API Developer Notes

A Galileo Web Services Java Connection Class Using Axis

29 June 2012

Version 1.3

THE INFORMATION CONTAINED IN THIS DOCUMENT IS CONFIDENTIAL AND PROPRIETARY TO TRAVELPORT

Copyright

Copyright © 2012 Travelport and/or its subsidiaries. All rights reserved.

Travelport provides this document for information purposes only and does not promise that the information contained in this document is accurate, current or complete. This document is subject to change without notice.. No part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or any means electronic or mechanical, including photocopying and recording for any purpose other than the licensee's personal use without the prior written permission of Travelport and/or its subsidiaries.

Trademarks

Travelport and/or its subsidiaries may have registered or unregistered patents or pending patent applications, trademarks copyright, or other intellectual property rights in respect of the subject matter of this document. The furnishing of this document does not confer any right or licence to or in respect of these patents, trademarks, copyright, or other intellectual property rights.

All other companies and product names are trademarks or registered trademarks of their respective holders.

Contents

Overview

When developing a Java client application that uses Galileo Web Services, there are several options for tools to generate the code that connects your application to the Web Service interfaces. A popular option is to use the Axis tools to generate the stub that your application then calls to access Galileo Web Services (GWS). 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

Before you can use the Axis tools, you must set up the Axis environment. Setting up the environment requires you to:

- Download the Axis tools.
- Download the Web Service Definition File.
- Generate Stubs Using Axis.
- Install the appropriate Java SDK security certificates.

Downloading the Axis Tools

You can download the Axis Tools from the Apache web site.

1. Open the Apache web site at http://ws.apache.org/axis/.
2. Download the Axis tools (**axis-1_1.zip**).
3. Unzip the file to project folder, where it automatically creates a file structure that starts with **axis-1_1**.

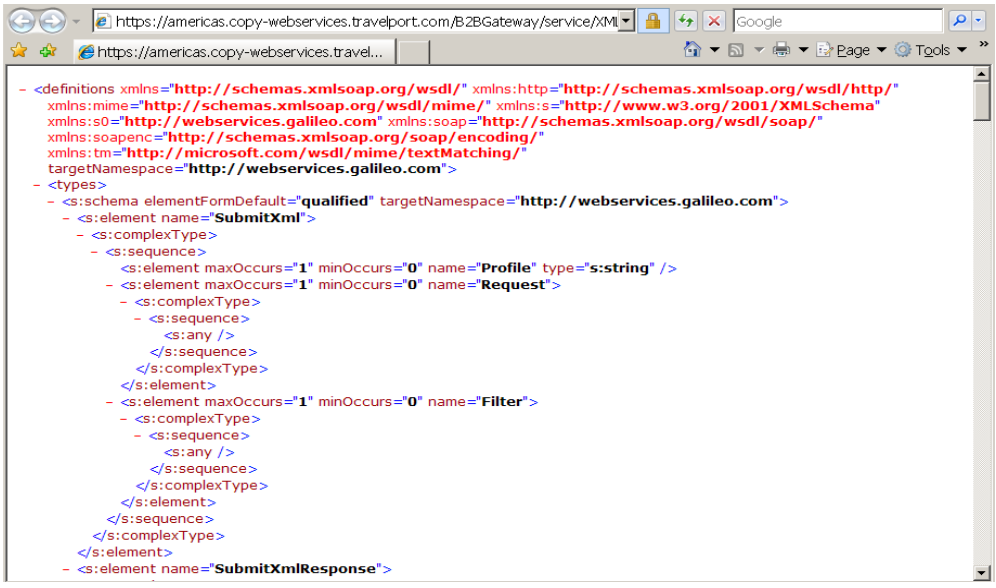
Downloading the WSDL File from GWS

The WSDL file can be downloaded from the GWS site.

4. Open the WSDL for 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

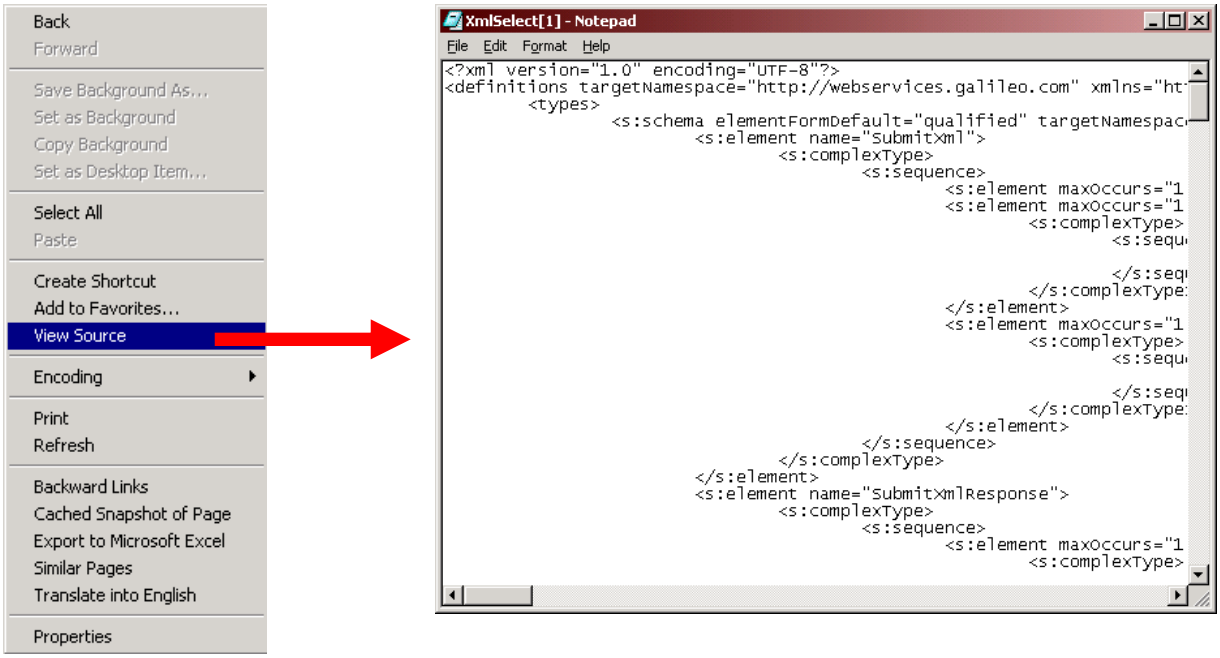
A window similar to the following displays:



Presently, the Axis tools do not interpret the Galileo WSDL as intended.

5. Save a copy of the WSDL file to your computer.

   a. Right-click on the text and select View Source. The source code displays, as shown:



   b. Save the file as **XMLSelect.wsdl** in the **axis-1_1\lib** directory that you created in the preceding *Downloading the Axis Tools* topic.

6. Edit the XMLSelect.wsdl file that you saved.

Editing the WSDL File

Throughout the WSDL file, there are sequences that include the type definition **any**. The Axis tools do not process this type very well, so you must modify the sequences to something Axis handles correctly. When you find a sequence similar to the one shown as the *Original* below, modify it to the entry shown as *Modified*.

Original

```
<s:element maxOccurs="1" minOccurs="0" name="Request">
      <s:complexType>
            <s:sequence>
                  <s:any/>
            </s:sequence>
      </s:complexType>
</s:element>
```

Modified

```
<s:element maxOccurs="1" minOccurs="0" name="Request"
type=”apachesoap:Element”/>
```

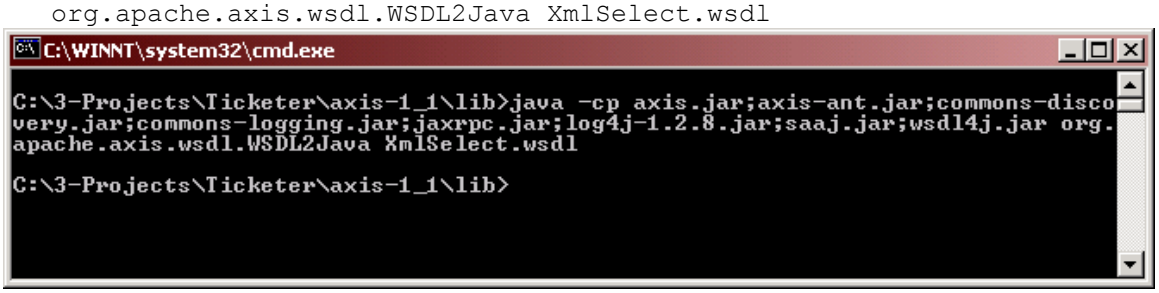
The modified WSDL for the XML Select Web Service is shown in *Appendix A: ApolloConn Class: Modified WSDL for the XML Select Web Service*.

Generating Stubs Using Axis

Now that the WSDL is Axis-ready, use Axis to generate the stubs that your application will call to create a connection and to execute transactions using the Web Service.

1. Open a command window and navigate to the **axis-1_1\lib** directory.
2. Execute the following command line:

```
java     -cp axis.jar;axis-ant.jar;commons-discovery.jar;commons
-logging.jar;jaxrpc.jar;log4j-1.2.8.jar;saaj.jar;wsdl4j.jar
org.apache.axis.wsdl.WSDL2Java XmlSelect.wsdl
```



A file structure is created in the **lib** directory that looks like:

```
com
   galileo
         webservices
               XmlSelect.java
               XmlSelectLocator.java
               XmlSelectSoap.java
               XmlSelectSoapStub.java
```

The preceding structure and the four files in this package need to be incorporated into your Java project.

Installing Security Certificates

Galileo Web Services operates in a secure SSL environment. The Java SDK versions 1.4.2 and above have the necessary security certificates built in. However, earlier versions of the Java SDK require security certificates.

Accessing the XML Select Web Service Using the Connection Class Method

After the environment is set up, you can begin writing code. The following example shows a connection class that accesses the XML Select Web Service. The intention of the connection class is to encapsulate all of the details of the connection to the Apollo system, including not only the Web Service methods, but also the credentials needed to access the system. The class methods take a simple XML string request and add the necessary formatting and credentials to make the Web service call.

Sessionless transactions are used for air, car, and hotel availability, as well as all cruise transactions.

Connection Class Method Example

The following steps show how to build a connection class. Similar steps can be followed for other classes.

1) Import classes from the Axis tools for your connection class. To include these classes, use:

```
import org.w3c.dom.*;
import com.galileo.webservices.*;
```

2) Define the variables that will be used throughout the class:

```
private XmlSelectSoapStub xws;   // Represents the connection
private String gwsHap;                  // Galileo HAP
private Element defaultFilter;   // Default response filter
private String token = "";          // Session token
```

The connection class uses a simple **AppConfig** class to read configuration information from an XML formatted file. You can also use the standard **Java System.getProperty**. A simple version of AppConfig is included in *Appendix* .

First, the constructor opens the configuration file and reads the endpoint (URL) of the Web Service:

```
// Open the configuration file and get the URL of the Web service
AppConfig config = new AppConfig("GWSConfig.xml" );
String gwsUrl = config.getParam( "GWSURL" );
```

The URL is included in the WSDL file, but it is a good idea to have it in the configuration file in case something changes (such as moving your application from the copy system to production).

3) Hook the SSL certificate to the application. The hook requires two lines, which specify the key store file and password. The SSL certificate must be hooked to the application before you try to bind to the Web Service.

```
// Hook the SSL certificate to this application
System.setProperty("javax.net.ssl.trustStore", "C:\\truststorecopy");
System.setProperty("javax.net.ssl.trustStorePassword", "trustword");
```

4) Set up the connection information. Axis uses a three-step process.

   a) Instantiate a connection locator using the **XmlSelectLocator** class that was created above:

```
// Establish a connection locator
XmlSelectLocator xwsLocator = new XmlSelectLocator();
```

   b) Use the locator you instantiated to create the connection object that the application will use to invoke the Web Service transactions:

```
// Use the locator to generate a stub for the service interface
```

```
xws = (XmlSelectSoapStub) xwsLocator.getXmlSelectSoap(new
java.net.URL(gwsUrl));
```

c) Add your credentials to the connection object by reading them from the configuration file:

```
// Set the credentials
xws.setUsername(config.getParam( "USERNAME" ));
xws.setPassword(config.getParam( "PASSWORD" ));
gwsHap = config.getParam( "GWSHAP" );
```

The connection is now represented by the object xws.

5) Set up a default filter that can be added to each of the access methods. The filter uses another method in the connection class to create a Document from a string and then convert the Document into an Element (**string2Element** is shown in *Appendix*).

```
defaultFilter = string2Element("<_/>");
```

Editing the constructor is completed.

6) Add the XmlSubmit method to submit non-sessioned XML Select transactions. The XmlSubmit method accepts the XML-formatted request string and converts it into an 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.

```
String XmlSubmit(String request) throws Exception
// Simple interface for XmlSubmit. Adds the HAP & filter,
// and converts request to an XML Element
{
   Element Xmlrequest = string2Element(request);
   Element submitXmlResponse = xws.submitXml(gwsHap, Xmlrequest,
defaultFilter);
   return submitXmlResponse.toString();
}
```

A similar method that returns an Element can be added by simply replacing the **String** return type with **Element** and removing the **toString** in the return line.

Accessing the XML Select Web Service Using Sessioned Transactions

Sessioned transactions are only a little more work than sessionless. In the following example, the XMLSessionSubmit method creates a session if one does not already exist.

XMLSessionSubmit Method Example

The session identifier is held in the **token** variable. The rest of the method looks very similar to XmlSubmit.

```
String XmlSessionSubmit(String request) throws Exception
 // Simple interface for SubmitXmlOnSession. Adds the HAP & filter,
 // and converts request to an XML Element.
 // Begins a session if one does not exist.
 {
      if(token == "") token = xmlSel.BeginSession(gwsHap);
      Element Xmlrequest = string2Element(request);
      Element response = xmlSel.SubmitXmlOnSession(token, Xmlrequest,
defaultFilter);
      return response.toString();
}
```

To end the session you must call 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:

1. Create a request string.
2. Create an instance of the connection class.
3. Submit a transaction.

To execute the XML Select transaction to get a local time, the code looks similar to:

```
        String request = "<LocalDateTimeCT_6_0><LocalDateTimeMods>"
                       + "<ReqCity>DEN</ReqCity>"
                       + "</LocalDateTimeMods></LocalDateTimeCT_6_0>";
    try
{
            ApolloConn host = new ApolloConn();
            String response = host.XmlSubmit(request);
           // Process the response here
        }
        catch (Exception e)   // Put your error code here
{
            responseDisp.setText(e.toString());
}
```

Appendix A: ApolloConn Class

The ApolloConn class encapsulates the XML Web Service interface. This class reads the credential information from a configuration file and establishes the connection. The class also provides simplified interfaces to the XML Web Service by adding HAP and filter parameters. For completeness, methods should be added for

- GetIdentityInfo
- MultiSubmitXml
- SubmitTerminalTransaction
- SubmitCruiseTransaction

```
//package Projects.Ticketer.Files;

import com.galileo.webservices.*;
import org.w3c.dom.*;
import java.io.*;

class ApolloConn
{
    private XmlSelectSoapStub xws;  // Represents the connection
    private String gwsHap;                // Galileo HAP
    private Element defaultFilter;  // Default response filter
    private String token = "";          // Session token

//---------------------------------------------------------------------
    ApolloConn() throws Exception
    {
        // Open the configuration file and get the URL of the Web service
        AppConfig config = new AppConfig("GWSConfig.xml" );
        String gwsUrl   = config.getParam( "GWSURL" );

        // Hook the SSL certificate to this application
        System.setProperty("javax.net.ssl.trustStore", "C:\\truststorecopy");
        System.setProperty("javax.net.ssl.trustStorePassword", "trustword");

        // Establish a connection locator
      XmlSelectLocator xwsLocator = new XmlSelectLocator();

        // Use the locator to generate a stub for the service interface
      xws = (XmlSelectSoapStub) xwsLocator.getXmlSelectSoap(new
java.net.URL(gwsUrl));

        // Set the credentials
        xws.setUsername(config.getParam( "USERNAME" ));
        xws.setPassword(config.getParam( "PASSWORD" ));
        gwsHap = config.getParam( "GWSHAP" );
        defaultFilter = string2Element("<_/>");
    }

//---------------------------------------------------------------------
      String XmlSubmit(String request) throws Exception
```

```
        // Simple interface for XmlSubmit. Adds the HAP & filter,
        // and converts request to an XML Element
      {
            Element Xmlrequest = string2Element(request);
            Element submitXmlResponse = xws.submitXml(gwsHap, Xmlrequest,
defaultFilter);
            return submitXmlResponse.toString();
      }

//---------------------------------------------------------------------
      Element XmlSubmit(Element request) throws Exception
        // Simple interface for XmlSubmit. Adds the HAP & filter,
        // and converts request to an XML Element
      {
            Element submitXmlResponse = xws.submitXml(gwsHap, request,
defaultFilter);
            return submitXmlResponse;
      }
//---------------------------------------------------------------------
      String XmlSessionSubmit(String request) throws Exception
        // Simple interface for submitXmlOnSession. Adds the HAP & filter,
        // and converts request to an XML Element.
        // Begins a session if one does not exist.
        {
            if(token == "") token = xws.beginSession(gwsHap);
            Element Xmlrequest = string2Element(request);
            Element response = xws.submitXmlOnSession(token, Xmlrequest,
defaultFilter);
            return response.toString();
      }

//---------------------------------------------------------------------
      void EndSession() throws Exception
        // Simple interface to WS endSession that erases the session token
      {
            xws.endSession(token);
            this.token = "";
      }

//---------------------------------------------------------------------
       public static Element string2Element(String s) throws
javax.xml.parsers.ParserConfigurationException,
                                  java.io.IOException,
org.xml.sax.SAXException
      {
            return org.apache.axis.utils.XMLUtils.newDocument
                    (new org.xml.sax.InputSource(
new StringReader(s))).getDocumentElement();
      }
}
```

Appendix B: Simplified AppConfig Class

```
// Configuration file class for reading and writing application configuration
parameters
// This is a generic configuration utility, using the Electric XML libraries
// The configuration file is assumed to be one level deep, as in the
following format:
//
//          <?xml version='1.0' encoding='UTF-8'?>          ** this line
optional
//          <config>
//                <ParamName1>parameter 1</ParamName1>
//                            . . .                   ** as necessary
//                <ParamNameN>parameter n</ParamNameN>
//          </config>
//
// The parameter names and values are arbitrary Strings (within the XML
standard)

import java.io.*;
import electric.xml.*;

public class AppConfig
{
      protected Document docIn;

//---------------------------------------------------------------------
      // Read the XML configuration file named fileName
      public AppConfig(String fileName) throws Exception
      {
            docIn = new Document ( new File( fileName ) );
      }

//---------------------------------------------------------------------
      // Get a specified parameter from the current config file document.
      public String getParam ( String paramName ) throws Exception
      {
            if ( docIn == null ) return null;
            Element root = docIn.getRoot();
            if ( root.getElement( paramName ) != null )
                  return root.getElement( paramName ).getTextString();
            else return null;
      }
}
```

The configuration file looks similar to:

```
<config>
  <GWSURL>https://americas.copy-
webservices.travelport.com/B2BGateway/service/XMLSelect</GWSURL>
  <WSDLURL> https://americas.copy-
webservices.travelport.com/B2BGateway/service/XMLSelect?WSDL</WSDLURL>
  <USERNAME>yourUserName</USERNAME>
  <PASSWORD>yourPassword</PASSWORD>
  <GWSHAP>yourHAP</GWSHAP>
  <PCC>yourPcc</PCC>
</config>
```