API Developer Notes

Migrating from XML Select to GWS

29 June 2012

Version 1.4

Contents

Overview

An application built on Galileo's XML Select API can be a valuable addition to your travel business. However, the deployment of an XML Select application also carries the responsibility of maintaining your GTIDs, the infrastructure of your XML Select server complex, and dedicated communication lines to Galileo.

The primary incentives to move from XML Select to the Galileo Web Services (GWS) XML Select API are:

- GTID management that is handled entirely by Galileo.
- No XML Select servers to maintain at your site.
- Less costly Internet connections to communicate with the Computer Reservation System (CRS).

These factors can mean significant operational cost savings. In addition, the GWS API adds new functionality that can improve the performance and efficiency of your travel applications and allows development in non-COM compatible languages.

See the *Getting Started* topic in the Galileo Web Services Help for details about design considerations for your client application, connecting to Galileo Web Services, and making initial test calls to the Galileo Web Services.

XML Select to GWS Migration

All XML Select functionality remains in the Web Service version of XML Select. However, the commands to start and end Host (CRS) sessions are different, although still very simple. Additionally, creating and maintaining Host sessions is slightly different in the Web Service environment.

Host sessions are difficult to maintain for long periods of time when no transactions occur, because GWS sessions time out in three minutes. As a result, your session strategy could require reevaluation.

The amount of effort needed to move from XML Select to GWS is highly dependent on the architecture of your application. The issues that need to be explored include:

- Is SUTA the interface to XML Select?
- Do you create and maintain Host sessions for long periods of time?
- Are the calls to XML Select centralized, or are they dispersed throughout the application?
- Are the credentials hard-coded, or are they kept in a configuration file?
- Does the language used to develop your application support SOAP tools?

If your application does not use long Host sessions, but does use one centralized class to send XML Select requests to and receive responses from the SUTA object, and if the XML Select credentials are known only to that class, the transition to GWS is trivial. You can replace the centralized class with a new one and the process is done. However, if your application uses long host sessions and multiple classes, the Migration Plan and Appendices in this document provide further details.

Migration Plan

Your plan to migrate to GWS must include a review of:

- Host sessions.
- Methods used to send requests and receive responses.
- Available SOAP tools for your development environment.

SUTA Interface

This document addresses the conversion of applications that use the XML Select SUTA interface, which is similar to the Galileo Web Services interface. Applications that interface directly to UTA, the HCM, and the HCM Manager are considerably more difficult to migrate, and may not be candidates for conversion to GWS.

Sessions

Generally, Host sessions are only a problem if more than three minutes can elapse with no activity. This typically occurs when the application is waiting for human input, such as a traveler entering their personal information. The first rule for GWS applications is: never hold a session while waiting for a person to act. Although an application can keep a session alive by periodically issuing a useless request, such as local date and time, this increases your transaction count, and can result in Excessive Transaction Charges (ETAs).

An application can keep a session open as long as necessary if it is interacting with the CRS because each transaction request resets the three-minute timeout. Therefore, an application that opens a session, signs into a queue, and works even a large number of PNRs on the queue can easily keep a session open.

You should avoid scenarios such as opening a session, selling airline seats, and then waiting for the traveler to input their name, address, phone number, and credit card information. If your application requires this operating model, GWS may not be a viable option, unless you are willing to change the application architecture and program flow.

If your application follows a more typical online model of collecting both itinerary information and passenger information before beginning a booking, then a session can be opened and all of the booking transactions completed within a single session.

Sending and Receiving Transactions

The preferred architecture for submitting requests and receiving responses is to create a single connection class that the entire application can use. The purpose of this class is to encapsulate the actual access method, and to hide the access credentials from the rest of the application. The single connection class also centralizes the credentials so that only the connection class is affected if a password must be changed.

If your application is not structured this way, the first step is to re-factor the code to use a connection class. Search through the source files for methods such as *SyncSubmit*, *TerminalSubmit*, *BeginSession,* and *EndSession*, as well as your identity string.

Each time you find a request, be sure to note if the request is submitted within a Host session. Also, note the data types that are used in the request and response. For an XML Select application, both the request and the response data are typically an XML-formatted string. Galileo Web Services sends requests and returns responses as XML documents, but this conversion can be accomplished in the connection class.

After all the transaction locations are identified, create a connection class to replace the individual calls to the XML Select SUTA COM object. In addition to the constructor, the class must have at least the following methods:

Method	Description
`SubmitXml(string request)`	▪ For non-session requests ▪ Uses the default filter
`SubmitXmlOnSession(string request)`	▪ Creates a session if necessary ▪ Uses the default filter
`EndSession()`	▪ For session requests ▪ Ends the current session

An example of a C# connection class is included in *Appendix A: XML Select Connection Class*. You can add methods for terminal transactions or for requests that use filters.

Note that the connection class makes a distinction between sessioned and non-sessioned requests, whereas XML Select uses the same method for both. Sessions are created automatically as needed when *SubmitXmlOnSession* is called, so *BeginSession* calls can be removed. The connection class also uses a configuration file to hold the credentials to allow passwords to change without re-compiling the code.

Moving to GWS

After your application is working with the new connection class, you can complete the transition to GWS by replacing the connection class with an equivalent class that uses Galileo Web Services. An example of a GWS connection class is included in *Appendix B: GWS Connection Class*. The ***API Developer Notes: GWSdotNETConnectionClassUsingC#*** describes the development of a GWS connection class with optimization and Gzip compression in C# for use with any .NET language. API Developer Notes for Java are also available.

Planning is the key to a successful migration. Analysis of session activity and interface call patterns will determine if your application is suitable for Galileo Web Services. The process described in this document allows for an orderly development as well as a fall back plan by reverting to the XML Select connection class if necessary.

Mapping of XML Select to GWS Methods

The following list provides equivalent methods between XML Select and GWS.

XML Select Methods	GWS Methods
SyncSubmit	SubmitXML SubmitXmlOnSession MultiSubmitXml
TerminalSubmit	SubmitTerminalTransaction
GetTerminalBuffer*	
BeginSession	BeginSession
EndSession	EndSession
	GetIdenityInfo**

*GetTerminalBuffer retrieves a second response from a terminal request. GWS only returns one terminal response.

**Identity information cannot be displayed via XML Select. Instead, it is provided to the end user with their provisioning and is submitted with the calls.

Appendix A: XML Select Connection Class

```
using System;
using SCRIPTABLEUNIVERSALTRANSAGTLib;
using System.Configuration;

namespace XMLSampleApp
{
        /// <summary>
        /// ApolloConn is the abstracted connection class to the Apollo GDS. The
        /// intention of this class is to create a single point of connection to
        /// GDS that can be easily replaced by a GWS connection class
        /// </summary>
        internal class ApolloConn
        {
                ScriptableUniversalTransAgent xmlConn;
                // Set up the static information strings (From App.config file)
                static string identity = "<Application><VendorId>"
                + ConfigurationSettings.AppSettings["VENDORID"] + "</VendorId>"                                 +
"<VendorType>" + ConfigurationSettings.AppSettings["VENDORTYPE"] + "</VendorType>"
                + "<SourceId>" + ConfigurationSettings.AppSettings["SOURCEID"] + "</SourceId>"
                + "<SourceType>" + ConfigurationSettings.AppSettings["SOURCETYPE"]
                + "</SourceType></Application>"
                + "<User><UserId>" + ConfigurationSettings.AppSettings["USERID"] + "</UserId>"
                + "<Pseudo>" + ConfigurationSettings.AppSettings["PCC"] + "</Pseudo></User>";
                private bool sessionActive = false;
                //-------------------------------------------------------------------------
                internal ApolloConn()
                {
                        // Establish the connection to Apollo via the SUTA
                        xmlConn = new ScriptableUniversalTransAgentClass();
                        // Set the HCM name (From App.config file)
                        xmlConn.HcmName = ConfigurationSettings.AppSettings["HCMNAME"];
                }
                //-------------------------------------------------------------------------
                internal string SubmitXml(string request)
                {
                        // Submit the request using the default filter
                        return xmlConn.SyncSubmit(idenity, request, "<_/>");
                }
```

```
        //-------------------------------------------------------------------------------
        internal string SubmitXml(string request, string filter)
        {
                // Submit the request using the provided filter
                return xmlConn.SyncSubmit(idenity, request, filter);
        }
        //-------------------------------------------------------------------------------
        internal string SubmitXmlOnSession(string request)
        {
                // Simple overload for session requests. String input, adds ID and filter
                // and begins a session if one does not exist.
                // Create a session if one does not already exist
                if (!sessionActive)
                {
                  xmlConn.BeginSession(System.Convert.ToInt16("0xFFFFFFFF", 16));
                  sessionActive = true;
                }
                return xmlConn.SyncSubmit(idenity, request, "<_/>");
        }
        //-------------------------------------------------------------------------------
        internal void EndSession()
                // Simple overload that erases the session token
        {
                xmlConn.EndSession(System.Convert.ToInt16("0xFFFFFFFE", 16));
                sessionActive = false;
        }
    }
}
```

Appendix B: GWS Connection Class

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

namespace ApolloConnection
{
        /// <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.
        /// ApolloConnection is marked internal, as it is intended that it be used only
        /// by classes in the ApolloAccess class library.
        /// Inherits from the Web Service proxy, so that all of the GWS transaction methods
        /// are available to this instance.
        /// This class also manages the session, creating a new session when needed.
        /// The public class location is an example. To determine the location for you, see
        /// http://testws.galileo.com/GWSSample/Help/GWSHelp/connecting_to_gws.htm
        /// </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()
                {
                        // 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["URL"];
                        gwsHAP   = ConfigurationSettings.AppSettings["GWSHAP"];
                        NetworkCredential netCredentials = new NetworkCredential(userName, password);
                        //Create 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 XmlElement SubmitXml(string request)
{
            // Simple overloaded version of SubmitXml transaction with a string request,
            // using the default filter
            XmlDocument xmlRequest = new XmlDocument();
            xmlRequest.LoadXml(request);
            return this.SubmitXml(this.gwsHAP, xmlRequest.DocumentElement, defaultFilter);
}
//-----------------------------------------------------------------------------
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 XmlElement SubmitXml(string request, string filter)
{
            // Simple overloaded version of SubmitXml transaction with string inputs
            // Convert the strings to XML Documents
            XmlDocument xmlRequest = new XmlDocument();
            XmlDocument xmlFilter  = new XmlDocument();
            xmlRequest.LoadXml(request);
            xmlFilter.LoadXml(filter);
            return this.SubmitXml( this.gwsHAP,
                                   xmlRequest.DocumentElement,
                                   xmlFilter.DocumentElement );
}
```

```
        //-----------------------------------------------------------------------------
        public void EndSession()
                // Simple overload that erases the session token
        {
                this.EndSession(this.token);
                this.token = "";
        }
        //-----------------------------------------------------------------------------
        public XmlElement 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 );
        }
        //-----------------------------------------------------------------------------
        public XmlElement MultiSubmitXml(string request)
        {
                // Simple overloaded version of MultiSubmitXml transaction with a
                // string request, adding the HAP
                XmlDocument xmlRequest = new XmlDocument();
                xmlRequest.LoadXml(request);
                return this.MultiSubmitXml(this.gwsHAP, xmlRequest.DocumentElement);
        }
        //--Properties---------------------------------------------------------------
        public string HostProfile
        {
                get { return gwsHAP; }
        }
    }
}
```