AI Developer Notes

Galileo Web Services Implementation Best Practices and Development Guidelines

11 October 2011

Version 1.3

Contents

Implementation Best Practices

Overview

This section outlines options, recommendations, and best practices for implementing a software application that uses Galileo Web Services (GWS). This section should be used in conjunction with the Development Guidelines section.

This guide assumes familiarity with Galileo Web Services, SOAP, XML, Java and/or .Net programming languages.

Introduction and Objectives

Galileo Web Services (GWS) allow developers to access most of the CRS content (air, hotel, and car) using structured XML requests via a Web services wrapper.

It is very important to optimize the source code to get the maximum performance from the system. This document can be used as a reference to realize a level of optimization that helps to achieve better response times and reduce the system's overhead at peak times.

The examples depicted in this document are in .NET or Java, but this code can be readily converted to any appropriate programming language. Galileo does not provide any support on the techniques or particulars of any programming language, and the examples within this document are for reference only.

The term "pipelining" is used as a synonym for the term "connection pooling". However, the use of these terms in this document is:

- *Connection Pooling* – Reusing an HTTP connection for multiple HTTP requests. When the connection is idle, it is kept in a pool of available connections.
- *Pipelining* – Performing multiple, simultaneous HTTP requests on a single HTTP connection.

Java Developers can use the following URL (http://java.sun.com/j2se/1.5.0/docs/guide/net/http-keepalive.html) to learn more about how to reuse TCP connections when calling a Web Service.

Special HTTP Request Parameters

The following recommendations for HTTP request parameters should be used when transacting with GWS.

Expect 100-Continue

Overview

The purpose of the **100-Continue** status (on the Internet, find and refer to section 10.1.1 of the Request for Comment (RFC) 2616 for more details) is to allow a client who sends a request message with a request body to determine if the origin server will accept the request (based on the request headers) before the client sends the request body. It is highly inefficient for the client to send the body if the server will reject the message without looking at the body.

Effect

Using 100-Continue adds anywhere from 25 to 150 milliseconds on a typical transaction, plus any network propagation delays.

Recommendation

It is recommended that 100-Continue be disabled when using GWS services. In Java, 100-Continue is normally disabled by default. In .NET, the Expect100Continue property of the ServicePointManager defaults to **true**.

Note: Expect100-Continue is valid only for HTTP 1.1 and above.

KeepAlive

Overview

When using the **Connection: KeepAlive** header on HTTP 1.1, the .NET Framework pools HTTP connections. Connection pooling can provide some savings by eliminating the time to establish a connection (when connections are available in the pool). However, depending on whether connections in the pool get closed by the server or network, a Web Service call can get 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 also more reliable because connection exceptions are less likely to occur.

With Windows, the limit of 10 outstanding SYN packets can force the use of KeepAlive to gain performance if more than 10 concurrent connections setups are anticipated. Windows XP SP2 and Windows 2003 Server have this limit hard coded. You can change the limit for Windows 2000 and Windows XP (up to SP1) by changing the registry; however, registry changes are not supported by Microsoft.

Recommendation

It is not possible to make a generic recommendation as many factors, such as projected load and OS can affect performance.

Pre-Authenticate (.NET)

Pre-Authenticate authenticates user credentials before they are sent to the server so the server does not have to challenge the credentials. Therefore, the unnecessary exchange of packets is avoided, which yields a small reduction in response times.

.NET developers must send the pre-authentication in the HTTP request to avoid receiving an unauthorized error response from the server, which forces the client to resend the credentials in another request. Setting up pre-authentication ensures the client application is only required to send one request and wait for one host response, which improves the overall response time of the system. For further information about pre-authenticating requests, please log on to the AIS support web site:

http://testws.galileo.com/GWSSample/Help/GWSHelp/gzip_c_sharp_.net_gziphttpwebrequest.cs.htm

GZIP Compression

It is important to consider the amount of data that needs to be sent back and forth for a GWS transaction. The following table illustrates some examples of transactions that are suitable for compression and their data traffic savings:

Transaction	Request with no compression	Request with GZIP	Response with no compression	Response with GZIP	Savings for response
FareQuoteSuperBB (round trip & KLR	2687 Bytes	704 Bytes	41034 Bytes	2714 Bytes	93%

optimization)					
FareQuoteSuperBB (round trip with no KLR optimization)	2413 Bytes	516 Bytes	219648 Bytes	10532 Bytes	95%
AirAvailability_6_5	644 Bytes	232 Bytes	26496 Bytes	1265 Bytes	95%
LocalDateTimeCT_6_0	120 Bytes	62 Bytes	112 Bytes	71 Bytes	47%
MultiSubmit with 2 SuperBB	6874 Bytes	806 Bytes	44600 Bytes	2705 Bytes	94%

The preceding table shows an average traffic size saving of 94% by using GZip compression for the response. However, it is also significantly important for the requests. GZip becomes more effective for large transactions such as FareQuoteSuperBB_# and AirAvailability_#, and when using MultiSubmit.

To implement GZip at request time, developers must set the HTTP header field **Content-Encoding** as **gzip** to indicate to the Galileo server that the Web Service request is zipped and ensure that the request is GZip compressed.

The client application must compress the request before encapsulating it into the HTTP request packet, and unzip the response after it is received.

- .NET does not natively support sending GZip-compressed requests to Web Services. You must implement this yourself using either third-party compression libraries (DLLs), such as SharpZipLib (http://www.icsharpcode.net/OpenSource/SharpZipLib/), or the native System.IO.Compression.GZipStream in .NET 2.0 and later.
- In Java, you can use GZIPOutputStream to zip the request.

Developers can also specify that they want to receive the host responses using GZip compression by setting the HTTP request header field **Accept-Encoding** to **gzip**.

- Using .NET 1.0 and 1.1, you must use third-party compression libraries (DLLs) because these versions do not support GZip by default.
- .NET 2.0 and later supports receiving GZip compression by default via the EnableDecompression property of the HttpWebClientProtocol class. For earlier versions of .NET, you must use third-party compression libraries (DLLs) because these versions do no support GZip by default.
- In Java, you can use GZIPInputStream to unzip the response.

Examples can be found at ais.galileo.com.

.NET ZIP examples can be found at:

http://testws.galileo.com/GWSSample/Help/GWSHelp/gzip_c_sharp_.net.htm

Recommendation

Use GZip as often as is practical to alleviate network bandwidth and significantly speed up the response time for large transactions with GWS.

Note: Using GZip can add a slight delay on small transactions, or on processor or hard-disk bound machines.

Other Parameters

Maximum Connections

.Net

The default two-connection limit for connecting to a Web resource can be controlled via a configuration element called **connectionManagement.** The connectionManagement setting allows you to add the names of sites where you want a connection limit that is different than the default.

.Net 1.1

Add the following code snippet to a typical Web.config file to increase the default value for all servers you are connecting, to a connection limit of 500.

```
<configuration>
  <system.net>
    <connectionManagement>
      <add address="*" maxconnection="500" />
    </connectionManagement>
  </system.net>
  <system.web>
```

.Net 2.0

The following code gets or sets the maximum number of connections that can be made to a remote computer.

Namespace: System.Net.Configuration

Assembly: System (in system.dll)

```
public int MaxConnection { get; set; }
```

Recommendation

The connectionManagement element should be set to correspond to your system's projected peak load.

Note: Failure to set connectionManagement correctly can severely impact performance.

SYN Connection Limit

Protect Against SYN Flood Attacks (Windows XP/2003)

Windows includes protection that limits the maximum unacknowledged SYN packets to 10. The limit prevents the propagation of viruses that use this technique from flooding the internet. However, this limit also prevents legitimate performance systems from rapidly opening new sockets at the same rate. Its affect on most production systems is small, but is often seen as very slow responses during performance testing.

The limit is hard-coded directly into the tcpip.sys file in Windows XP SP2 and Windows 2003 Server SP1. Earlier systems, such as Windows 2000 Windows XP SP1, do not have a hard-coded limit.

Recommendation

If it is likely that your system will require more that 10 sockets to be initialized simultaneously, consider using an alternative OS. Also consider using KeepAlives to alleviate this issue. See *KeepAlive* for details.

Special GWS Transaction Considerations

FareQuoteSuperBB <optimize>

FareQuoteSuperBB_# is a transaction that returns availability and price at the same time. It is used as the core transaction for many Internet Booking Engines in the B2C world. When you optimize the use of this transaction, response time is dramatically improved. Optimization can be achieved by indicating the specific KLRs you want to receive in the response.

See the **API Developer Notes:** ***Using Optimize for FareQuoteSuperBB*** document for instructions on how to specify which KLRs you want from the host.

Filters

It is recommended that you use filters to limit the amount of data that is returned by the GWS complex to the client application. Filters that are applied in the GWS tier also minimize the amount of data sent in the response and reduce the response time.

Shopping Options

If the responses to your shopping requests do not return the expected results or options, please contact your Galileo representative to discuss your concerns. There are various options that can change the results, depending on your business requirements.

Development Guidelines

Overview

This section outlines recommendations and best practices for developing a software application that uses Galileo Web Services (GWS). This section should be used in conjunction with the Implementation Best Practices section.

This guide assumes familiarity with Galileo Web Services, SOAP, XML, Java and/or .Net programming languages.

Reference Data

- Ensure results are cached when using the Travel Codes Translator eBL Web Service (encode/decode).
- Do not refresh more than once per day.
- Download the full Galileo Reference Data database as an alternative to caching.

Session Management

- Use sessionless requests whenever possible.
- Batch submit sessioned entries (if required) in a series followed by a call to end the session. Batching promotes the most efficient use of GTID sessions.

Transactions

- Use the latest versions of each transaction available at the start of development.
- Limit spawning transactions (e.g., submitting multiple transactions at the same time.
- Use caching where appropriate, and create cached data incrementally instead of all at one time.
- Validate travel codes and other local verifiable data from Galileo's Reference Data against a database on the customer's side.
- Do not use Sell transactions to determine availability before confirming a sell.
- XML Select and XML API Desktop customers that use cruise transactions must ensure that the first three alphanumeric characters (a–z, 0–9) of the user field in the identity record are randomly generated by their customer application. Unique user fields help to avoid requests being mismatched to responses in cruise vendor systems.

Web Services

- Use the *MultiSubmitXml* method whenever possible, but avoid bundling too many transactions into the MultiSubmit call because the response times are affected by the slowest responding transaction. The recommended limit to the number of requests is 15 - 20 included in a single call, and there are diminishing returns if the number is extremely high.
- Use Pre-Authentication for HTTP Basic Authentication Credentials to prevent a scenario where requests are being sent without credentials followed by a second request being sent with credentials. See the Galileo Web Services Help: *Getting Started > Accessing Galileo Web Services*.
- Use of GZIP functionality is recommended, and can increase performance. See the Galileo Web Services Help: *Getting Started > Accessing Galileo Web Services > gzip Compression*.
- Do not use the Connection: KeepAlive header.
- Disable Nagling or the Nagle Algorithm.

Shopping

- Do not follow Shopping requests with a FareQuoteFlightSpecific_# transaction.
- Use Shopping requests carefully because they are resource intensive. Indiscriminate use of these requests can cause system instability and poor performance of your client. Work with the API Technical Support Team on the most efficient use for your particular application.

Development/Troubleshooting

- Use your logging and monitoring functionality to facilitate quicker and more efficient troubleshooting responses from the API Technical Support Team.
- Ensure that the browser **Back** button does not resend all of the requests.