Galileo API Developer Notes

A Galileo Web Services .NET Connection Class Using C#

25 October 2011

Version 1.2

Contents

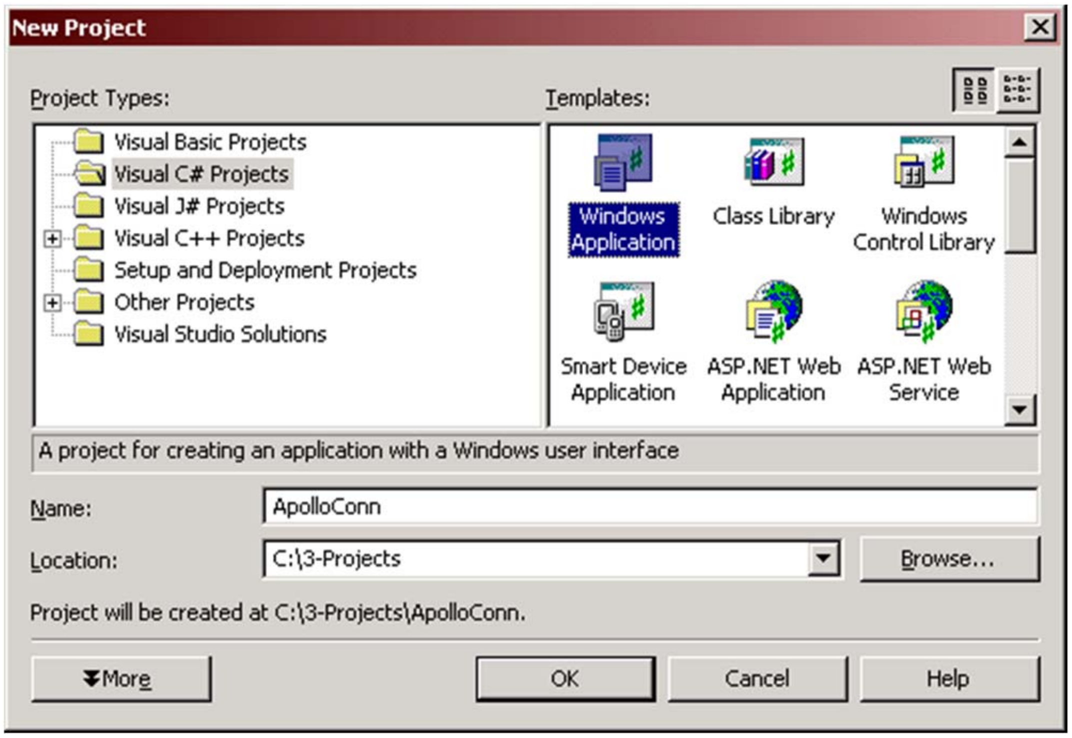
Overview

One of the fastest ways to develop a client application for Galileo Web Services (GWS) is to take advantage of the built-in features of Visual Studio for Microsoft .NET. Much of the work is handled behind the scenes by the Web Reference feature of Visual Studio. This short tutorial demonstrates how to develop an Apollo connection class that your application can use to call the GWS XML Select methods.

Getting Started

This example develops a simple application that includes the connection class and a simple class that uses the connection. The tutorial assumes that you are developing in C# using Visual Studio. You also need the GWS credentials provided by your Galileo account manager.

1. Open a project in Visual Studio. The project can be a Windows Application or a Class Library, if you want to re-use this connection class for many projects.



A project for creating an application with a Windows user interface

Name: ApolloConn

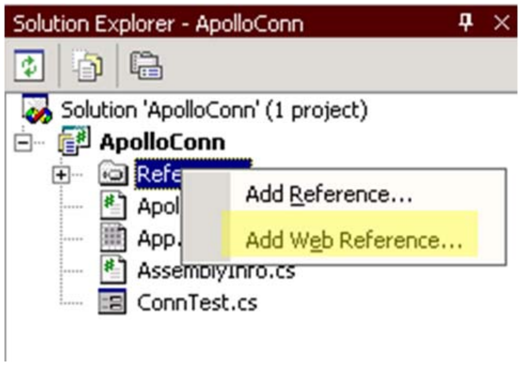
Location: C:\3-Projects

Project will be created at C:\3-Projects\ApolloConn.

Creating the Connection Class

The first step in creating a connection class is to establish a reference to the Galileo Web Services.

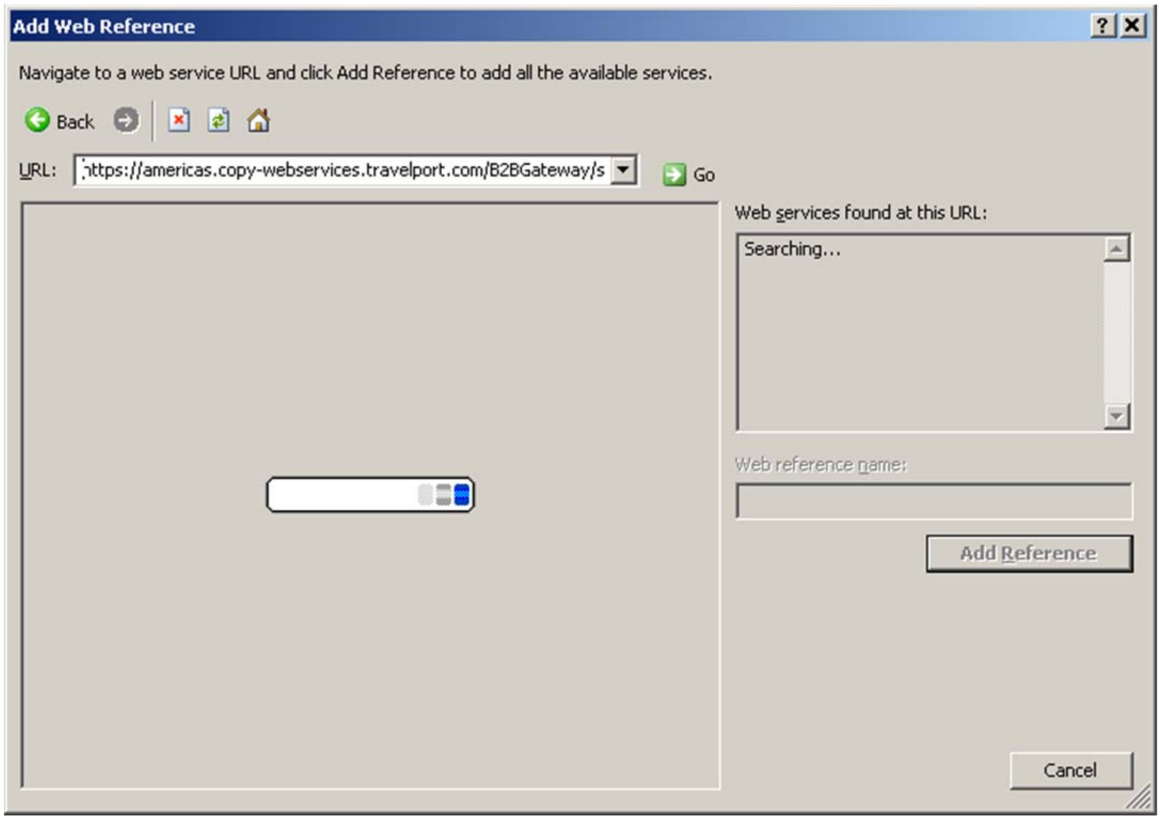
1. In the Solution Explorer, right-click on **Reference** and select **Web Reference**.



2. Enter the URL of the XML Select Web Service based on your region. To review the correct URL for your region, see the table at http://testws.galileo.com/GWSSample/Help/GWSHelp/connecting_to_gws.htm

   - To use as an example, the copy system for US Markets is:

https://americas.copy-webservices.travelport.com/B2BGateway/service/XMLSelect?WSDL



Navigate to a web service URL and click Add Reference to add all the available services.

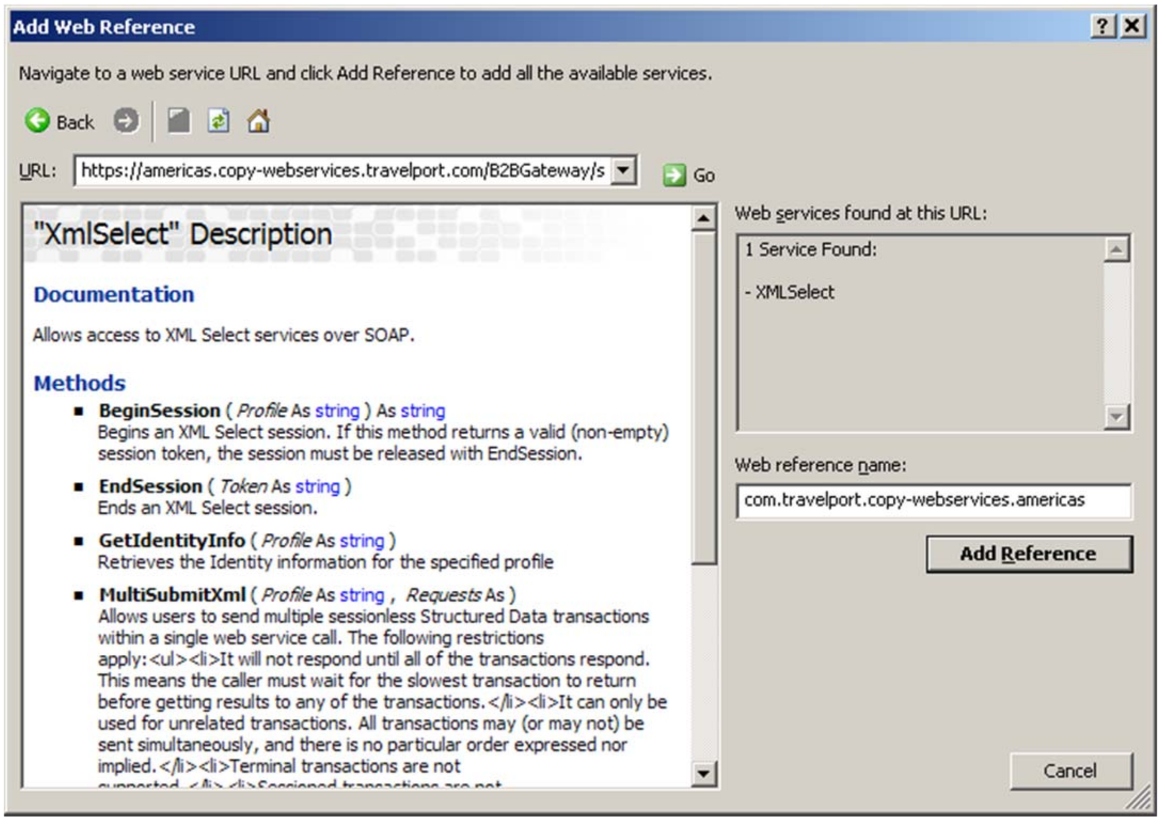
URL: https://americas.copy-webservices.travelport.com/B2BGateway/s

Web services found at this URL:

Searching...

Web reference name:

3. Click the **Add Reference** button.



Navigate to a web service URL and click Add Reference to add all the available services.

URL: https://americas.copy-webservices.travelport.com/B2BGateway/s

"XmlSelect" Description

Documentation

Allows access to XML Select services over SOAP.

Methods

- **BeginSession** (*Profile* As string) As string
  Begins an XML Select session. If this method returns a valid (non-empty) session token, the session must be released with EndSession.
- **EndSession** (*Token* As string)
  Ends an XML Select session.
- **GetIdentityInfo** (*Profile* As string)
  Retrieves the Identity information for the specified profile
- **MultiSubmitXml** (*Profile* As string , *Requests* As)
  Allows users to send multiple sessionless Structured Data transactions within a single web service call. The following restrictions apply:<ul><li>It will not respond until all of the transactions respond. This means the caller must wait for the slowest transaction to return before getting results to any of the transactions.</li><li>It can only be used for unrelated transactions. All transactions may (or may not) be sent simultaneously, and there is no particular order expressed nor implied.</li><li>Terminal transactions are not

Web services found at this URL:

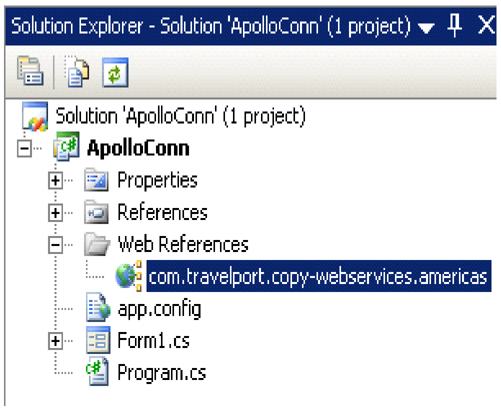
1 Service Found:

- XMLSelect

Web reference name: com.travelport.copy-webservices.americas

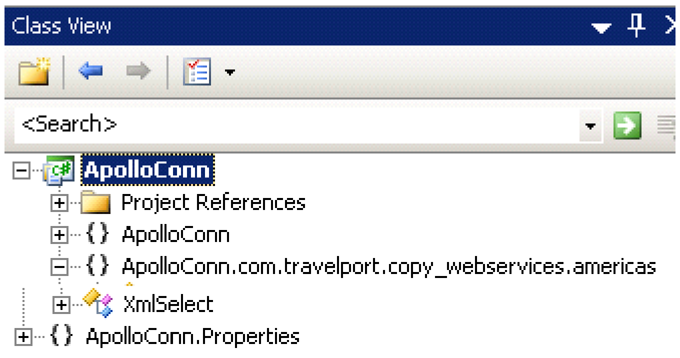
The **com.travelport.copy-webservices.americas** reference is added to your project, and an **XmlSelect** class is created, which encapsulates the SOAP protocols and the method calls to the Web service.

Web Reference



```
Solution 'ApolloConn' (1 project)
ApolloConn
    Properties
    References
    Web References
        com.travelport.copy-webservices.americas
    app.config
    Form1.cs
    Program.cs
```

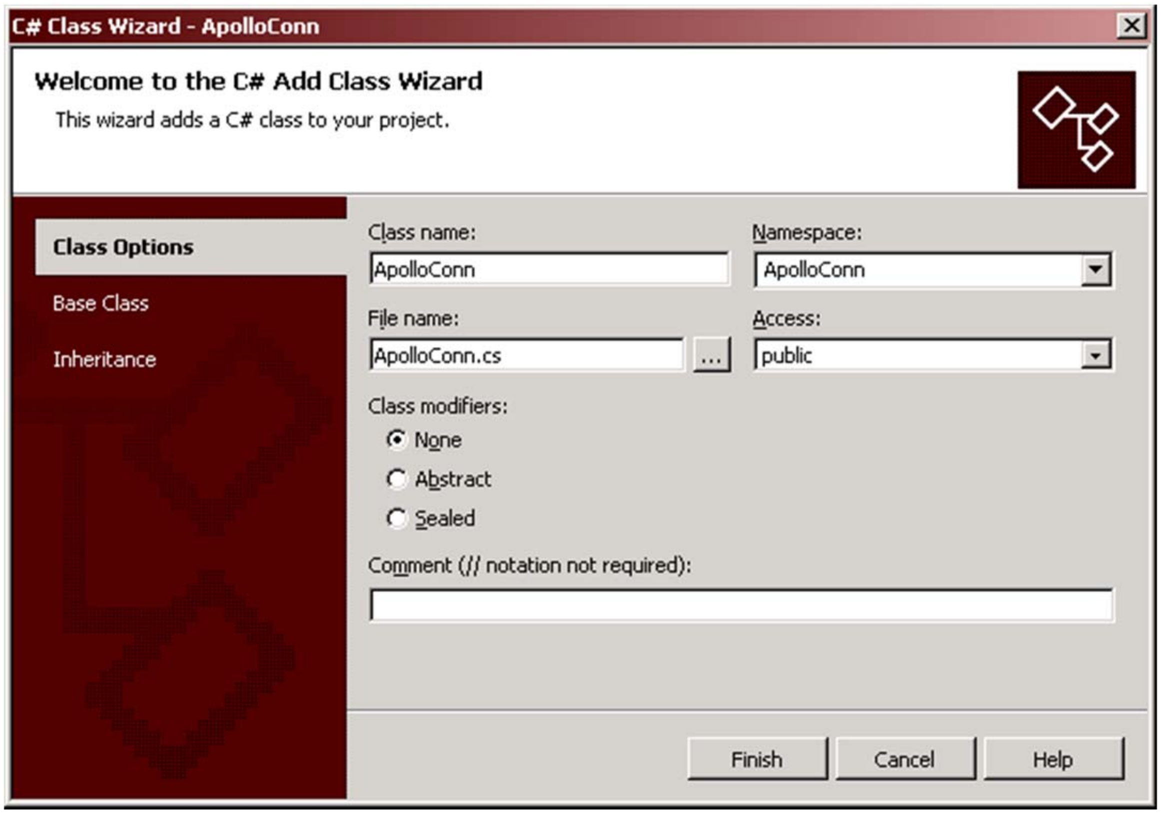
Class



```
ApolloConn
    Project References
    ApolloConn
    ApolloConn.com.travelport.copy_webservices.americas
        XmlSelect
ApolloConn.Properties
```

The XMLSelect class can be used directly by the rest of your application. However, a better approach is to create a new class which inherits from the XMLSelect class. This new class encapsulates all of the details of the connection to the Apollo system, including the credentials needed to access the system, as well as the Web service methods.

4. Add a class called **ApolloConn** to your project. This class inherits from **com.travelport.copy_webservices.americas.XmlSelect**.


C# Class Wizard - ApolloConn
Welcome to the C# Add Class Wizard
This wizard adds a C# class to your project.
Class Options
Base Class
Inheritance
Finish
Cancel
Help

Class name: ApolloConn
Namespace: ApolloConn
File name: ApolloConn.cs
Access: public
Class modifiers: [✓ None] Abstract Sealed
Comment (// notation not required):

5. Add these imports:

```
using System;
using System.Configuration;
using System.Net;
using System.Xml;
```

6. Define the variables that will be used throughout the class:

```
// Define variables used by multiple methods
private string gwsHAP;
private XmlElement defaultFilter;
private string token = "";              // Session token
```

The class, with comments, now looks like:

```
using System;
using System.Configuration;
using System.Net;
using System.Xml;
namespace ApolloConn
{
    /// <summary>
```

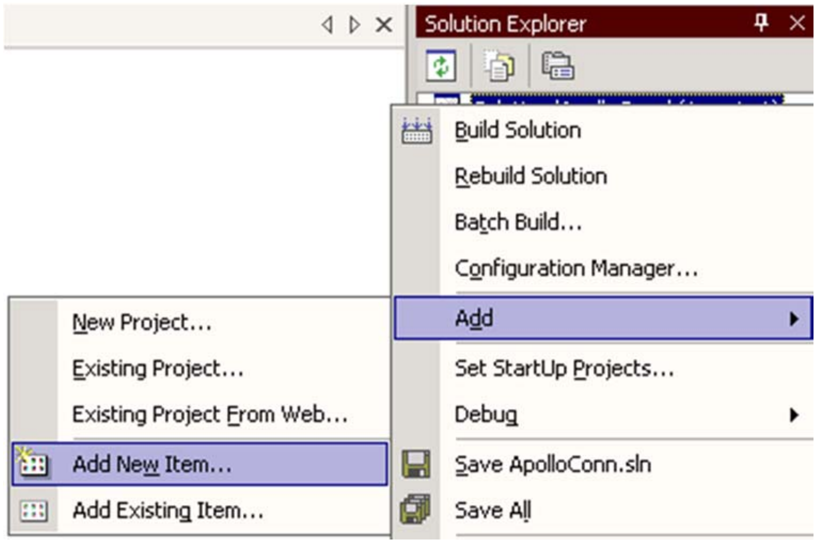
```
        /// This class provides the actual connection to the Apollo GDS system for
        /// executing specific XML transactions. The goal of the class is to encapsulate the
        /// actual connection method and the specific credentials needed for access.
        /// Inherits from the Web Service proxy, so that all of the GWS transaction methods
        /// are available to this instance.
        /// This class also helps manage the session, creating a new session when needed.
        /// </summary>
        public class ApolloConn : com.travelport.copy_webservices.americas.XmlSelect
                {
                // Define variables used by multiple methods
                private string gwsHAP;
                private XmlElement defaultFilter;
                private string token = "";           // Session token

                //------------------------------------------------------------------------------------
                public ApolloConn()
                {
                        //
                        // TODO: Add constructor logic here
                        //
                }
        }
    }
```

Using App.config

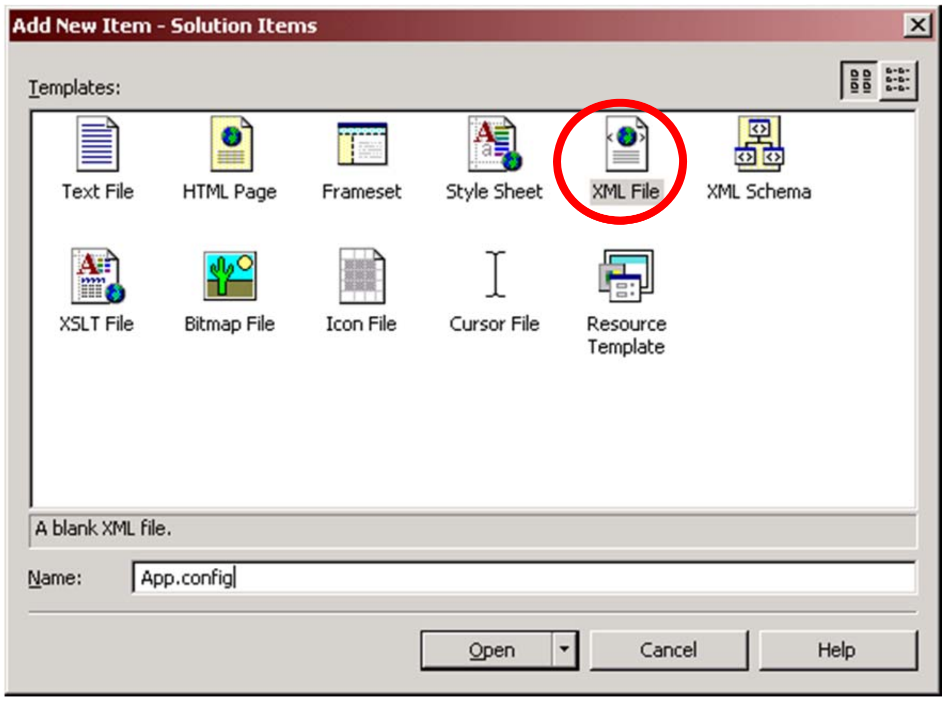
The next step is to add the access to the GWS credentials. Use the .NET *Dynamic Properties* feature, which stores properties that are read at runtime from an external file in your project, **App.config**,. App.config is not a secure location for data such as passwords, so access to this file should be limited.

The configuration file contains the user name, password, Host Access Profile (HAP), and a URL that defines the end point of the GWS Web Service. Even though the URL is included in the WSDL file, it is a good idea to have it in the configuration file, in case there is a change, such as a move from the copy system to production.

1. Right-click on the project in the Solution Explorer and select **Add/Add New** to add the App.config file to your project.



2. Select **XML File**, and name this file **App.config**.



A blank XML file.

Name: App.config

3. Add your credentials to the file (replacing *your . . .* with your values) so that it looks similar to:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
            <add key="GWSUSERNAME" value="yourUserName" />
            <add key="GWSPASSWORD" value="yourPassword" />
            <add key="GWSURL" value=" https://americas.copy-
webservices.travelport.com/B2BGateway/service/XMLSelect " />
            <add key="GWSHAP" value="ApolloCopy_yourHAP" />
    </appSettings>
</configuration>
```

When your project is compiled, App.config is copied to the project output folder and renamed ApolloConn.exe.config. This file is distributed with your application.

4. Retrieve the individual values by reading them from the configuration file, to continue with the constructor:

```
// Create and set up the credentials for XMLSelect WebService.
string userName = ConfigurationSettings.AppSettings["GWSUSERNAME"];
string password = ConfigurationSettings.AppSettings["GWSPASSWORD"];
gwsHAP   = ConfigurationSettings.AppSettings["GWSHAP"];
this.Url = ConfigurationSettings.AppSettings["GWSURL"];
```

5. Store the local credentials by creating a **NetworkCredential** using the retrieved user name and password.

```
NetworkCredential netCredentials = new NetworkCredential(userName, password);
```

6. Add a credential cache to combine your credentials with the GWS end point and set a few parameters. The credential cache is used each time a method is called on the Web Service. The necessary settings for the credential cache are:

```
//Create the Credential Cache to assign to XMLSelectWebService client
CredentialCache cc = new CredentialCache();
//Xml Select uses Basic Authentication, but Windows XP defaults to Digest
cc.Add(new Uri(this.Url), "Basic", netCredentials);
Credentials = cc;
PreAuthenticate = true;
```

7. Create a default filter to be added to the various method calls. The filter is an XML element (not a string) because this parameter type is used by the Web Service methods.

```
// Create a default filter document for use in the overloaded simplified requests
XmlDocument dFilter = new XmlDocument();
dFilter.LoadXml("<_/>");
defaultFilter = dFilter.DocumentElement;
```

The constructor is complete.

SubmitXml

Add the method to submit sessionless XML transactions. Sessionless transactions are used for air, car, and hotel availability, as well as all cruise transactions.

This method accepts the XML-formatted request string and converts it into an XML Element, as required by the service. It then adds the default filter and the Host Access Profile (HAP) and calls the service. The response is converted back to a string and returned.

```
public string SubmitXml(string request)

{
        // Simple overloaded version of SubmitXml transaction with a string request,

        // adding the HAP and using the default filter

        XmlDocument xmlRequest = new XmlDocument();

        xmlRequest.LoadXml(request);

        return this.SubmitXml(   this.gwsHAP,

                                 xmlRequest.DocumentElement,

                                 defaultFilter).OuterXml;

}
```

A similar method that returns an **XmlElement** can be added simply by replacing the string return type with **XmlElement** and removing the **OuterXml** in the return line.

SubmitXMLOnSession

Sessioned transactions require little more work. To simplify things, this method creates a session if one does not already exist. The session identifier is held in the token variable. The rest of the method looks a lot like SubmitXml.

```
public string SubmitXmlOnSession(string request)

{

        // Simple overload for session requests. String input, adds HAP and filter

        // and begins a session if one does not exist.

        XmlDocument xmlRequest = new XmlDocument();

        xmlRequest.LoadXml(request);


        // Create a session if one does not already exist

        if (this.token == "") this.token = this.BeginSession(this.gwsHAP);

        return this.SubmitXmlOnSession(this.token,

                                       xmlRequest.DocumentElement,

                                       defaultFilter).OuterXml;

}
```

Again, a similar method that returns an **XmlElement** can be added simply by replacing the string return type with **XmlElement** and removing the **OuterXml** in the return line.

You must end the session explicitly by calling the EndSession method of this class. Keep in mind that GWS sessions expire quickly if there is no activity.

Using the Connection Class

To use the connection class:

8. Create a request string.

9. Create an instance of the connection class.

10. Submit a transaction.

For example, if your main application has a button called **button1** and a text box called **textBox1**, execute the simple XML Select transaction to get a local time:

```
        private void button1_Click(object sender, System.EventArgs e)

{

        string request = "<LocalDateTimeCT_6_0><LocalDateTimeMods>”

                                    += “<ReqCity>DEN</ReqCity>”

                                    += “</LocalDateTimeMods></LocalDateTimeCT_6_0>";

        ApolloConn host = new ApolloConn();

        textBox1.Text = host.SubmitXml(request);

}
```

Now you can use Galileo Web Services.

Optimizing the GWS Connection

Although .NET handles all of the details needed to create the SOAP envelope, marshal the data, and handle the data transport, not all the Microsoft defaults work with the Galileo Web Services.

GWS supports the GZip compression standard for both requests and responses. GZip is not included in the .NET Web service tools. This section describes overcoming the defaults and adding GZip capability to your connection class.

Optimization Approach

The automatically generated code that the .NET framework created when you added the Web Reference inherits from a class called:

System.Web.Services.Protocols.SoapHttpClientProtocol

This class, with the classes it inherits from, provides the Web Service framework that the connection class you created is based upon. Many of the settings and defaults in these underlying classes are hidden and hard to access.

Therefore, create a new class that inherits from System.Web.Services.Protocols.SoapHttpClientProtocol, adds access to several properties, and overrides methods to ensure that the properties are used correctly. Support for the GZip compression is obtained by adding the class library ICSharpCode.SharpZipLib.dll, which is used by the new class.

What Optimization Provides

The new optimization class provides the access to the following properties:

- **Expect100Continue**: Using Expect 100-Continue can add anywhere from 25 to 150 milliseconds to a typical transaction, and network delays can make the time much longer. Any client that uses GWS services should disable Expect 100-Continue. The framework hides access to this setting. The new class provides access to this setting as a property named Expect100Continue.
- **KeepAlive**: Using the Connection:KeepAlive header on HTTP 1.1 causes the framework to pool HTTP connections. Connection pooling can provide some performance improvement by eliminating the time to establish a connection when connections are available in the pool.

  However, connections in the pool can be closed by a server for a variety of reasons. When connections are closed, a Web Service call returns an exception with a connection error.

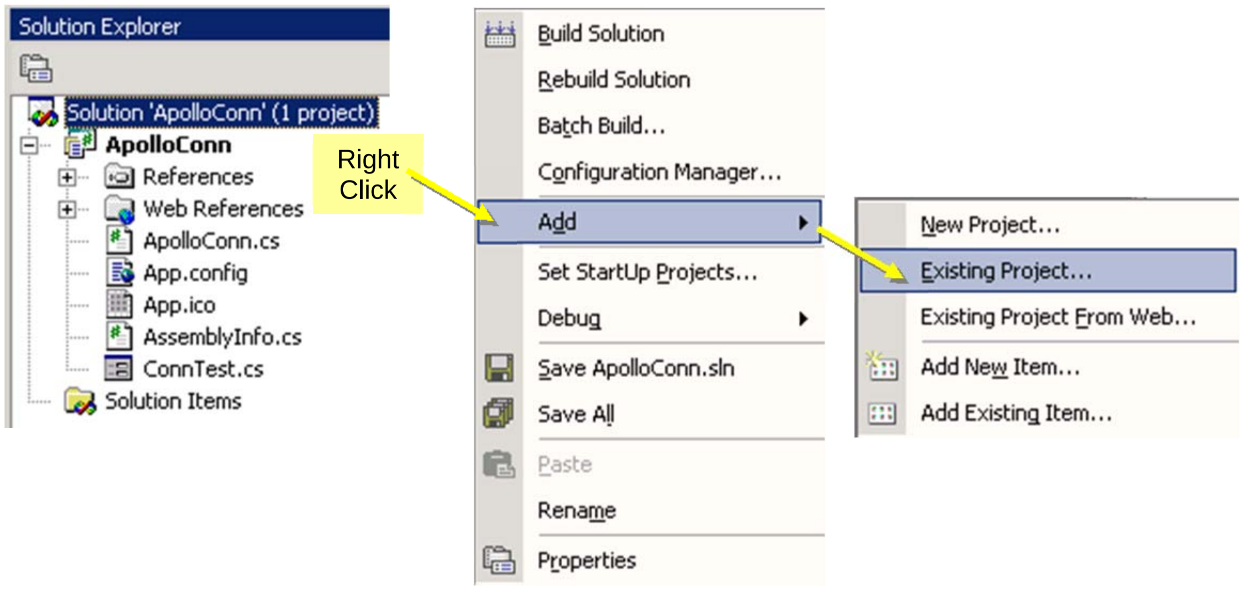
  Using a new socket for every connection can add a little connection time (usually only milliseconds if you have a fast network connection), but it is more reliable because you are less likely to encounter connection exceptions. For reliability, always set the KeepAlive property in the new class to *false* (which is now the default value).
- **The Nagle Algorithm**: The Nagle Algorithm was created to make TCP traffic more efficient by combining several small messages into one packet. For Web Services, this algorithm does not help, and can cause delays of up to 200 milliseconds. Galileo recommends turning off the Nagle Algorithm. In the new class, this is a property setting called UseNagleAlgorithm. The default value is *false*.
- **GZip compression**: GZip compression can be added to your client by using the GZipRequest and AcceptGZip properties provided by the optimization class.

- GZipRequest enables the outgoing request to be compressed.
- AcceptGZip provides the ability to receive and decode a compressed response.

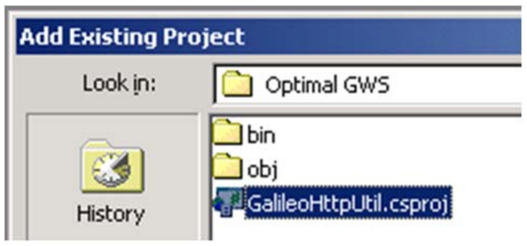
The two options are independent of each other. A setting of *true* enables compression. Compression can significantly reduce the transmitted size of large responses, such as the response from a Super Best Buy (FareQuoteSuperBB_#) request. The default values are *true*.

Using Optimization

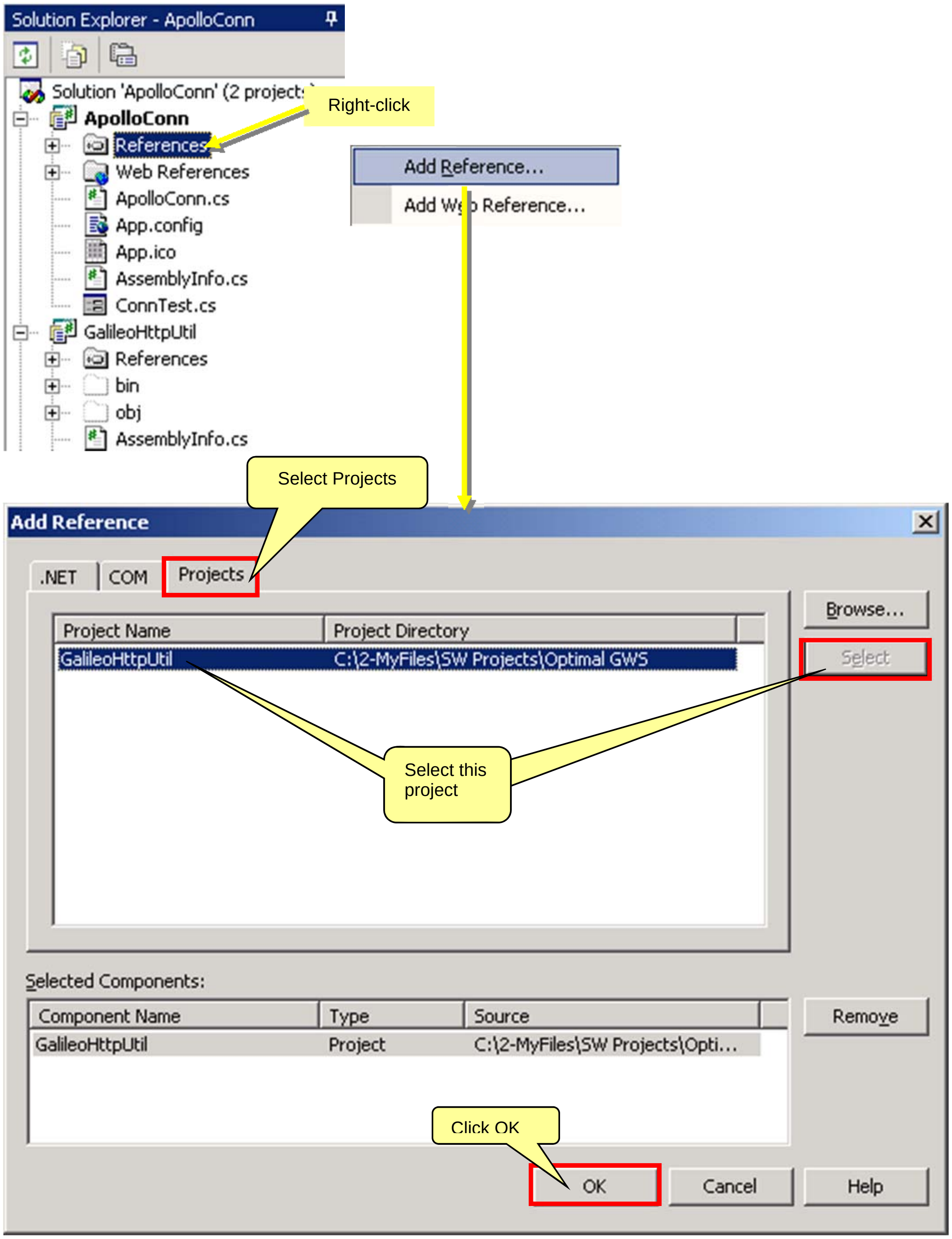
The code for the optimization class is available from the GWS Sample Site as a project. Download the project and unzip it into a folder. The easiest way to use the optimization is to add it to your solution as a project:



Browse to the location where you saved the optimization project and select the **.cproj** file and click **Open**:



The optimization project is now included in your solution as designated by the “2” in the Solution line of your project.



You must also add the Optimization Product to your Web Reference.

Adding the Optimization Product to your Web Reference

The optimization project is now included in your solution, but you still need to connect it to your existing Web Reference:

1. Modify the XmlSelect code (created in the connection class) that was automatically generated when you added the Web Reference. Find this code under the Web References /com.travelport.copy_webservices.americas and locate the line that defines the class, similar to:

   ```
   public class XmlSelect : System.Web.Services.Protocols.SoapHttpClientProtocol {
   ```

2. Comment this line out and add a new line similar to:

   ```
   public class XmlSelect : Galileo.Web.Services.EnhancedSoapHttpClientProtocol {
   ```

   The new line causes your Web Reference class to inherit from the new optimized class, which adds the properties and methods to your connection. Keep in mind that if you regenerate the Web Reference class for any reason, you need to repeat this modification.

   By way of inheritance, your connection class now includes the properties described above, as well as two methods in the optimized class that override the standard GetWebRequest and GetWebResponse classes. If you want to change the default values of the new properties, you can modify them in the EnhancedSoapHttpClientProtocol constructor. You can also change their values directly in the ApolloConn class.

Appendix A: The ApolloConn Class

```
using System;
using System.Configuration;
using System.Net;
using System.Xml;

namespace ApolloConn
{
        /// <summary>
        /// This class provides the actual connection to the Apollo GDS system for
        /// executing specific XML transactions. The goal of the class is to encapsulate the
        /// actual connection method and the specific credentials needed for access.
        /// Inherits from the Web Service proxy, so that all of the GWS transaction methods
        /// are available to this instance.
        /// This class also helps manage the session, creating a new session when needed.
        /// </summary>
        public class ApolloConn : com.travelport.copy_webservices.americas.XmlSelect
                {
                // Define variables used by multiple methods
                private string gwsHAP;
                private string token = "";          // Session token
                private XmlElement defaultFilter;

                //-------------------------------------------------------------------------------------
                public ApolloConn()
                {
                        // Default constructor. Create and set up the credentials for
                        // XMLSelect WebService.
                        string userName = ConfigurationSettings.AppSettings["GWSUSERNAME"];
                        string password = ConfigurationSettings.AppSettings["GWSPASSWORD"];
                        this.Url = ConfigurationSettings.AppSettings["GWSURL"];
                        gwsHAP   = ConfigurationSettings.AppSettings["GWSHAP"];

                        NetworkCredential netCredentials = new NetworkCredential(userName, password);

                        //Create the Credential Cache to assign to XMLSelectWebService client
                        CredentialCache cc = new CredentialCache();
                        //Xml Select uses Basic Authentication, but Windows XP defaults to Digest
                        cc.Add(new Uri(this.Url), "Basic", netCredentials);
                        Credentials = cc;
                        PreAuthenticate = true;

                        // Create a default filter document for use in the overloaded simplified requests
                        XmlDocument dFilter = new XmlDocument();
                        dFilter.LoadXml("<_/>");
                        defaultFilter = dFilter.DocumentElement;
                }

                //-----------------------------------------------------------------------------------
                public string SubmitXml(string request)
                {
                        // Simple overloaded version of SubmitXml transaction with a string request,
                        // adding the HAP and using the default filter
                        XmlDocument xmlRequest = new XmlDocument();
                        xmlRequest.LoadXml(request);
                        return this.SubmitXml(      this.gwsHAP,
                                                    xmlRequest.DocumentElement,
                                                    defaultFilter).OuterXml;
                }
                //-----------------------------------------------------------------------------------
                public XmlElement SubmitXml(XmlElement xmlRequest)
                {
                        // Simple overloaded version of SubmitXml transaction
                        // Uses an XmlElement input and adds the HAP and default filter
                        return this.SubmitXml(this.gwsHAP, xmlRequest, defaultFilter);
                }
                //-----------------------------------------------------------------------------------
                public string SubmitXmlOnSession(string request)
                {
                        // Simple overload for session requests. String input, adds HAP and filter
                        // and begins a session if one does not exist.
                        XmlDocument xmlRequest = new XmlDocument();
                        xmlRequest.LoadXml(request);
                        // Create a session if one does not already exist
```

```
                if (this.token == "") this.token = this.BeginSession(this.gwsHAP);
                return this.SubmitXmlOnSession(this.token,
                                                        xmlRequest.DocumentElement,
                                                        defaultFilter).OuterXml;
            }
            //-----------------------------------------------------------------------------------
            public void EndSession()
                // Simple overload that erases the session token
            {
                this.EndSession(this.token);
                this.token = "";
            }

        }
    }
```

For completeness, methods should be added for:

- GetIdentityInfo
- MultiSubmitXml
- SubmitTerminalTransaction
- SubmitCruiseTransaction

Appendix B: Prototype App.Config File

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
        <appSettings>
                <add key="GWSUSERNAME" value="yourUserName" />
                <add key="GWSPASSWORD" value="yourPassword" />
                <add key="GWSURL" value=" https://americas.copy-webservices.travelport.com/B2BGateway/service/XMLSelect
" />
                <add key="GWSHAP" value="yourHAP" />
        </appSettings>
</configuration>
```

Other parameters, such as the Pseudo City Code (PCC), can be added using the same format.