Retrieve System Information in Silverlight

In a Silverlight application we are building for a client, they wanted an About screen that would display system information such as the current URL, the operating system name and version, the product name and various other information. In the same application, we built a logging system to gather this same information and write that information to a file to help developers troubleshoot issues. We decided to create a Silverlight class that would gather the information shown in Figure 1.

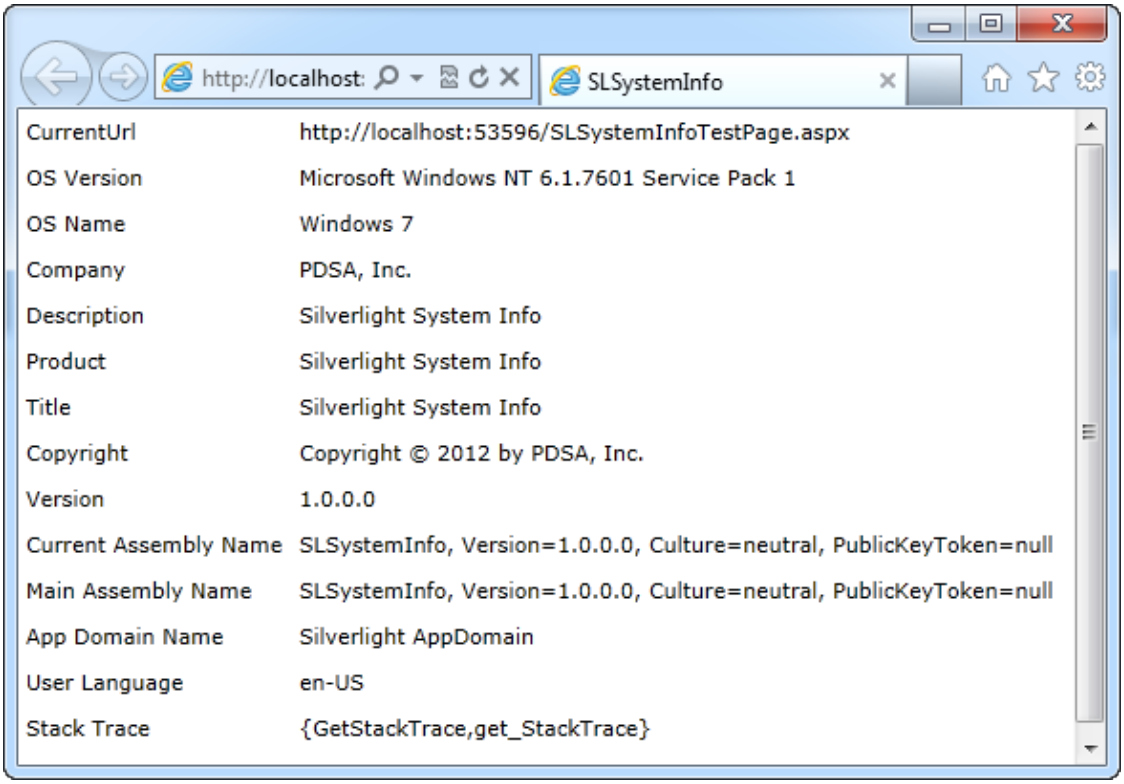

Figure 1: A Silverlight system information class to gather information about the user's environment.

The class where all of this data comes from is named PDSASystemInfo. This class contains a set of properties that get information from the current executing assembly, the Environment class, and a few other classes that give

you system information in your Silverlight application. Let's look at each of these properties in turn.

The Constructor

First off, the constructor for this class retrieves the main assembly for your Silverlight application. This is the first assembly that runs. I assume that the first assembly is where your Silverlight user controls run from, so we need a reference to that assembly in order to retrieve copyright, title, company, and description information. You use the Application.Current.GetType() method to get the Assembly object. From this object you will be able to retrieve the Assembly information you place into your Silverlight application from Visual Studio. A reference to this assembly is placed into a private variable called **_CurrentAssm**.

```
public class PDSASystemInfo
{
  private Assembly _CurrentAssm = null;

  public PDSASystemInfo()
  {
    _CurrentAssm = Application.Current.GetType().Assembly;
  }

   …
   …
}
```

Assembly Information Properties

In Visual Studio if you go into the Project Properties of your solution you can click on the Assembly Information button and enter information about your assembly as shown in Figure 2.

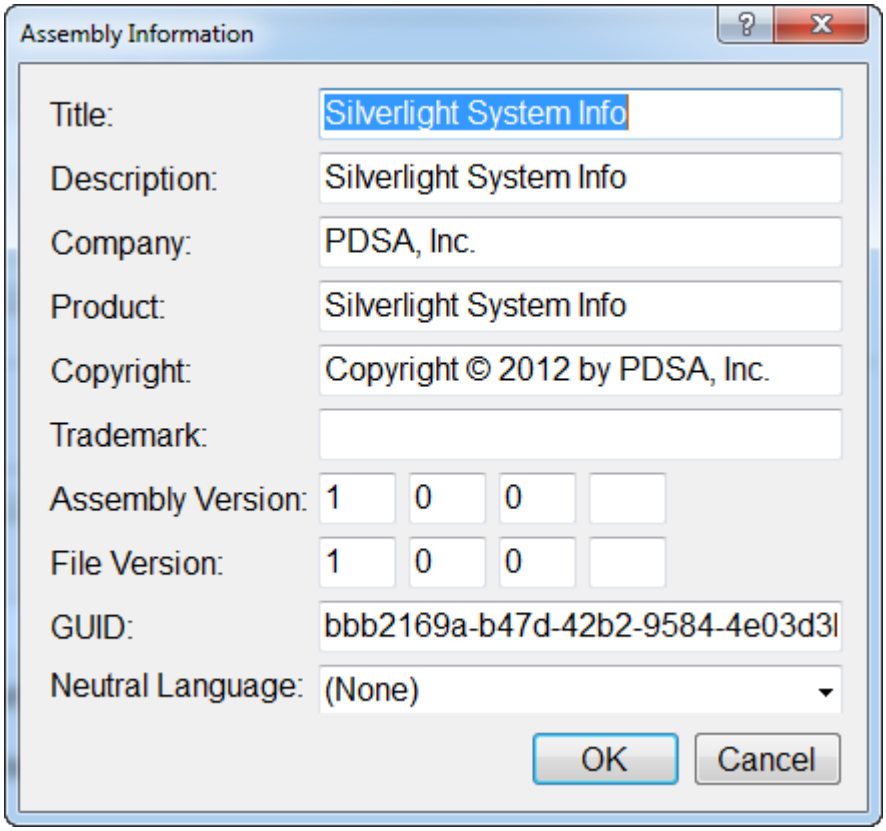

Figure 2: The Visual Studio Assembly Information screen is where you enter information about your application.

To retrieve this information at runtime, you use the Application's main assembly object to call the GetCustomAttributes() method. You pass to this method a type that represents the piece of information you wish to retrieve such as the Company, Product, etc.

```
public string Company
{
  get
  {
    Type at = typeof(AssemblyCompanyAttribute);
    object[] c = _CurrentAssm.GetCustomAttributes(at, false);
    AssemblyCompanyAttribute att =
      ((AssemblyCompanyAttribute)(c[0]));
    return att.Company;
  }
}

public string Description
{
  get
  {
    Type at = typeof(AssemblyDescriptionAttribute);
    object[] c = _CurrentAssm.GetCustomAttributes(at, false);
    AssemblyDescriptionAttribute att =
       ((AssemblyDescriptionAttribute)(c[0]));
    return att.Description;
  }
}

public string Product
{
  get
  {
    Type at = typeof(AssemblyProductAttribute);
    object[] c = _CurrentAssm.GetCustomAttributes(at, false);
    AssemblyProductAttribute att =
       ((AssemblyProductAttribute)(c[0]));
    return att.Product;
  }
}

public string Title
{
  get
  {
    Type at = typeof(AssemblyTitleAttribute);
    object[] c = _CurrentAssm.GetCustomAttributes(at, false);
    AssemblyTitleAttribute att =
       ((AssemblyTitleAttribute)(c[0]));
    return att.Title;
  }
}

public string Copyright
{
  get
  {
    Type at = typeof(AssemblyCopyrightAttribute);
    object[] c = _CurrentAssm.GetCustomAttributes(at, false);
    AssemblyCopyrightAttribute att =
      ((AssemblyCopyrightAttribute)(c[0]));
```

```
    return att.Copyright;
  }
}

public string Version
{
  get
  {
    AssemblyName an = new AssemblyName(_CurrentAssm.FullName);
    return an.Version.ToString();
  }
}
```

Operating System Name and Version

To retrieve the operating system name and version, it is the same as in any .NET application. You will use the Environment class and the associated properties on that class.

```
public string OSVersion
{
  get
  {
    return Environment.OSVersion.ToString();
  }
}

public string OSName
{
  get
  {
    string ret = string.Empty;

    switch (Environment.OSVersion.Version.Major)
    {
      case 7:
        ret = "Windows 8";
        break;
      case 6:
        if (Environment.OSVersion.Version.Minor == 0)
          ret = "Windows Vista";
        else if (Environment.OSVersion.Version.Minor == 1)
          ret = "Windows 7";
        break;
      case 5:
        if (Environment.OSVersion.Version.Minor == 0)
          ret = "Windows 2000";
        else if (Environment.OSVersion.Version.Minor == 1)
          ret = "Windows XP";
        break;
      case 4:
        ret = "Windows NT";
        break;
      default:
        ret = "Unknown Version";
        break;
    }

    return ret;
  }
}
```

The Current URL

A useful property in your application is finding out what the current URL is that is running your Silverlight user control. This is very easy to get at using the System.Windows.Browser.HtmlPage class. You access the Document.DocumentUri property to retrieve the current URL that is running on the user's machine.

```
public string CurrentUrl
{
  get
  {
    try
    {
      return HtmlPage.Document.DocumentUri.ToString();
    }
    catch
    {
      return string.Format(_ERROR_MSG, "Current URL");
    }
  }
}
```

Get the Stack Trace

When you need to debug an application, having access to the stack trace is very helpful. Getting the stack trace in a Silverlight client-side user control is accomplished using the StackFrame class. If you get an exception in your application, the Exception object you get has a StackTrace property that will return your stack trace that got you to your exception. However, if you are not in an exception and just want to get the stack trace information, you write code like that shown in the GetStackTrace() method below.

```
public string GetStackTrace()
{
  StringBuilder sb = new StringBuilder(512);
  int loop = 0;
  string comma = string.Empty;
  string nameToMatch = MainAssemblyName;

  try
  {
    StackFrame sf = new StackFrame(loop);
    if (sf.GetMethod() != null)
      sb.Append("{");

    while (sf.GetMethod() != null)
    {
      // Get methods contained in this assembly only.
      if (sf.GetMethod().DeclaringType.Assembly.
           FullName.Equals(nameToMatch))
      {
        sb.Append(comma + sf.GetMethod().Name);
        comma = ",";
      }

      loop++;
      sf = new System.Diagnostics.StackFrame(loop);
    }
    if (sb.Length > 0)
      sb.Append("}");
  }
  catch
  {
    sb.AppendFormat(_ERROR_MSG, "Stack Trace");
  }

  return sb.ToString();
}
```

If you simply loop through all methods contained in the StackFrame you will get a lot of methods that are part of Silverlight and not your application. In this method you simply check to see if the methods are contained only in the main assembly name. This will probably work fine for a simple Silverlight application, but you may need to expand on this method if you have many client-side DLLs and wish to use this method when you are within any of those other DLLs.

Summary

There are a couple of other properties in the class that you can look at when you download the source code. This class will give you a lot of information that you will find useful when logging or creating an About page for your application. There is also a method named GetAllSystemInfo() that can be used to concatenate all of these properties together into one string. This method is great for logging all of these properties into a file. I will cover a client-side logging utility for Silverlight in my next blog post.

NOTE: You can download the sample code for this article by visiting my website at http://www.pdsa.com/downloads. Select “Tips & Tricks”, then select “Retrieve System Information in Silverlight” from the drop down list.