can see a powerful combination. This powerful combination is glued together by three classes: **HtmlPage**, **HtmlDocument**, and **HtmlElement**.

→

Using Managed Code to Manage the HTML DOM, continued

The **HtmlPage** class is the core element to use when working with the HTML DOM. This statically visible class exposes a variety of properties that give you access to the key items of a web page. These properties are listed and described in the following table.

Property	Description
BrowserInformation	Gives us access to the user's browser information.
Document	This is an **HtmlDocument** object that represents a web page. This specific property represents the web page hosting the calling Silverlight application.
Plugin	This is an **HtmlElement** that represents the HTML element that is hosting the running Silverlight application.
Window	Gives us access to the user's browser window. This will be demonstrated shortly.

The properties listed in the previous table serve as entry points into the HTML world. The main entry point we will use in this section is provided through the Document property. This property is an **HtmlDocument** object which in turn gives us two properties that serve as entry points into a web page. These properties are listed and described in the following table.

Property	Description
DocumentElement	This property represents the root element of the HTML DOM. Because of this, it always represents the "HTML" element of a web page.
Body	This property gives us immediate access to the contents of the "BODY" element of a web page.

Both of the elements listed in the previous table are actually **HtmlElement** objects. An **HtmlElement** is an in-memory representation of an HTML element. This means that if you have an **input** tag or **span** element in HTML, they will both be represented as **HtmlElement** objects in managed code. This gives us the ability to work with common, useful properties from managed code. These properties are listed in the following table.

Property	Description
Children	A collection of **HtmlElement** items that are hosted by the current **HtmlElement**.
CssClass	The name of the CSS (Cascading Style Sheet) class in use by the **HtmlElement**.
Id	The unique identifier of the **HtmlElement**.
Parent	The **HtmlElement** that hosts the calling item. If the calling item is the **DocumentElement**, this value will be **null**.
TagName	The name of the tag used by the **HtmlElement**.

The properties in the previous table define an **HtmlElement**. An **HtmlElement** is a critical component of an **HtmlDocument** object. This object is accessible through the statically visible **HtmlPage** class. With this class in hand we can perform a number of valuable tasks. Not the least of which include: finding HTML elements, setting element properties, setting CSS information, and managing state information.

Finding HTML Elements

Searching for HTML elements from managed code is an important part of taking advantage of the HTML DOM API. We can traverse through the hierarchy of a web page using the **DocumentElement**, **Children**, and **Parent** properties mentioned earlier. However, the **HtmlDocument** class exposes two methods that empower us to retrieve **HtmlElement** items more efficiently. These methods are listed and described in the following table.

Property	Description
GetElementById	Empowers us to find any element within an **HtmlDocument** by referencing its unique identifier. If the element is found, an object-oriented version of the element, known as an **HtmlElement**, is returned. If the element is not found, **null** will be returned.
GetElementsByTagName	Finds all of the elements with a specified tag name. The results are returned as a collection of **HtmlElement** items.

The previous table shows the two methods that can be used to find items within a web page. To demonstrate how to find an individual item, please look at the following code sample:

```
HtmlDocument document = HtmlPage.Document;

HtmlElement myHtmlElement =
document.GetElementById("myHtmlElement");
```

This element was assumed to be named "myHtmlElement". This name needs to be the id of an element within the hosting web page. Once we have retrieved the element though, we can actually set any of the properties that define it.

Setting Element Properties

The **HtmlElement** class exposes an all-purpose method called **SetProperty**. This method empowers us to set an element's attribute from managed code. To accomplish this, we must use two **string** parameters:

- The first parameter is a string that represents the name of the property to set.
- The second parameter is a string that represents the value to set to the property.

This is demonstrated in the following code sample:

```
HtmlDocument document = HtmlPage.Document;

HtmlElement myHtmlElement =
document.GetElementById("myHtmlElement");

myHtmlElement.SetProperty("innerText", "Hello, HTML");
```

This code sample sets the **innerText** property of an imaginary HTML element to "Hello, HTML". Essentially, imagine setting the text of a **span** named "myHtmlElement" to "Hello, World". This is what the code sample accomplishes. Significantly, this approach is applicable to any element in a web page.

Hot Tip

Be aware of what values you pass to the **SetProperty** method. For instance, it doesn't make sense to set the "checked" property on a **select** tag (the **select** element doesn't have a "checked" property, but a check box does). However, with the flexibility of CSS, we do not have to be as careful when setting CSS information.

Setting CSS Information

In addition to setting properties, we can also set an HTML element's style information from managed code. This can be accomplished by using the **SetStyleAttribute** method. This method takes two **string** parameters:

- The first parameter specifies the CSS property we want to set.
- The second parameter signals the value we want to set the property to.

This approach is demonstrated on a fictional **span** in the following sample:

```
HtmlDocument document = HtmlPage.Document;
HtmlElement myHtmlElement =
document.GetElementById("myHtmlElement");
myHtmlElement.SetStyleAttribute("color", "green");
```

Hot Tip

When setting style attributes from managed code, you must use the scripting naming approach instead of the CSS naming approach to reference a style attribute. For instance, if you wanted to set the background color of an HTML element, you would need to use 'backgroundColor' instead of 'background-color'.

This basic code sample changes the text color of an assumed **span** to green. This demonstrates how to set the CSS information of an **HtmlElement**. This object also allows us to set the properties that separate it from other elements in a web page. This web page is represented as an **HtmlDocument**. This web page may have information that is specific to the current request. For instance, the web page may display information specific to an individual customer or order. These types of situations are often times exposed through state variables.

Managing State Variables

State variables allow you to store information across page post backs. While Silverlight is a client-side platform that helps to alleviate post backs, you may still run into them. The reason why is simply because a Silverlight application is hosted within a web page. Either way, you can read state-related information through the two properties shown in the following table.

Property	Description
Cookies	A string that gives you the ability to manage the cookies associated with a page.
QueryString	A dictionary of the key/value pairs associated with a page's query string.

The two properties shown in the previous table are part of the **HtmlDocument** class. These two properties can be very useful in integration scenarios. In addition, these types of scenarios may have pre-existing JavaScript that can be used. Fortunately, Silverlight gives you the flexibility to call JavaScript from managed code.

CALLING JAVASCRIPT FROM MANAGED CODE

Calling JavaScript from managed code is especially useful if you are attempting to integrate Silverlight into a pre-existing web application. For instance, let's pretend you have a JavaScript function called "myJSFunction". You could call this JavaScript function from managed code using the **Invoke** method as shown here:

```
HtmlWindow window = HtmlPage.Window;
window.Invoke("myJSFunction", new object[]{});
```

The **Invoke** method takes two parameters:

- The first parameter is the name of the JavaScript function to execute. Notice that this value does not include the opening and closing parenthesis associated with a function call.
- The second parameter is an **object** array that stores the parameters to pass to the function. Thus, if "myJSFunction" expected the date and time as parameters, you could pass them using the following:

This approach is demonstrated on a fictional **span** in the following sample:

```
HtmlWindow window = HtmlPage.Window;
window.Invoke("myJSFunction",
  new object[] {DateTime.Now.Date,
  DateTime.Now.TimeOfDay});
```

After the JavaScript function has been invoked, the result will be returned back to the managed code. The result will be available as an object. This gives you the flexibility to interact with any existing JavaScript function.

Hot Tip

Silverlight does not have a print feature built-in. However, if you need to print the contents of a screen, you can use the browser window's built-in print function. This can be accomplished from managed code by using the following:

```
HtmlWindow window = HtmlPage.Window;
window.Invoke("print", new object[]{});
```

ACCESSING THE USER'S BROWSER WINDOW

Silverlight gives us the flexibility to work with a user's browser window through the **HtmlWindow** class. An instance of this class can be obtained from the statically visible **Window** property of an **HtmlPage**. Once retrieved, we can use this object to navigate the user to another location. Alternatively, we can actually use the browser window to display some valuable prompts to a user.

Navigating the Browser

Navigating the browser window is an important part of the hyperlinked world of the internet. Because of this, the **HtmlWindow** class exposes two methods to address navigation:

Accessing the User's Browser Window, continued

- One method is designed to be used with a location within the current page.
- The other method is intended to be used to go to other locations on the internet.

Either way, both of these methods are described in the following table:

Method	Description
Navigate(...)	This method will redirect the browser window to the provided URI. This URI can be loaded in an optional target window. The specifications of this target window can be set via an optional third parameter. The name and specification of the target window parameters match those used by the HTML DOM window.open function.
NavigateToBookmark(...)	This method is used to navigate to a location within the current HTML page. This location must be defined by an anchor tag with the web page.

The two methods listed in the previous table ensure that your Silverlight application isn't isolated. In order to escape from an island of isolation, you can use the following code to go to another page on the internet.

```
Uri uri =
    new Uri("http://www.manning.com/affiliate/
    idevaffiliate.php?id=513_100");
HtmlWindow window = HtmlPage.Window;
window.Navigate(uri, "_blank");
```

This code sample will open a web page in a new browser window. In addition to performing basic navigation though, we can also use the browser to display prompts.

Prompting the User

The **HtmlWindow** class gives us the ability to provide prompts through the user's browser window. These prompts are listed and described in the following table.

Method	Description
Alert(...)	Shows a single message in an HTML alert window.
Confirm(...)	Prompts the user to agree or disagree with a statement or question. This prompt displays two buttons: OK and Cancel. The text of these buttons cannot be customized. If a user selects OK, this method will return **true**. However, if a user selects Cancel, this method will return **false**.
Prompt(...)	Creates a dialog window that displays a single message. In addition, this dialog displays a single text box that the user can enter information into. If the user selects the OK button from this dialog window, the value of that text box will be returned as a **string**. Otherwise, if a user selects Cancel or exits the window, **null** will be returned.

The prompt options shown may be familiar to you. They mimic the prompt options found in the HTML DOM Window object. Either way, in order to deliver a prompt from C#, you can use code like the following:

```
HtmlWindow window = HtmlPage.Window;
window.Alert(DateTime.Now.ToString());
```

This code snippet simply displays the current date and time to the user. That's all there is to it. These kinds of prompts are a valuable part of the browser environment. Equally valuable and useful though is the information associated with the browser itself.

Collecting Browser Information

As a web developer, you may have witnessed how HTML content sometimes renders differently on different browsers and platforms. However, Silverlight content is designed to render consistently across different browsers and platforms. But, if you are editing the HTML DOM through Silverlight, you will still need to take these browser differences into consideration. Fortunately, the **BrowserInformation** class gives us access to this information through the following properties.

Property	Description
BrowserVersion	Stores the major, minor, build, and revision information of a browser. This information is available as a **System.Version** class instance.
CookiesEnabled	A **bool** value that retrieves whether or not the user has enabled cookies in their browser.
Name	The name of the browser that the user is using (i.e. "Microsoft Internet Explorer" or "Netscape").
Platform	A **string** that identifies the operating system the user is using (i.e. "Win32").
UserAgent	The value of the user-agent header that will be sent from the browser to the server.

With these properties, you can easily account for differences between browsers. For instance, absolutely positioning an element in an HTML page is a common pain point across browsers. To do this from Silverlight, you can do something like the following:

```
// Retrieve an html element
HtmlDocument document = HtmlPage.Document;
HtmlElement element = document.GetElementById
("myHtmlElement");

// Absolutely position it based on the browser
BrowserInformation bi = HtmlPage.BrowserInformation;
if (bi.Name == "Internet Explorer")
    element.SetStyleAttribute("top", "300px");
else
    element.SetStyleAttribute("top", "305px");
```

As you can imagine, the **BrowserInformation** class is an important part of working with the HTML DOM. Perhaps an even more powerful part of the HTML DOM API though is the **HtmlWindow.** This API also allows you to manage the HTML DOM from managed code. On the contrary, you can expose your managed code to the HTML DOM. No matter the direction you want to go, Silverlight can help you get there. You can find out more about the incredibly powerful Silverlight platform in the book *Silverlight 2 in Action*.

Hot Tip Be careful when using the Navigate method. If you redirect your user away from the current web page in the current browser window, your Silverlight application will be unloaded. This may not be what you have intended. Because of this, you may want to consider using the "blank" value as the target window.

RESOURCES

Resource	URL
Experience Silverlight	http://silverlight.net/Showcase/
Get Started with Silverlight	http://silverlight.net/GetStarted/
Go Deeper with Silverlight: Silverlight 2 in Action by Chad Campbell and John Stockton	http://books.dzone.com/books/silverlight

Resource	URL
Stay Connected with Silverlight: Popular Silverlight RSS Feeds	http://silverlight.net/blogs/community/rss.aspx http://feeds.feedburner.com/JesseLiberty-SilverlightGeek http://feeds.timheuer.com/timheuer

ABOUT THE AUTHOR

Chad A. Campbell

Chad Campbell is a Microsoft MVP and solutions architect. He has been developing enterprise-level web applications with a wide variety of technologies since 1999. Beginning with the initial public release of what would become Silverlight in 2006, Chad hit the ground running and has not looked back. He holds MCSD and MCTS certifications. In addition, Chad has a BS degree from Purdue University where he focused his studies on computer science and minored in psychology.

Publication

Silverlight 2 in Action (Manning)

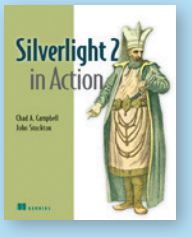
Blog

http://cornucopia30.blogspot.com/

Twitter Feed

http://twitter.com/chadcampbell

RECOMMENDED BOOK



Silverlight 2 in Action is the first book to cover Silverlight 2, a far more robust implementation of Silverlight than the current 1 release that supports only JavaScript. The much-anticipated 2 release adds powerful new features along with the ability to code in multiple languages and integrate your work with Visual Studio and the new Expression suite of tools.

BUY NOW

books.dzone.com/books/silverlight

Get More FREE Refcardz. Visit refcardz.com now!

Upcoming Refcardz:

Core Seam
Core CSS: Part III
Hibernate Search
Equinox
EMF
XML
JSP Expression Language
ALM Best Practices
HTML and XHTML

Available:

Esential Ruby
Essential MySQL
JUnit and EasyMock
Getting Started with MyEclipse
Spring Annotations
Core Java
Core CSS: Part II
PHP
Getting Started with JPA
JavaServer Faces
Core CSS: Part I
Struts2
Core .NET
Very First Steps in Flex
C#
Groovy
NetBeans IDE 6.1 Java Editor
RSS and Atom
GlassFish Application Server
Silverlight 2

Visit refcardz.com for a complete listing of available Refcardz.

Design Patterns
Published June 2008

DZone communities deliver over 4 million pages each month to more than 1.7 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheatsheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
1251 NW Maynard
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-934238-11-0
ISBN-10: 1-934238-11-2
50795
9 781934 238110
$7.95

Version 1.0