LOGIN AND LOGOUT WITH THE SILVERPOP API

June 8, 2015

OVERVIEW

1. Background

This article is a continuation of a series of articles on the SilverPop API. Do the setup in the article, http://c-l-associates.com/Home/Getting_Started_With_the_SilverPop_API, if you have not already done so. In this article we will do the first communication with the SilverPop XML API. This article will provide a foundation for all later SilverPop API calls.

SETUP FOR LOGIN AND LOGOUT

1. Changes Needed to the SilverPop Service

1. Fire up Visual Studio and open the SilverPop Tests solution.
2. Add a new class under models named LoginResponse.
3. Edit the class to be as follows:

```
namespace SilverPop_Service.Models
{
  public class LoginResponse
  {
      public bool Success
      {
          get;
          set;
      }
      public string SessionId
      {
          get;
          set;
      }
      public string SessionEncoding
      {
          get;
          set;
      }
      public string OrganizationId
      {
          get;
          set;
      }
      public string Fault
      {
          get;
          set;
      }
  }
}
```

4. Add another class under models named LogoutResponse.

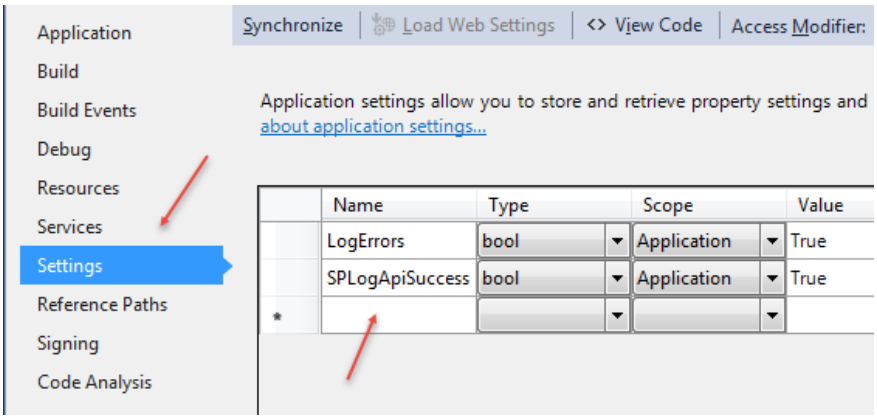
5. Edit the class to be as follows:

```
namespace W4.SilverPop.Service.Models
{
    public class LogoutResponse
    {
        public bool Success { get; set; }
        public string Fault { get; set; }
    }
}
```

6. We are going to log our API calls to SilverPop. If you are doing SilverPop development this is a very handy way to do things. You can send your log to SilverPop support and they can see exactly what the problem is. So, create a new folder under the SilverPop service project called Helpers.
7. Add a new class called LogHelper under the Helpers folder.
8. Add a constructor and a member variable to hold the log directory. Also make the class public and add a reference to the System.IO namespace:

```
using System.IO;

namespace SilverPop_Service.Helpers
{
  public class LogHelper
  {
     private string _logDir;

     public LogHelper( string baseDir )
     {
        if( baseDir.EndsWith( "\\" ) )
        {
           _logDir = baseDir + "Logs";
        }
        else
        {
           _logDir = baseDir + "\\Logs";
        }
        if( !Directory.Exists( _logDir ) )
        {
           Directory.CreateDirectory( _logDir );
        }
     }
  }
}
```

9. Go to the SilverPop Service project settings. Go to the Settings page. Click the link to create a settings file. Add two new settings items:



10. Go back to the LogHelper file and add the new method LogSpApi. What this method does is:
    a. Checks that we want to do logging by using the settings in the settings file
    b. Creates a date time stamped log file name
    c. Logs SilverPop API information to the log file

```
// Used to log one call to the SilverPop API
public void LogSpApi( int pod, string userName, string apiName, string body, string response, bool
success )
{
    //  Check if we are not logging
    if( !Settings.Default.LogErrors )
        return;
    if( !Settings.Default.SPLogApiSuccess && success )
        return;

    //  Create the timestamped file
    var fileName = String.Format( "{0} {1}{2} {3}-{4}-{5} {6}{7}{8}.txt", _logDir + "\\", success ?
    "[SUCCESS] " : "[FAIL] ", apiName, DateTime.Now.Month, DateTime.Now.Day, DateTime.Now.Year,
    DateTime.Now.Hour, DateTime.Now.Minute, DateTime.Now.Second );

    //  Start loggging
    var sw = File.CreateText( fileName );

    try
    {
        sw.WriteLine( "----------------------------------------------------------------------------------
        ----------------------------------" );
        sw.WriteLine( "This file contains the following SP API Information: the POD number, the user,
        success, response, body " );
        sw.WriteLine( "----------------------------------------------------------------------------------
        ----------------------------------" );
        sw.WriteLine( "[POD]" );
        sw.WriteLine( pod );
        sw.WriteLine();
        sw.WriteLine( "[User]" );
        sw.WriteLine( userName );
        sw.WriteLine();
        sw.WriteLine( "[SUCCESS]" );
        sw.WriteLine( success );
        sw.WriteLine();
        sw.WriteLine( "[RESPONSE]" );
        sw.WriteLine( response );
        sw.WriteLine();
        sw.WriteLine( "[BODY]" );
        sw.WriteLine( body );
        sw.Flush();
        sw.Close();
    }
    catch( Exception e )
    {
        sw.WriteLine( e.Message );
    }
    sw.Close();
}
```

11. Go to the ISilverPopApi interface and add one new property and two new methods:

```
public interface ISilverPopApi
{
    LoginResponse LoginResponse
    {
        get;
    }

    void Login( int pod, string userName, string password );
    LogoutResponse Logout( int pod );
```

```
}
```

12. Under the Api folder add a new class called SilverPopApi.cs. Make this class implement the ISilverPopApi interface:

```
using SilverPop_Service.Interfaces;
 using SilverPop_Service.Models;

namespace SilverPop_Service.Api
{

  public class SilverPopApi : ISilverPopApi
  {
     public LoginResponse LoginResponse
     {
        get;
        private set;
     }

     public void Login( int pod, string userName, string password )
     {
        throw new System.NotImplementedException();
     }

     public Models.LogoutResponse Logout( int pod )
     {
        throw new System.NotImplementedException();
     }
  }
}
```

13. Add two new members to the SilverPopApi class. One is an instance of the WebClient class and the other is for our logger. Add the required namespaces:

```
private readonly WebClient _webClient;
private readonly LogHelper _log;
```

14. Create a constructor for the SilverPopApi class and initialize the two new variables. Again add in the new namespace needed:

```
public SilverPopApi()
{
   _webClient = new WebClient
   {
      Headers = new WebHeaderCollection()
   };
   _log = new LogHelper( AppDomain.CurrentDomain.BaseDirectory );
}
```

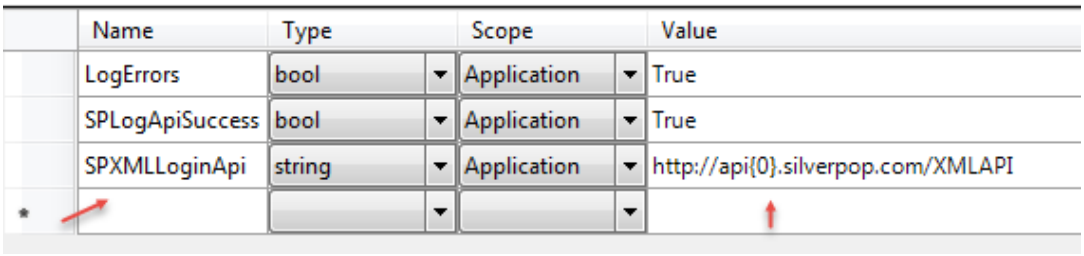
15. The Login and Logout API calls each need a XML template which will be used to construct the body of the API request. Add these two templates to the SilverPopApi class:

```
private const string LoginRequest =
@"<Envelope><Body><Login><USERNAME>{0}</USERNAME><PASSWORD>{1}</PASSWORD></Login></Body></Envelope
>";
private const string LogoutRequest = @"<Envelope><Body><Logout/></Body></Envelope>";
```

16. Compile the application. Now we are ready to code the first SilverPop API call.

2. Coding the Login API Call

1. Go to the project settings and add a new string called SPXMLLoginApi as follows:



Name	Type	Scope	Value
LogErrors	bool	Application	True
SPLogApiSuccess	bool	Application	True
SPXMLLoginApi	string	Application	http://api{0}.silverpop.com/XMLAPI
*			

2. Add a new class under Helpers called ParserHelper. This class will parse the responses brought back from the SilverPop API. Add a method to this class called ParseLogin. This method will parse the SilverPop response using an XmlReader. Refer to the SilverPop API documentation to see the XML that is returned from the Login request:

```
using System;
using System.IO;
using System.Xml;
using SilverPop_Service.Models;

namespace SilverPop_Service.Helpers
{

  public class ParserHelper
  {
     public static LoginResponse ParseLogin( string xml )
     {
        var elementName = "";
        var loginResponse = new LoginResponse();

        // Create an XmlReader
        using( var reader = XmlReader.Create( new StringReader( xml ) ) )
        {
           try
           {
              // Parse the file for each of the nodes
              while( reader.Read() )
              {
                 switch( reader.NodeType )
                 {
                    case XmlNodeType.Element:
                       elementName = reader.Name;
                       break;
                    case XmlNodeType.Text:
                       if( elementName == "SUCCESS" )
                       {
                          loginResponse.Success = reader.Value.ToLower() == "true";
                       }
                       else if( elementName == "SESSIONID" )
                       {
                          loginResponse.SessionId = reader.Value;
                       }
                       else if( elementName == "SESSION_ENCODING" )
                       {
                          loginResponse.SessionEncoding = reader.Value;
                       }
                       else if( elementName == "ORGANIZATION_ID" )
                       {
                          loginResponse.OrganizationId = reader.Value;
                       }
                       break;
                    case XmlNodeType.CDATA:
```

```
                        if( elementName == "FaultString" )
                        {
                            loginResponse.Fault = reader.Value;
                        }
                        break;
                    case XmlNodeType.EndElement:
                        break;
                }
            }
        }
        catch( Exception )
        {
            throw new NotImplementedException();
        }
    }
    return loginResponse;
  }
 }
}
```

3. Now code the Login method in the SilverPopApi class. Notice the Headers assignment. This is absolutely essential when coding against the SilverPop API. Without this line, nothing works. I have to point out that this is not stressed in the SilverPop API. It is buried in one of the Java samples. Also, note we are calling the logger to log our API call. We also, use our template to create the body of the request:

```
public void Login( int pod, string userName, string password )
{
    try
    {
        _webClient.Headers["Content-type"] = "text/xml;charset=utf-8";
        var url = string.Format( Settings.Default.SPXMLLoginApi, pod );
        var body = string.Format( LoginRequest, userName, password );

        var postData = Encoding.ASCII.GetBytes( body );
        var responseBytes = _webClient.UploadData( url, postData );
        var response = Encoding.ASCII.GetString( responseBytes );
        LoginResponse = ParserHelper.ParseLogin( response );
        _log.LogSpApi( pod, Environment.UserName, "Login", body, response,
        LoginResponse.Success );
    }
    catch( Exception e )
    {
        throw new Exception( e.Message );
    }

    if( LoginResponse.Success != true )
    {
        throw new Exception( LoginResponse.Fault );
    }
}
```

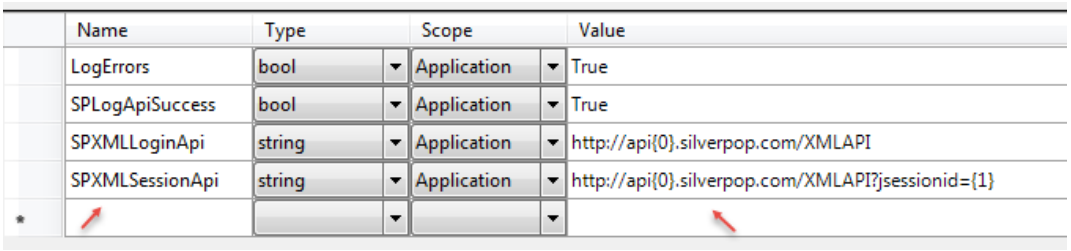
3. Coding the Logout API Call

1. Add another method to ParserHelper for parsing the Logout response from SilverPop Refer to the SilverPop API documentation to see the XML that is returned from the Logout request:

```
public static LogoutResponse ParseLogout( string xml )
{
    var elementName = "";
    var logoutResponse = new LogoutResponse();

    // Create an XmlReader
    using( var reader = XmlReader.Create( new StringReader( xml ) ) )
    {
```

```
            try
            {
                // Parse the file for each of the nodes
                while( reader.Read() )
                {
                    switch( reader.NodeType )
                    {
                        case XmlNodeType.Element:
                            elementName = reader.Name;
                            break;
                        case XmlNodeType.Text:
                            if( elementName == "SUCCESS" )
                            {
                                logoutResponse.Success = reader.Value.ToLower() == "true";
                            }
                            break;
                        case XmlNodeType.CDATA:
                            if( elementName == "FaultString" )
                            {
                                logoutResponse.Fault = reader.Value;
                            }
                            break;
                        case XmlNodeType.EndElement:
                            break;
                    }
                }
            }
            catch( Exception )
            {
                throw new NotImplementedException();
            }
        }
        return logoutResponse;
    }
```

2. Add another setting to the settings values for the Session API string:



Name	Type	Scope	Value
LogErrors	bool	Application	True
SPLogApiSuccess	bool	Application	True
SPXMLLoginApi	string	Application	http://api{0}.silverpop.com/XMLAPI
SPXMLSessionApi	string	Application	http://api{0}.silverpop.com/XMLAPI?jsessionid={1}
*			

3. Add a method to the SilverPopApi class called GetResponse. This will be our generic way to send a request and get a response from SilverPop for all API calls going forward:

```
    public string GetResponse(int pod, string body)
    {
        _webClient.Headers["Content-type"] = "text/xml;charset=utf-8";
        var url = string.Format(Settings.Default.SPXMLSessionApi, pod, LoginResponse.SessionId);
        var postData = Encoding.ASCII.GetBytes(body);
        var responseBytes = _webClient.UploadData(url, postData);
        var response = Encoding.ASCII.GetString(responseBytes);
        return response;
    }
```

4. Add another method to SilverPopApi for logging out of the API:

```
public LogoutResponse Logout( int pod )
```

```
{
    if( LoginResponse == null )
    {
        throw new Exception( "Login to SP API site was not attempted." );
    }

    const string body = LogoutRequest;
    var response = GetResponse( pod, body );

    var logoutResponse = ParserHelper.ParseLogout( response );
    if( logoutResponse.Success != true )
    {
        throw new Exception( logoutResponse.Fault );
    }
    _log.LogSpApi( pod, Environment.UserName, "Logout", body, response, logoutResponse.Success );
    return logoutResponse;
}
```

5. Compile the application. Now it is time to use our unit test project to test our API.

4. Unit Testing the Login and Logout API Calls

1. Go to the unit test project and rename the default UnitTest1.cs file to SilverPopTests.cs.
2. Create a new folder under the unit test project called App_Data.
3. Create two private variables in the test file. One for the default AppData directory (which will need later) and one for the instance of the SilverPopApi class:

```
private string _dir;
private SilverPopApi _silverPopApi;
```

4. Create the TestInitialize fixture that will be called for every SilverPop test. This is a standard MSTest feature:

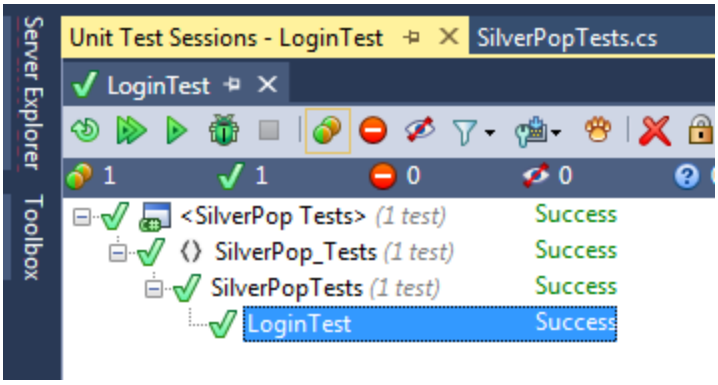
```
[TestInitialize]
public void Setup()
{
    _dir = Directory.GetCurrentDirectory();
    _dir = _dir.Replace("\\bin\\Debug", "");
    _dir += @"\App_Data\";

    _silverPopApi = new SilverPopApi();
}
```

5. Remove the default test that existed in the SilverPopTests.cs file. Now create the LoginTest method Replace the username and password that you setup at SilverPop with the ones below. Your API call will undoubtedly fail without the correct credentials. Also the value 2 is the pod that SilverPop has assigned to you:

```
[TestMethod]
public void LoginTest()
{
    try
    {
        // CHANGE ME: Pod, SP user name and password
        _silverPopApi.Login( 2, "yourname@yourdomain.com", "yourpassword" );
    }
    catch( Exception )
    {
        Assert.IsFalse( true );
    }
    Assert.IsNotNull( _silverPopApi.LoginResponse.SessionId );
    Assert.IsNotNull( _silverPopApi.LoginResponse.OrganizationId );
    Assert.IsNotNull( _silverPopApi.LoginResponse.SessionEncoding );
    Assert.IsNotNull( _silverPopApi.LoginResponse.Success );
    Assert.IsTrue( _silverPopApi.LoginResponse.Success );
```

```
}
```

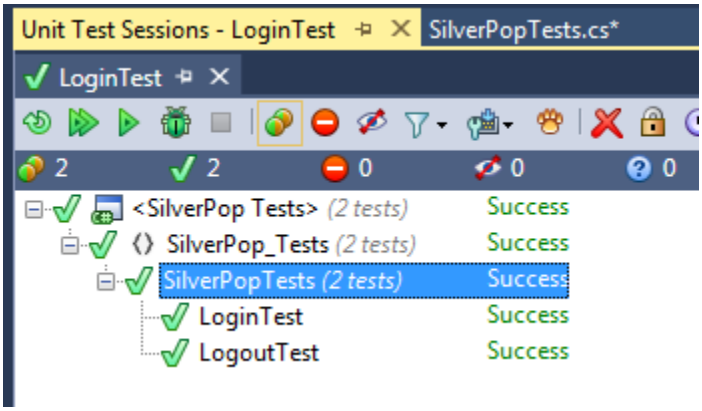
6. Run the unit test and verify the results:



7. Now let's add the Logout test method. Here again you will need to add your credentials and use the correct pod number:

```
[TestMethod]
public void LogoutTest()
{
    try
    {
        // CHANGE ME: Pod, SP user name and password
        _silverPopApi.Login( 2, "yourname@yourdomain.com", "yourpassword" );
        Assert.IsNotNull( _silverPopApi.LoginResponse.SessionId );
        Assert.IsNotNull( _silverPopApi.LoginResponse.OrganizationId );
        Assert.IsNotNull( _silverPopApi.LoginResponse.SessionEncoding );
        Assert.IsNotNull( _silverPopApi.LoginResponse.Success );
        Assert.IsTrue( _silverPopApi.LoginResponse.Success );

        var logoutResponse = _silverPopApi.Logout( 2 );
        Assert.IsNotNull( logoutResponse.Success );
        Assert.IsTrue( logoutResponse.Success );
    }
    catch( Exception )
    {
        Assert.IsFalse( true );
    }
}
```

8. Run both tests. Verify the results:



9. That's it for logging in and logging out. The next article will show how to add a SilverPop database or contact list. The next article in this series can be found at: http://c-l-associates.com/Home/SilverPop.