Guide to Accessing and Consuming SOAP-Based Sabre APIs

March 31, 2015 v1.42
Guide to Accessing and Consuming SOAP-Based Sabre APIs, March 31, 2015 v1.42

© 2003-2015 Sabre Inc. All rights reserved.

This documentation is the confidential and proprietary information of Sabre Inc. Any unauthorized use, reproduction, preparation of derivative works, performance, or display of this document, or software represented by this document, without the express written permission of Sabre Inc., is strictly prohibited.

Sabre, Sabre Travel Network, Sabre Airline Solutions and Sabre APIs are trademarks and/or service marks of an affiliate of Sabre. All other trademarks, service marks, and trade names are the property of their respective owners.

Disclaimer of Warranty and Limitation of Liability

This software and any compiled programs created using this software are furnished “as is” without warranty of any kind, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. No oral or written information or advice given by Sabre, its agents or employees shall create a warranty or in any way increase the scope of this warranty and you may not rely on any such information or advice.

Sabre does not warrant, guarantee, or make any representations regarding the use, or the results of the use, of this software, compiled programs created using this software, or written materials in terms of correctness, accuracy, reliability, currentness, or otherwise. The entire risk as to the results and performance of this software and any compiled applications created using this software is assumed by you. Neither Sabre nor anyone else who has been involved in the creation, production or delivery of this software shall be liable for any direct, indirect, consequential, or incidental damages (including damages for loss of business profits, business interruption, loss of business information, and the like) arising out of the use of or inability to use such product even if Sabre has been advised of the possibility of such damages.

Sabre Inc.
3150 Sabre Drive, Southlake, TX 76092
Tel: 682 605 1000
www.sabre.com
# Table of Contents

**Preface** ................................................................................................................................. 5
**SOAP-Based Sabre APIs Resources** ..................................................................................... 9
**SOAP-Based Sabre APIs Usage Requirements** ................................................................. 11
**External Resources** ............................................................................................................. 14
**Technical Support** ............................................................................................................... 16
**Chapter 1: Getting Started with SOAP-Based Sabre APIs** ................................................. 17
  **About SOAP-Based Sabre APIs** ......................................................................................... 18
  **Types of SOAP-Based Sabre APIs** .................................................................................. 18
  **Standards and Specifications** ........................................................................................... 22
  **Requesting Content** .......................................................................................................... 26
  **Security** ............................................................................................................................. 26
  **Network Connectivity** ...................................................................................................... 28
  **Error Types** ....................................................................................................................... 28
**Chapter 2: Sabre XML** ........................................................................................................... 29
  **WSDL Documents for Sabre XML** .................................................................................. 29
  **Sabre XML Schemas** ...................................................................................................... 36
  **Technologies for Working with SOAP-Based Sabre APIs** ............................................. 40
  **Versioning of Sabre XML Schema and WSDL Documents** ........................................... 43
**Chapter 3: SOAP Formats/Requirements** ........................................................................... 47
  **SOAP Message Overview** ............................................................................................... 47
  **SOAP Message Sequence and Format** ............................................................................ 51
**Chapter 4: Connection Management** ................................................................................ 72
  **SOAP-Based Sabre APIs Connections** ......................................................................... 72
  **Connection Management Messages** ............................................................................... 73
**Chapter 5: Business and Application Logic** ...................................................................... 96
  **Maintaining Session State** .............................................................................................. 96
  **SOAP-Based Sabre APIs Workflows** ............................................................................. 101
  **Minimizing Scans** .......................................................................................................... 103
**Chapter 6: SOAP-Based Sabre APIs Environments** ....................................................... 104
Preface

About This Guide

This document provides guidance in developing, accessing, and consuming SOAP-based Sabre® APIs.

Caution

When a client or solution books travel arrangements utilizing a URL that resolves to the back-end production system, the transactions are recorded in the live, production Sabre system, and real-time inventory is decremented. This applies to the following URL:

- https://webservices3.sabre.com/websvc

Please be sure to cancel any production environment bookings created for test purposes. If these bookings are not canceled, you and possibly your customers will be billed by suppliers or other vendors for all associated fees.

Scan charges may apply whenever a client application interacts with any of the environments established for Sabre APIs. Please consult your contract for a description of these charges. For tips on minimizing scans please refer to the section of this document titled, “Minimizing Scans.”

Advisories

To assist with capacity planning, advanced notification is required for the following activities.

- Performance and heavy load testing. These types of tests require notification a minimum of 5 business days before conducting the tests.

- Planned production dates and projected volumes. Notification must be a minimum of 120 business days prior moving to production.

- Changes to production volumes on an ongoing basis.

For complete information about the systems and environments available for client use, please refer to the section of this document titled, “SOAP-Based Sabre APIs Environments.”

Organization

- Chapter 1 introduces the SOAP-based Sabre APIs product, the standards and specifications the product is designed to meet, the versioning strategy associated with the SOAP-based Sabre APIs, as well as general comments related to connectivity and security.

- Chapter 2 discusses the Sabre SOAP-based XML specifications, versioning of the
WSDL and schema documents, as well as the versioning strategy that is applied to the SOAP-based Sabre APIs.

- **Chapter 3** describes the format and sending sequence of the SOAP messages used to connect to the SOAP-based Sabre APIs gateway in order to consume SOAP-based Sabre APIs. Complete requirements are also provided in Appendix B.

- **Chapter 4** presents several SOAP-based Sabre APIs connection strategies.

- **Chapter 5** includes topics related to business and application logic, managing content in a Sabre session, and requesting service versions.

- **Chapter 6** describes the environments that are available for consuming SOAP-based Sabre APIs.

- **Chapter 7** includes information related to troubleshooting general and system errors.

- **Appendix A** provides the SOAP envelope field size reference.

- **Appendix B** illustrates how to identify the URLs for WSDL documents and their associated schema documents.

- **Appendix C** illustrates several common travel workflows.

- The **Glossary** defines the various terms and acronyms utilized in this document.
## Document Conventions

### Terms

The use of terminology in this document is defined in the following table. For additional terms and information please refer to the glossary.

<table>
<thead>
<tr>
<th>This term…</th>
<th>Refers to…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>An application that uses or consumes a SOAP-based Sabre API.</td>
</tr>
<tr>
<td>Connection</td>
<td>An open channel to the SOAP-based Sabre APIs infrastructure.</td>
</tr>
<tr>
<td>Sabre Dev Studio</td>
<td>The repository where Sabre APIs-related information is stored and made available.</td>
</tr>
<tr>
<td>Domain</td>
<td>One of the security credentials used to establish a connection with SOAP-based Sabre APIs. When the documentation references a Domain, send the value you are given for Domain when you are set up to access SOAP-based Sabre APIs.</td>
</tr>
<tr>
<td>Internet Pseudo City Code (IPCC)</td>
<td>The unique code that identifies an organization. Application developers are given a value for Organization as part of the security credentials provided for accessing Sabre APIs. The code may or may not be an IPCC; it may be a PCC or other identifier.</td>
</tr>
<tr>
<td>SOAP-based Sabre APIs</td>
<td>All SOAP-based APIs provided by Sabre. These APIs include those that obtain their content from the Sabre global distribution system or Sabre open systems as well as APIs used to connect to the Sabre APIs infrastructure.</td>
</tr>
<tr>
<td>SOAP-based TPF Connector Sabre APIs</td>
<td>SOAP-based Sabre APIs that retrieve content from the Sabre global distribution system, also referred to as the Sabre host system or PSS (Passenger Service System).</td>
</tr>
<tr>
<td>SOAP-based Open systems Sabre APIs</td>
<td>SOAP-based Sabre APIs that obtain their content with direct connections to a variety of open systems of service providers within Sabre.</td>
</tr>
<tr>
<td>SOAP-based Session management Sabre APIs</td>
<td>SOAP-based Sabre APIs managed by the Sabre APIs gateway (also referred to as the USG) that connect to, verify, and disconnect from the Sabre APIs infrastructure.</td>
</tr>
<tr>
<td>RQ/RS</td>
<td>An abbreviation for request and response message pairs.</td>
</tr>
<tr>
<td>Sabre session</td>
<td>A terminal address (TA).</td>
</tr>
<tr>
<td><strong>Sabre system</strong></td>
<td>The Sabre GDS or “host” system, the system that stores travel inventory and itineraries. This system is the source of the travel-related content for SOAP-based TPF Connector Sabre APIs and other systems and applications.</td>
</tr>
<tr>
<td><strong>Sabre work area/AAA</strong></td>
<td>The Agent Assembly Area (AAA) or buffer in the Sabre host system where data is retained while a Sabre session is active.</td>
</tr>
<tr>
<td><strong>Sabre XML</strong></td>
<td>Sabre XML specifications are the WSDL and schema documents for Sabre APIs which have been modified from the OpenTravel specifications to accommodate proprietary data in the Sabre system and other Sabre applications.</td>
</tr>
<tr>
<td><strong>Security token</strong></td>
<td>The binary security token that is returned to the client after successfully connecting to the SOAP-based Sabre APIs gateway with the SessionCreateRQ Sabre API call. This security token is returned in the wsse:BinarySecurityToken element in the SessionCreateRS Sabre API call.</td>
</tr>
<tr>
<td><strong>Subscriber</strong></td>
<td>A travel organization that is a contracted customer of Sabre and Sabre APIs. Sabre subscribers include businesses or other entities such as travel agencies, online travel providers, travel suppliers (including airlines) and travel software development organizations who are involved with travel marketing and/or travel distribution. Sabre subscribers must have a valid Sabre access agreement to use the Sabre APIs.</td>
</tr>
</tbody>
</table>
SOAP-Based Sabre APIs Resources

The following resources are all available via the Sabre Dev Studio, located at [http://developer.sabre.com](http://developer.sabre.com). Accessing anything beyond basic Sabre API overviews, such as design documentation, WSDLs, schemas, etc. requires a user name and password, which is provided when clients sign up for the product.

**SOAP-Based Sabre APIs Developer Start-up Kit**

The information in the SOAP-based Sabre APIs Developer Start-up Kit helps developers get started quickly.

**SOAP-Based Sabre APIs Documentation**

Each Web service has an artifact on the Sabre Dev Studio.

Each artifact contains:

- A request design document
- A response design document
- A set of sample payloads
- Request and response schemas along with a WSDL

**Note:** Please consult these documents for the valid list of elements and attributes that are included in the service. The design documents list the valid elements and attributes applicable to each API/version, along a brief description and sample values. They also contain the equivalent Sabre formats for users familiar with native Sabre.

**Note:** The majority of SOAP-based Sabre APIs are based on OpenTravel specifications, and consequently, the associated schemas may contain elements and attributes defined by OpenTravel that the SOAP-based Sabre APIs do not leverage. Therefore, it is important to format request payloads to use only the elements and attributes that are present in the request and response design XML documentation located on the Sabre Dev Studio.

**Sample Clients**

The following sample clients are available on the Sabre Dev Studio. They assist with developing and consuming the SOAP-based session management and TPF Connector Sabre APIs. Each sample is contained in a ZIP file which describes the sample, has installation information for the platform of the sample, steps for running the sample, and any required JAR files.

The following samples are available:

- Sample Java test client for non-WSDL consumption. This client can execute any of the session management services and SOAP-based TPF Connector Sabre APIs, one at a time, in sequence. The purpose of this utility is to demonstrate how to connect to the SOAP-based Sabre APIs. This has the JAR files needed to run the sample and the licenses.
• Sample C# client code that consumes a SOAP-based TPF Connector Sabre API via WSDL using the Microsoft® .NET Framework.

• Sample Java client code that consumes a SOAP-based TPF Connector Sabre API via WSDL utilizing Apache Axis. This has three source code files that consume both the SOAP-based session management messages and a SOAP-based TPF Connector Sabre API. It also includes the necessary Axis JAR files needed to run this client.

Sabre System Formats, Keyword, and Functionality Assistance

Sabre® Travel Network™ customers can consult Sabre® FormatFinderSM. To access or download this reference system, visit https://agencyeservices.sabre.com, and choose FormatFinder from the Support menu. Please note that a Sabre system login ID is required to log in. The login ID for the Sabre system is the same as your SOAP-based Sabre APIs security credentials.

Sabre Airline Solutions® customers can consult FOCUS, the Automated Reference System. Access to this reference system is available via any Sabre terminal emulator by simply typing “FOCUS” on the command line.
SOAP-Based Sabre APIs Usage Requirements

Technical and System Access

There are several general requirements for being successful in developing with SOAP-based Sabre APIs:

Access to a Sabre Subject Matter Expert (SME)

While the SOAP-based Sabre APIs mask many of the complexities related to accessing content within the various Sabre systems it is important to consult with an SME to ensure that the client application being developed utilizes the most effective workflows and processes.

Connectivity

If the application being developed is behind a corporate firewall, the application developer needs the following proxy server-related information to be able to access the Internet:

- Proxy host name
- Proxy port
- Proxy user name
- Proxy password

JAVA-Based Developer Tool Kits

If developing with Java, the client needs to procure the necessary hardware, operating systems, files, and libraries that support Java development. Please note that Java Software Development Kit (J2SE) Version 1.3.1_04 is the minimal version required.

The following is also required:

- Java Secure Sockets Extension (JSSE) and related JAR files
- Java Web Services Developer Pack and related JAR files
- XML parser and related JAR file

For Java-based clients using SSL, Java Runtime Environment versions 1.3.1_10 and later, 1.4.1_06 and later, 1.4.2_03 and later

Apache Axis versions 1.1/1.1.1 can be used to consume Sabre APIs.

For the development kits, see the Sun Microsystems Web site at the following URL:

Microsoft .NET-based Developer Tool Kits

If developing with Microsoft .NET Framework the client needs to procure the necessary hardware, operating systems, files, and libraries that support .NET development.

The Microsoft Windows operating platform must be one of the following: Windows XP Professional or Home edition with Service Pack 1 or Windows 2000 with Service Pack 3 or greater.

Minimum requirements to generate proxy classes from the WSDL documents for SOAP-based Sabre APIs are listed below.

Microsoft .NET Framework 1.1 Requirements
Microsoft .NET Framework 1.1 Service Pack 1 – The WSDL documents require SP1.

(Optional) Visual Studio 2003

Visual Studio patch VS7.1 - KB823639-X86-Enu.exe

Service Pack 1 patch KB892202 – This patch fixes proxy client generation for Service Pack 1.

For more information about .NET Framework, see https://msdn.microsoft.com/en-us/vstudio/aa496123.

Microsoft .NET Framework 2.0 Requirements

It is possible to use the .NET Framework 2.0 with Visual Studio 2005 to generate proxy code. Special instructions for Sabre APIs are not necessary.

Session Resources

When a client is set up to access SOAP-based Sabre APIs, an Internet pseudo-city code (IPCC) is created. Each IPCC comes with an associated pool of session-related resources commonly referred to as a session pool, or a TAM pool. Please note that each IPCC comes with a finite number of session-related resources. For complete information about connection and session management, please refer to the section of this document titled, “Connection Management.” For information about the various available SOAP-based Sabre APIs environments, please refer to the section of this document titled, “SOAP-Based Sabre APIs Environments.”

Client Accounts

Each IPCC is allocated one administrative user account (sometimes, this user name is referred to as a user sign or Sabre sign). The administrative account can be used to change
the passwords of the non-administrative user accounts.

Each IPCC is allocated 1 non-administrative account for every 50 Sabre sessions in its session pool. (Sabre sessions are also referred to as TAs, and the session pool and TAM pool are the same.)

An account’s security credentials consist of the following:

- Username
- Password
- Organization
- Domain

**Note:** The passwords of user IDs for connecting to the SOAP-based Sabre APIs do not expire because the IDs are set up as robotic accounts. As a result, it is not necessary to change them every 90 days.

**Format Finder**

(Optional) Sabre Travel Network-based customers who want to use Format Finder require a login ID for the Sabre system. Format Finder is available via [https://agencyeservices.sabre.com](https://agencyeservices.sabre.com). Sabre APIs security credentials can be used to log into this system.
## External Resources

To learn more about XML, SOAP, WSDL, the W3C, Web services, OpenTravel, and other related technologies and organizations, please visit the Web sites below:

<table>
<thead>
<tr>
<th>To obtain this...</th>
<th>Visit this Web site...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information about the global consortium that develops e-business standards,</td>
<td><a href="http://www.oasis-open.org">http://www.oasis-open.org</a></td>
</tr>
<tr>
<td>including ebXML</td>
<td></td>
</tr>
<tr>
<td>Guidance, best practices, and resources for developing solutions with Web services</td>
<td><a href="http://www.ws-i.org">http://www.ws-i.org</a></td>
</tr>
<tr>
<td>This site also has samples of implementations of Web services created by various</td>
<td></td>
</tr>
<tr>
<td>vendors.</td>
<td></td>
</tr>
<tr>
<td>Information about XML and its components, such as XSLT, XLink, XML schema,</td>
<td><a href="http://www.w3c.org/XML/Schema">http://www.w3c.org/XML/Schema</a></td>
</tr>
<tr>
<td>including tutorials</td>
<td></td>
</tr>
<tr>
<td>OpenTravel specifications and information about creating and implementing industry-</td>
<td><a href="http://www.opentravel.org">http://www.opentravel.org</a></td>
</tr>
<tr>
<td>wide applications using these open e-business specifications</td>
<td></td>
</tr>
<tr>
<td>Information about vendors of Web services, industry news and articles, and</td>
<td><a href="http://www.webservices.org">http://www.webservices.org</a></td>
</tr>
<tr>
<td>developing with Web services</td>
<td></td>
</tr>
<tr>
<td>Information about working groups for architecture, protocols, descriptions,</td>
<td><a href="http://www.w3c.org">http://www.w3c.org</a></td>
</tr>
<tr>
<td>and choreography of Web services</td>
<td></td>
</tr>
<tr>
<td>Specifications, information about working groups, and industry updates,</td>
<td><a href="http://www.ebxml.org">http://www.ebxml.org</a></td>
</tr>
<tr>
<td>especially ebXML Message Service Specification V2.0</td>
<td><a href="http://www.ebxml.org/specs">http://www.ebxml.org/specs</a></td>
</tr>
<tr>
<td>WSDL</td>
<td><a href="http://www.w3c.org">http://www.w3c.org</a></td>
</tr>
<tr>
<td>Information about SOAP</td>
<td><a href="http://www.w3c.org">http://www.w3c.org</a></td>
</tr>
<tr>
<td>To obtain this...</td>
<td>Visit this Web site...</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>Information from The Apache Software Foundation about open source software, in</td>
<td><a href="http://www.apache.org">http://www.apache.org</a></td>
</tr>
<tr>
<td>particular, Apache Axis software development tools</td>
<td></td>
</tr>
<tr>
<td>Apache Web services Axis project site, where you can read about Apache Axis</td>
<td><a href="http://ws.apache.org/axis/index.html">http://ws.apache.org/axis/index.html</a></td>
</tr>
<tr>
<td>and select software tools</td>
<td></td>
</tr>
<tr>
<td>The Axis binary file needed to consume Web services with Apache Axis is</td>
<td></td>
</tr>
<tr>
<td>available on this page.</td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td>reference.html</td>
</tr>
<tr>
<td>Information about WSDL and Microsoft .NET Framework, as well as downloads of</td>
<td><a href="https://msdn.microsoft.com/en-us/vstudio/aa496123">https://msdn.microsoft.com/en-us/vstudio/aa496123</a></td>
</tr>
<tr>
<td>Microsoft .NET Framework tools, including the SDK, Visual Studio, and code</td>
<td></td>
</tr>
<tr>
<td>samples</td>
<td></td>
</tr>
</tbody>
</table>
Technical Support

There are several ways to obtain technical support. Please note that a pseudo-city code, or PCC, is required.

**Telephone:**

When reporting production or other critical/time sensitive issues, please contact the Global Support Center via the telephone:

- **USA:** 800-678-9460
- **Canada:** 682-605-5570
- **International:** 598-2-518-6020, or the customer’s regional Sabre Software help desk.

**Email:**

Email is monitored 24 x 7 with a response within 24 hours or less:

- [webservices.support@sabre.com](mailto:webservices.support@sabre.com)

Providing the support desk with the necessary files at the time of initial contact improves their ability to troubleshoot and provide resolution in a timely manner.

In order to better serve customers please note the following:

- Please include the Sabre pseudo-city code (PCC) where the issue is occurring.
- When reporting an issue with Sabre APIs, input and output payloads are required. Please attach the payloads as separate files, and name them clearly.
- To help ensure that the Sabre environment is free of viruses, our policy mandates that all messages received by Sabre from external sources follow special file name guidelines. File names must end in ".sabre.zip" or the zipped attachment will be removed by the e-mail server (for example, "docs.zip" would need to be renamed to "docs.sabre.zip").
- If correspondence is regarding a previously reported issue, please include the service incident ("SI") number in the subject line of the message.
Chapter 1: Getting Started with SOAP-Based Sabre APIs

Web Services

Web services are a specific type of APIs for application-to-application communication exposed via the Internet. The SOAP-based Sabre APIs are Web services.

A client application calls a Web service by sending an XML-based request message, and then the Web services infrastructure returns an XML-based response to the client. Because all communication is formatted in XML, a Web service is not tied to any particular operating system, programming language, or platform.

XML

XML is the basis for Web services and Web services technologies that exchange data. XML is used to define and describe the format of the data, its layout, and its logical structure through a schema. Software programs are usually written to transform this XML-formatted data to formats that other software applications and systems can understand, and then to transform the data back to XML.

SOAP

SOAP stands for simple object access protocol, and is a mechanism for transporting data from one network to another. In the SOAP-based Sabre APIs world a SOAP-based message is composed of the following parts:

- An envelope that contains communication information
- A header with attributes that describe the communication
- A body that contains the message or information about the message

WSDL

WSDL stands for Web Services Definition Language and uses a common format to describe and publish the formats, operations, and protocols of a Web service. WSDL elements describe data using one or more XML schemas. These schemas are passed to the Web
service. The description of the data tells the receiver how to process the data, and the binding to a protocol or transport instructs the sender how to send the data. Both parties must have access to the same XML schema.

**About SOAP-Based Sabre APIs**

The SOAP-based Sabre APIs are the preferred programmatic method for subscribers to access Sabre’s content and functionality. This content and functionality is exposed via a common access gateway infrastructure in the form of structured XML messages. This infrastructure manages sessions, security, logging, and routing of messages.

The SOAP-based Sabre APIs are delivered over HTTPS. SOAP is the message protocol that encodes Web services messages before they are sent.

Several SOAP-based URLs are available for client testing and production. For details about the environments and their corresponding URLs, please refer to the section of this document titled, “SOAP-Based Sabre APIs Environments.”

The SOAP-based Sabre APIs use document style information for the messages. The document style is used with both XML and WSDL.

The SOAP-based Sabre APIs utilize Sabre XML specifications, which are an extension of OpenTravel’s specifications, specifically tailored to meet the needs of Sabre and its clients.

The SOAP-based Sabre API artifacts, such as the WSDL and schema documents, and their URLs are available to subscribers via the Sabre Dev Studio.

**Types of SOAP-Based Sabre APIs**

When clients are developed to consume SOAP-based Sabre APIs, they are actually using multiple types of APIs: APIs that manage connections along with APIs that retrieve travel-related content. Of the travel-related SOAP-based Sabre APIs currently in place, four general types exist: session management APIs, TPF Connector APIs, open systems APIs, and orchestrated APIs.

**SOAP-Based Session Management Sabre APIs**

Messages that are used to establish and manage connections to the SOAP-based Sabre APIs infrastructure are referred to as session management Sabre APIs. These services are used to request new SOAP-based Sabre APIs sessions, validate existing SOAP-based Sabre APIs sessions, and close existing SOAP-based Sabre APIs sessions, ending the allocated Sabre session behind the scenes. For additional information please refer to the sections of this document titled, “SOAP Formats/Requirements,” and “Connection Management.”
SOAP-based TPF Connector Sabre APIs

SOAP-based TPF Connector Sabre APIs retrieve their content from the Sabre legacy host system. These services are a fast, reliable mechanism for accessing content in the Sabre legacy host system, handling the complexities of HSSP connection management, Sabre Data Source (SDS) conversion, as well as screen scraping where applicable, thereby eliminating the need for developers to deal with these aspects of the legacy Sabre host system.

SOAP-based TPF Connector Sabre APIs provide access to air, car, hotel, Passenger Name Records (PNR), and other miscellaneous functionality, such as queues within the legacy Sabre host system.

These APIs represent a powerful set of Sabre system commands, similar to building blocks. These APIs contain little to no business logic.

Being that these services utilize the legacy Sabre host system behind the scenes there are several important concepts to be aware of when using them.

The most important thing to be aware of is session management. Whenever a client application configured to access the SOAP-based TPF Connector Sabre APIs signs into the Sabre APIs infrastructure a host session is allocated from the pool of available sessions associated with the particular point of sale location, commonly referred to as a session pool or a TAM pool. Within the session/TAM pool there are a finite number of sessions available to each user so it is critical that the client application manages them efficiently by not exceeding the maximum number of sessions available at any given time, and by explicitly closing sessions that are not needed rather than letting them time out on their own. Please note that Sabre APIs sessions and Sabre host sessions remain active until they are explicitly closed or time out.

Another important item that client applications need to be aware of is the host buffer, commonly referred to as the Sabre work area, or AAA. The Sabre work area/AAA retains the content that is retrieved by the SOAP-based TPF Connector Sabre APIs. There are several instances where SOAP-based TPF Connector Sabre APIs depend on the presence of previously retrieved content in the Sabre work area/AAA. The most common illustration of this is modifying an existing Passenger Name Record (PNR). In order to modify an existing PNR the client application must first explicitly retrieve the record, which causes the legacy Sabre system to load the content into the Sabre work area/AAA. Once the content is loaded into the Sabre work area/AAA it can then be modified via subsequent Web service calls. Please note that the content remains in the Sabre work area/AAA while the session is active, or until the client saves and finalizes the content within the work area via the EndTransactionLLSRQ SOAP-based Sabre API call.

SOAP-Based Open Systems Sabre APIs

SOAP-based open systems Sabre APIs obtain their content from various back-end systems under the Sabre umbrella, outside of the legacy Sabre host system. These APIs provide
access to functionality that is not available in the Sabre legacy host system. An excellent example of a SOAP-based open systems Sabre API call is OTA_AirTaxRQ, which is used to retrieve tax-related information for a specified fare basis code/flight leg.

**SOAP-Based Orchestrated Sabre APIs**

SOAP-based orchestrated Sabre APIs combine multiple operations into a single API call. There are presently several SOAP-based orchestrated Sabre APIs available for consumption, PassengerDetailsRQ, EnhancedAirBookRQ.

The SOAP-based PassengerDetailsRQ Sabre API combines several, underlying SOAP-based TPF Connector Sabre API calls to create a basic Passenger Name Record (PNR).

The SOAP-based EnhancedAirBookRQ Sabre API combines several, underlying SOAP-based TPF Connector Sabre API calls for booking and pricing flight segments, and also includes the SOAP-based open systems OTA_AirTaxRQ Sabre API call for retrieving tax-related information for a specified fare basis code/flight leg.
Message Structure

The messages for SOAP-based Sabre APIs conform to the following specifications:

- The ebXML of the SOAP envelope conforms to SOAP with Attachments
- The content of the payload attachments conforms to Sabre XML

The structure of the messages is based on Internet standards such as HTTP, HTTPS, and the MIME mail extensions. HTTPS is the communications protocol.

The SOAP with Attachments protocol is a MIME multipart message with the following MIME parts:

- The header container – This is a SOAP envelope, which is an XML document.
- The payload container – This is the application payload, and it is formatted as Sabre XML.

The SOAP with Attachments protocol is used to format the messages for Java clients, and the payload is sent as an attachment.

Instead of sending the payload as an attachment, however, it can instead be included inside the SOAP wrapper. Java Axis clients include the payload inside the SOAP wrapper. If WSDL and Microsoft .NET Framework are used to format messages, the payload is included inside the SOAP wrapper.

For the format and sending sequence of the SOAP envelopes and payloads, please refer to the section of this document titled, “SOAP Formats/Requirements.” For specific tag requirements, please refer to the section of this document titled, “SOAP Envelope Field Size Quick Reference.”
Standards and Specifications

The standards and specifications that the SOAP-based Sabre APIs are based upon are listed below:

- **HTTP/1.1 [RFC2616]** – This is used for the transport protocol. Load balancing for the Sabre APIs infrastructure closely adheres to this protocol; hence HTTP messages headers that connect to Sabre APIs must conform to this.

- **MIME specifications [RFC2045], [RFC2046], and [RFC2387]** – These are used for the message headers and instructions.

- **SOAP, ebXML, and W3C XML standards** – These are used to define and describe the SOAP messages.

- **SOAP Messages with Attachments specification [SOAPAttach]** – This is used for the ebXML messages, which include the header and payload containers.

- **SOAP 1.1 [SOAP]** – This is used for the ebXML message packaging.

- The ebXML Message Service Specification Version 2.0 (http://www.ebxml.org/specs/ebMS2.pdf) – This is used for the header containers.

OpenTravel and the SOAP-based Sabre APIs have adopted ebXML messaging infrastructure for the packaging because ebXML specifies well-defined semantics for various messaging exchange patterns in the area of messaging over the Internet and Intranet. The Organization for the Advancement of Structured Information Standards (OASIS) drafts and maintains the ebXML standard.

- **WS-Security** – WS-Security standards have been partially adopted for some security elements.

- **W3C XML 1.0**

- **WSDL 1.1** – Sabre XML schemas have been simplified to comply with WSDL version 1.1.

- **OpenTravel specifications** (http://www.opentravel.org) – These are the basis for the travel-based request and response XML payloads. Sabre APIs are updated as needed to meet the newest OpenTravel specifications.

- **SOAP-based Sabre XML schema documents** – These are the schemas that validate the payloads in all SOAP-based Sabre APIs. The majority of them are based on OpenTravel message specifications.

- **WSDL documents for Sabre XML** – The WSDL documents are based on recommendations from the W3C, and conform to WS-I Basic Profile 1.0 Specification. When consuming SOAP-based Sabre APIs via WSDL, they are required to generate proxy code.
Sabre XML Specifications

As mentioned previously the majority of the SOAP-based Sabre APIs messages are based on OpenTravel specifications.

In the absence of approved specifications by OpenTravel, Sabre XML specifications are created utilizing the best practices concepts of OpenTravel. Therefore, some of the Sabre XML schemas have undergone slight modifications.

The types of modifications may include the following:

- Constraints on data types
- New elements

For information about working with WSDL, such as generating proxy classes, please refer to the section of this document titled, "Working with WSDL."

For more information related to managing connections and sessions, please refer to the section of this document titled, “Connection Management.” For the format and sending sequence of the SOAP envelopes and payloads, please refer to the section of this document titled, “SOAP Formats/Requirements.” For specific tag requirements, please refer to the section of this document titled, "SOAP Envelope Field Size Quick Reference."
SOAP-Based Sabre APIs Versioning Strategy

Individual, SOAP-based Sabre APIs are versioned to distinguish changes that are made to an API from one release to another. The first version of an API includes basic content, and upgraded versions include enhancements to existing content, new content, as well as corrections, i.e. bug fixes.

A new version of an API is created whenever any of the following occurs:

- Changes are made to an API that causes the request or response structure of the API to change, i.e. an enhancement.
- Changes are made to an API to correct an issue, i.e. a bug fix.

The SOAP-based Sabre APIs simultaneously supports up to five versions of a particular API. The APIs that are frequently upgraded have more versions available for consumption than those APIs which are seldom upgraded. Older API versions beyond the five supported versions are periodically removed from the system. Customers are provided with a minimum of 90 days advance notification prior to a particular API version being removed.

When APIs are upgraded, their corresponding WSDL and schema documents are also versioned in the same manner. Clients should check the release notes, which are available on the Sabre Dev Studio, http://developer.sabre.com, to ensure that they are aware of any changes/updates being made.

The first release of an API is assigned an initial version number. Whenever changes are made to an API, the first, second, or third numeral is incremented depending upon the nature of the change.

If the change causes a major request or response change, i.e. an API rewrite, or an entire platform upgrade, the first numeral, i.e. the major version level, is incremented. These types of changes are not deemed backwards compatible with previous API versions, i.e. API version 2.0.0 is not backwards compatible with API version 1.0.0. In these instances application developers will need to increment the major level digit, incorporate the functionality contained in the new major version into their application, and retest.

If the change causes a structural request or response change, i.e. an enhancement or a bug fix resulting in a schema change, the second numeral, the minor version level, is incremented, i.e. 2.1.0. These types of changes are not deemed backwards compatible with previous API versions, i.e. API version 2.1.0 is not backwards compatible with API version 2.0.0, or API version 2.0.1. In these instances application developers will need to increment the minor level digit, add the new functionality contained in the new minor version into their application, and retest.

If the change is to simply resolve a minor issue, i.e. a bug fix that doesn’t require any sort of schema change, the third numeral, the patch version level, is incremented, i.e. 2.0.1. These types of changes do not require schema updates so they are deemed backwards compatible between API versions sharing the same minor patch level, i.e. API version 2.0.1 is backwards compatible with API version 2.0.0. In these instances application developers
simply need to increment the patch level digit in their application and retest.

The client calls a particular version by specifying the desired version in the request payload at run-time.

**Guidelines for Upgrading Client Applications:**

<table>
<thead>
<tr>
<th>Type of API Upgrade</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform Upgrade, Major Enhancement / Re-Write</td>
<td>If clients want to take advantage of a platform upgrade, or a major enhancement/re-write, they must upgrade their application to consume the upgraded, major level version. Please note that these types of changes <strong>are not</strong> deemed backwards compatible between major versions, i.e. API version 2.0.0 is not backwards compatible with API version 1.0.0.</td>
</tr>
<tr>
<td>(The major level portion of the API version number is incremented, i.e. 2.0.0)</td>
<td></td>
</tr>
<tr>
<td>Minor Enhancement / Bug Fix With a Schema Change</td>
<td>If clients want to take advantage of a minor enhancement, i.e. new request or response elements/attributes, or a bug fix that resulted in a schema change, i.e. new request or response structures, or where the data type associated with an existing element or attribute is changed, they must upgrade their application to consume the upgraded, minor level version. Please note that these types of changes <strong>are not</strong> deemed backwards compatible between minor versions, i.e. API version 2.1.0 is not backwards compatible with API version 2.0.0.</td>
</tr>
<tr>
<td>(The minor level portion of the API version number is incremented, i.e. 2.1.0)</td>
<td></td>
</tr>
<tr>
<td>Minor Bug Fix (Without a Schema Change)</td>
<td>If clients want to take advantage of a bug fix that contained in a new API version that did not result in a schema change, they must upgrade their application to consume the upgraded, patch level version. This upgrade simply consists of incrementing the patch level digit, and retesting. These types of changes <strong>are</strong> deemed backwards compatible with the current minor version, i.e. API version 2.0.1 is backwards compatible with API version 2.0.0.</td>
</tr>
<tr>
<td>(The patch level portion of the API version number is incremented, i.e. 2.0.1)</td>
<td></td>
</tr>
<tr>
<td>Type of API Upgrade</td>
<td>Action</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>API Version Removal</td>
<td>The client must upgrade their application to consume a newer version.</td>
</tr>
</tbody>
</table>

**Requesting Content**

Content is requested by specifying the action code that corresponds to the SOAP-based Sabre API being called in the SOAP header along with the desired SOAP-based Sabre API version number in the body payload.

A unique action code identifies the request and response payloads for every one of the Sabre APIs. The name of a particular Sabre API and its action code, represented by the eb:Action element, are the same. The client provides the value for the action in the SOAP envelope. For more information about actions, please refer to the section of this document titled, “Sabre XML.” The action codes for each service are stated in the service overview provided for all SOAP-based Sabre APIs on the Sabre Dev Studio.

The payload requests of SOAP-based TPF Connector Sabre APIs must also include the desired version of the API being consumed. Each of the SOAP-based TPF Connector Sabre APIs has at least one version, and can have multiple supported versions at any given time. Payloads between versions can vary slightly, so it is important to consult the Sabre Dev Studio-based API documentation to identify differences between versions.

**Security**

The SOAP-based Sabre APIs leverage multiple layers of security for client applications. These layers include line security, authentication, authorization, and confidentiality.

**Line Security**

Line security is the layer that secures the data traveling on the line over the Internet between Sabre data centers and external systems. The SOAP-based Sabre APIs support point-to-point synchronous transport HTTPS using SSL with 128-bit encryption.

Clients that consume the SOAP-based Sabre APIs must implement line security with a secure sockets layer, and they must secure the payloads with HTTPS.

**Authentication**

Authentication is the layer that allows consuming applications access to the SOAP-based Sabre APIs. The URL for consuming SOAP-based Sabre APIs and security credentials provide authentication. Security credentials are the wsse:Username, wsse:Password,
Organization, and Domain elements present in the SOAP envelope in the request message of the SOAP-based SessionCreateRQ Sabre API call. Application developers receive the values for these elements when they are set up to use the SOAP-based Sabre APIs.

The SOAP-based Sabre APIs infrastructure authenticates the requestor using the security credentials in the request.

An example of the wsse:Security node that shows the security credentials is shown in Figure 1.

```
xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/12/utility">
  <wsse:UsernameToken>
    <wsse:Username>USERNAME</wsse:Username>
    <wsse:Password>PASSWORD</wsse:Password>
    <Organization>IPCC</Organization>
    <Domain>DEFAULT</Domain>
  </wsse:UsernameToken>
</wsse:Security>
```

Figure 1. Security Credentials in the wsse:Security node of SessionCreateRQ

**Authorization**

The authorization layer gives clients access to specific services or product packages.

When a client sends a request, the SOAP-based Sabre APIs infrastructure authorizes access to all services in the product packages to which an organization has subscribed.

**Confidentiality**

The confidentiality layer maintains the privacy of the data in a payload during its transmission. Sabre APIs leverage HTTPS with 128-bit SSL encryption.
Network Connectivity

Access to the SOAP-based Sabre APIs for external clients is available through the Internet. Consequently, resources used to develop and deploy production applications must have Internet access.

Error Types

Several types of errors are possible.

- SOAP-based Sabre APIs errors – These types of errors occur within the SOAP-based Sabre APIs infrastructure. The infrastructure detects and generates these errors, and returns them as SOAP faults, with or without ebXML headers.

- Business application errors – Business applications that are situated behind the Sabre APIs infrastructure generate errors which are caused by clients or the Sabre system. They are returned to clients in ErrorRS format.

- System errors generated by clients – Clients generate these errors which are external to the SOAP-based Sabre APIs infrastructure. They occur on the client side before reaching the SOAP-based Sabre APIs infrastructure.

When a response contains the <soap-env:fault> node, an HTTP status code of 500 is returned. If no SOAP fault exists, HTTP Status Code 200 is returned.
Chapter 2: Sabre XML

Chapter two describes the design of the WSDL and schema documents for Sabre XML. It also explains the numbering scheme and naming patterns of the WSDL and schema documents, as well as how they are versioned.

The Sabre XML specifications consist of the following:

- A unique WSDL document – This is used by WSDL software tools to build proxy classes. The tools reference the WSDL documents at run-time.
- A set of Sabre XML request and response XSD schema documents – They validate the XML payloads. If using WSDL tools, the WSDL document references them at run-time.
- An intermediate schema for every SOAP-Based Sabre API – This imports the request and response schemas.
- The content of the payloads.
- Session management messages for connecting to the SOAP-based Sabre APIs infrastructure.
- A set of common schemas shared by all SOAP-based TPF Connector Sabre APIs.

Each version of a SOAP-based Sabre API has its own set of Sabre XML documents.

The payload content is assigned a version number that is incremented whenever the content is enhanced or corrections are made to the code.

The WSDL and schema documents are available for each SOAP-based service housed on the Sabre Dev Studio can be found by signing into the site, searching for the name of the Web service, and then selecting the “Resources” link located at the bottom of each API overview page.

WSDL Documents for Sabre XML

Application developers can use the Sabre XML WSDL documents to develop and consume SOAP-based Sabre APIs by using development frameworks such as Microsoft .NET Framework or Apache Axis. WSDL documents simplify development of clients by generating
proxy classes for the client code. The proxy classes provide objects that let application developers access and update the underlying structure of the message, which is ebXML.

The WSDL documents are based on recommendations from the W3C. They conform to WS-I Basic Profile 1.0 Specification.

All SOAP-based TPF Connector Sabre APIs can be consumed with Microsoft .NET Framework and Apache Axis. The Sabre XML WSDL document format does not currently support the SOAP with Attachments model, so the SOAP envelope must include the message payload.

All data formats in the Sabre XML WSDL documents are defined as either character strings or integers. Most of the date formats are string types. The reason for this is various frameworks often define formats for the same data types differently, and these formats are incompatible among the different frameworks. By simplifying the data types, a single WSDL document for Sabre XML can accommodate multiple frameworks for WSDL.

**Format and Common Schemas**

The WSDL document includes a reference to an intermediate schema. This schema points to the request and response schemas for the Web service. The WSDL document also imports common Sabre XML schemas that provide instructions and data for consuming the Web service. These common schemas are used to build proxy classes.

The WSDL document structure has the standard definitions, import statements, and parent elements of `<message>`, `<portType>`, `<binding>`, and `<service>`. An example of a WSDL document that conforms to the WS-I recommendation is shown in Figure 4. The schemaLocation attribute of the xsd:import element in the types node has a fully qualified namespace. (See lines 2–10.)
Figure 4. WSDL Document That Conforms to WS-I Recommendations

(001)  <definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
webservices.sabre.com/websvcs xmlns:eb="http://www.ebxml.org/
namespaces/messageHeader" xmlns="http://schemas.xmlsoap.org/wsdl/
xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
targetNamespace="https://webservices3.sabre.com/websvc">
(002)   <types>
(003)     <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
(004)       <xsd:import namespace="http://webservices.sabre.com/sabreXML/
2003/07" schemaLocation="OTA_AirAvailLLS1.10.1RQRS.xsd"/>
(005)       <xsd:import namespace="http://www.ebxml.org/namespaces/
messageHeader" schemaLocation="msg-header-2_0.xsd"/>
(006)       <xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="xmldsig-core-schema.xsd"/>
(007)       <xsd:import namespace="http://www.w3.org/1999/xlink"
schemaLocation="xlink.xsd"/>
(008)       <xsd:import namespace="http://schemas.xmlsoap.org/soap/envelope/
" schemaLocation="envelope.xsd"/>
(009)       <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="xml.xsd"/>
(010)       <xsd:import namespace="http://schemas.xmlsoap.org/ws/2002/12/
secext" schemaLocation="wsse.xsd"/>
(011)   </xsd:schema>
(012) </types>
(013) <message name="OTA_AirAvailInput">
(014)   <part name="header" element="eb:MessageHeader"/>
(015)   <part name="header2" element="wsse:Security"/>
(016)   <part name="body" element="xsd1:OTA_AirAvailRQ"/>
(017) </message>
(018) <message name="OTA_AirAvailOutput">
(019)   <part name="header" element="eb:MessageHeader"/>
(020)   <part name="header2" element="wsse:Security"/>
(021)   <part name="body" element="xsd1:OTA_AirAvailRS"/>
(022) </message>
(023) <portType name="OTA_AirAvailPortType">
(024)   <operation name="OTA_AirAvailRQ">
(025)     <input message="tns:OTA_AirAvailInput"/>
(026)     <output message="tns:OTA_AirAvailOutput"/>
(027)   </operation>
(028) </portType>
(029) <binding name="OTA_AirAvailSoapBinding"
type="tns:OTA_AirAvailPortType">
(030)   <soap:binding style="document" transport="http://
schemas.xmlsoap.org/soap/http"/>
(031)   <operation name="OTA_AirAvailRQ">
(032)     <soap:operation soapAction="OTA"/>
(033)     <input>
(034)       <soap:header message="tns:OTA_AirAvailInput" part="header"
use="literal"/>
(035)       <soap:header message="tns:OTA_AirAvailInput" part="header2" use="literal"/>
(036)     </input>
(037)     <soap:body parts="body" use="literal"/>
(038)   </operation>
(039) </binding>
(040) </portType>
(041) </binding>
(042) </binding>
(043) </message>
(044) </portType>
(045) </binding>
(046) </portType>
(047) </definitions>
import Elements

The majority of the SOAP-based Sabre APIs conform to the OpenTravel specifications. Because of the complexity and nesting of the OpenTravel schemas, Sabre APIs import the schema files inside the WSDL document, which is not a common practice. The following example shows one of the import statements in a Sabre XML WSDL document.

<import namespace="http://webservices.sabre.com/sabreXML/2011/07" schemaLocation="OTA_AirAvailLLS1.10.1RQRS.xsd"/>

Of particular interest is the OTA_AirAvailLLS1.10.1RQRS.xsd intermediate schema pointed to by the schemaLocation attribute in the first import instruction.

This schema serves as a reference to two separate request and response schemas:

- OTA_AirAvailLLS1.10.1RQ.xsd and OTA_AirAvailLLS1.10.1RS.xsd

The implementation of an intermediate schema is needed for two reasons:

- The same namespace is defined within the request and response schemas. Within a WSDL document, repeatable namespace imports cannot be defined, and therefore, duplicate namespace imports are not permitted.

- The intermediate schema enables tools such as wsdl.exe to handle the complexity of WSDL documents so that the documents can be consumed by these tools.

Each WSDL document for Sabre APIs imports an intermediate schema specific to its corresponding SOAP-based Sabre API. The intermediate schema has references to namespace attributes and references to the request and response schemas specific to the Sabre API being called. The request and response schemas each define the payloads for the OTA_AirAvailLLSRQ service operation.

The WSDL file imports a set of common schemas that provide instructions for building
proxies and references to other schemas. This single set of common schemas is the same for all SOAP-based TPF Connector Sabre APIs.

Most of the common schemas are either imported into the WSDL documents for Sabre XML or referenced by other common schemas.

**message Elements**

The `<message>` element defines the data elements of an operation. Each message can be composed of one or more parts, with each part being equivalent to parameters of a function in a software program. An example from a SOAP-based Sabre APIs WSDL document is shown below.

```xml
<message name="OTA_AirAvailInput">
  <part name="header" element="eb:MessageHeader"/>
  <part name="header2" element="wsse:Security"/>
  <part name="body" element="xsd1:OTA_AirAvailRQ"/>
</message>

<message name="OTA_AirAvailOutput">
  <part name="header" element="eb:MessageHeader"/>
  <part name="header2" element="wsse:Security"/>
  <part name="body" element="xsd1:OTA_AirAvailRS"/>
</message>
```

SOAP-based Sabre APIs define two message elements in the WSDL documents, one for the request (lines 009–013) and one for the response (lines 014–018). Each message has multiple part elements that create the SOAP message header and body. While there are two major parts, there are actually three part parameters defined for each message because the header section is split into a MessageHeader and Security part. The body part creates the payload.

**portType Element**

The `<portType>` element defines the API, the operations that the API performs, and the messages that are involved. The portType element is the equivalent to a class in object-oriented programming. The operation is similar to a function call in structured programming. The operation and name combination calls an operation or function, and the function returns data. The request message is like the function and the response is like the data that the function returns. An example from a SOAP-based Sabre APIs WSDL document is shown below.

```xml
<portType name="OTA_AirAvailPortType">
  <operation name="OTA_AirAvailRQ">
    <input message="tns:OTA_AirAvailInput"/>
    <output message="tns:OTA_AirAvailOutput"/>
  </operation>
</portType>
```

The port defines the connection to a particular Sabre API.
In general, WSDL documents can define several types of operations, such as one-way, notification, or request-response.

The WSDL documents for SOAP-based Sabre APIs define the request-response type of operation. This is because a client sends a request and receives a response when consuming the SOAP-based Sabre API.

WSDL documents for Sabre APIs define two messages per operation. Lines 21–22 represent the input or request message, and the output or response message.

**binding Elements**

The `<binding>` element defines the data format and protocol for each port.

The `<operation>` element defines each operation that the port exposes. For each operation, the corresponding SOAP action is defined, and the method of encoding for the input and output must be specified. An example from a SOAP-based Sabre APIs WSDL document is shown below.

```
(025)  <binding name="OTA_AirAvailSoapBinding" type="tns:OTA_AirAvailPortType">
(026)   <soap:binding style="document" transport="http://schemas.xmlsoap.org/
          soap/http/>
(027)   <operation name="OTA_AirAvailRQ">
(028)     <soap:operation soapAction="OTA"/>
(029)     <input>
(030)       <soap:header message="tns:OTA_AirAvailInput" part="header"
          use="literal"/>
(031)       <soap:header message="tns:OTA_AirAvailInput" part="header2" use="literal"/>
(032)     </input>
(033)     <output>
(034)       <soap:header message="tns:OTA_AirAvailOutput" part="header" use="literal"/>
(035)       <soap:header message="tns:OTA_AirAvailOutput" part="header2" use="literal"/>
(036)     </output>
(037)   </operation>
(038)  </binding>
```

The binding element has two attributes, name and type. The name attribute defines the name of the binding, and the type attribute points to the port for the binding (line 25). In the example, the port is OTA_AirAvailPortType.

The soap:binding element has two attributes—style and transport (line 26). In general, the style attribute can be rpc or document. The Sabre APIs use the document style. The transport attribute defines the SOAP protocol to use. In the case of Sabre APIs, this is HTTP because the transport protocol is SOAP/HTTP.

The `<operation>` element (line 27) defines each operation that the port exposes. For each operation, the corresponding SOAP action has to be defined, and the method of encoding
for the input and output must be specified.

The OTA_AirAvailRQ operation has an input message called OTA_AirAvailInput (line 30), and an output message, OTA_AirAvailOutput (line 35).

The message elements define parts of each message and their associated data types. The parts are soap:header and soap:body (lines 30–32 and 35–37).

In terms of object-oriented programming, OTA_AirAvailPortType is a class, and OTA_AirAvailRQ is a function with the parameters OTA_AirAvailInput and OTA_AirAvailOutput.

**service Element**

In a WSDL document, the <service> element, subelements, and attributes define the Sabre API, the port, and the endpoint. An example from one of the SOAP-based Sabre APIs WSDL documents is shown below.

```
(041) <service name="OTA_AirAvailService">
(042)   <port name="OTA_AirAvailPortType" binding="tns:OTA_AirAvailSoapBinding">
(043)     <soap:address location="https://webservices3.sabre.com/websvc"/>
(044)   </port>
(045) </service>
```

The <service> element in a Sabre XML WSDL document defines a single Sabre API. The name attribute (line 041) is the name of the Sabre API, i.e. the OTA_AirAvailService.

The specific SOAP-based Sabre API is defined with the port element and name attribute (line 042). The combination of soap:address and location identify the endpoint into Sabre APIs. All WSDL documents for Sabre APIs include the production endpoint/URL (line 043).

**Common Schemas for All Travel-Based Sabre APIs**

Some of the common schemas imported in the WSDL documents provide namespace declarations for the ebXML SOAP extensions for the envelope, header, and body elements. Other import elements reference common schemas that are specifications. One set of common schemas has been created for use by all Sabre APIs. Brief descriptions of these common schemas follow:

- `msg-header-2_0.xsd` – Used for the message header
- `xmldsig-core-schema.xsd` – Used for XML signatures and encrypting data
- `xlink.xsd` – Used for NMTOKEN
- `envelope.xsd` – Used for namespaces. This file references `env.xsd`, which is used for the SOAP envelope.
- `xml.xsd` – Defines attributes and an attribute group
• wsse.xsd – Used for the WSSE security specification
• datatypes.xsd – The data types used in the XML schema documents
• XMLSchema.dtd – The data type definitions, of definitions of data types, used in the XML schema documents

**Sabre XML Schemas**

To provide more content and to accommodate the use of proprietary data in Sabre systems and applications, the Sabre XML request and response schema documents have the following types of modifications that are not present in the OpenTravel specifications:

• The use of TPA_Extensions
  The term “extension” refers to an element or attribute that is added to the OpenTravel specifications. Extensions let organizations use proprietary content that is not present in the OpenTravel specifications so that they can exchange content among their trading partners.
  Many Sabre XML schemas incorporate TPA_Extensions.

• Constraints on data types
  Many Sabre XML schemas have specific requirements for the values that are provided with some of the elements and attributes in the payloads. These requirements are referred to as constraints. Constraints include data types, restrictions on valid values that the elements and attributes can send, and whether an element or attribute is required or optional.

• New elements
  Elements have been added to some requests to make the data conform to the proprietary data format in the Sabre system.

The SOAP-based Sabre APIs use published XML schemas that specify the syntax of the messages. Document type definitions are not used. These Sabre XML schemas include the following information about the elements and attributes in the XML requests: data type, length, valid values, sending sequence, and minimum and maximum occurrences.

**Note:** The majority of the schemas for the SOAP-based Sabre APIs are based on OpenTravel specifications. Consequently, they contain many elements and attributes that Sabre APIs do not use.

The XML design documentation for each of the SOAP-based Sabre APIs lists the elements and attributes that are valid for the particular XML request and response payloads. While designing client applications, it is important to consult the design XML documents for the valid lists of data located on the Sabre Dev Studio.
Request and Response Schemas

Each of the SOAP-based Sabre APIs normally corresponds to one unique pair of request and response Sabre XML schemas. Many of the Sabre XML schemas are based on a pair of OpenTravel specifications for a request and response message. For example, the pair of schemas that corresponds to the OTA_AirAvailLLSRQ service are OTA_AirAvailLLS1.10.1RQ.xsd and OTA_AirAvailLLS1.10.1RS.xsd.

Some exceptions to the request and response pair exist. Several of the SOAP-based TPF Connector Sabre APIs with hotel content share an additional common XSD schema. The enhanced versions of the WSDL and schema documents for the SOAP-based OTA_HotelResLLSRQ, HotelPropertyDescriptionLLSRQ, HotelRateDescriptionLLSRQ, and OTA_HotelAvailLLSRQ Sabre APIs all reference the HotelCommonTypes.xsd schema. The HotelCommonTypes.xsd schema combines the data types for guarantee information that these hotel-based Sabre APIs share to ensure commonality between the calls.

Application developers can use these schemas to validate their XML payloads for non-WSDL consumption. If application developers are consuming SOAP-based Sabre APIs via WSDL, they can use them to review the structure of the data in the payloads.

Basis for Payload Content

The content of the payloads for SOAP-based Sabre APIs is based on the OpenTravel messages. The OpenTravel request message is the basis for the request payloads and the responses are based on the OpenTravel responses.

Because the content of the payloads varies for each Sabre API, action codes are used to distinguish the payloads. Like OpenTravel, each request and response message for each Sabre API has a unique action code. Please note that for the SOAP-based TPF Connector Sabre APIs, the action code is also the same as the name of the service, for example, “OTA_AirAvailLLSRQ” is the action code for the Sabre API named OTA_AirAvailLLSRQ.

How to pass the action codes in the SOAP envelopes is shown in the section of this document titled, “eb:Action.”

Data Types, Descriptions, and Constraints

The Sabre XML schema documents provide some or all of the following information about the data to include in the payloads.

- Data types
  The data types are defined in the schemas so that they can be validated. Most of the data types are text. The schemas also define constraints on the values that are sent in the payloads, such as character type (alphabetic, numeric, alphanumeric, or other) and length.
- If applicable, valid values for elements and attributes
In some cases, elements and attributes require valid values to process the requests successfully. If valid values are not provided in the requests, the services fail.

In other cases, providing valid values is preferred, but not required, to process the services successfully. If the provided values are not valid, the service substitutes default values during processing, and the content in the responses is associated with the substituted values.

- Whether the data is required or optional

While the schemas often provide this information, it is recommended that application developers refer to the appropriate Sabre APIs design documents for required data elements.

- Sequences for sending the data

- Minimum and maximum occurrences
  This is the quantity of times the data can be requested in a payload.

- Descriptions of elements and attributes

**Tips for Finding and Formatting Data**

A starting place for identifying data types is to open the request schema document that is associated with a specific service, and then look in the design XML document for the list of data elements in the request. Next, search the schema for those elements and attributes, and note the data types, valid values, sending sequences, and minimum and maximum occurrences.

1. For the complete list of required and optional data to include in the request payloads, always use the design XML documents. In many cases, the payload examples in the service documents show the minimal data in the requests, while the design documents list all possible data.

2. Occasions occur when an element is optional but its attribute is required. When this happens, the attribute is required only if the element is included in the payload.

3. Look at the annotations for information. They provide descriptions, and sometimes they provide restrictions on valid values.

4. The response schemas describe the data elements in the responses, and the design XML documents list all possible data elements that can be returned, depending on the elements in the request.

5. The design XML documents have annotations about incompatible combinations of elements and attributes in a request. This means they cannot be combined in a single request payload.

6. If application developers require more information or need help finding an
element, they can try appending the suffix Type to the name of the element. For example, if they cannot find CodeRef, they can try searching for “CodeRefType.” There may be information about the element in the annotation.

7. Use an XML editing tool that provides various views, such as the views described below:
   - A text view that displays all text in hierarchical fashion
   - A schema or design view that expands elements to display associated children, types, and other information
   - A grid view that provides a graphical representation of the elements

8. Use the service description, design XML, schema documents, and developer notes for complete information about the Sabre APIs, including comparable Sabre system formats for the elements and attributes. The developer notes on the Sabre Dev Studio describe the required service-specific values for the SOAP envelopes. Sample request and response payloads are also available.
Technologies for Working with SOAP-Based Sabre APIs

Application developers can create clients and consume Sabre APIs in the language of their choice, working directly with the XML or via WSDL.

Instructions on how to set up the development environment for .NET Framework 1.1 are located in the SOAP-based Sabre APIs Microsoft .NET Framework Installation Tips which is contained in the Sabre APIs Developer Start-Up Kit on the Sabre Dev Studio. Some information about set-up with Apache Axis and XML-Java appears in the readme files for these respective sample clients, which are also available on the Sabre Dev Studio.

**Working Directly with XML**

Application developers can develop clients to consume SOAP-based Sabre APIs in the language of their choice, such as Java. To consume SOAP-based Sabre APIs and use the sample Java code, the minimal required version of the Java Software Development Kit (J2SE) is Version 1.3.1_04.

When consuming SOAP-based Sabre APIs with clients such as Java and XML, application developers are free to use the parser of their choice. Please note that the parser must be namespace aware.

Responses are returned as XML documents. The client code parses the XML or uses APIs to map the XML to objects, and queries the objects for the data.

**Working with WSDL**

Application developers can use tools that are designed to consume SOAP-based Sabre APIs at run-time, such as Microsoft .NET Framework or Apache Axis. With tools like these, they can create clients using a variety of programming languages, and a combination of languages and operating platforms can be used to generate proxy code from WSDL documents. With WSDL, clients can consume APIs written in any language that is available with the development framework.

Working via WSDL is easier than working directly with XML because WSDL creates a proxy class in the language of your choice. The proxy class has instructions for mapping the XML response to objects. The client code has to parse the XML, but it does not need to map the XML to objects.

Another benefit of using tools to consume SOAP-based Sabre APIs at run-time is that errors on the client side are minimized.

Validation of the XML messages is done remotely at the URL where the WSDL documents for the Sabre API reside.
The WSDL documents define interfaces to SOAP-based Sabre APIs as a collection of operations with an endpoint. A WSDL document is a specific type of XML schema that defines a language for expressing API interfaces that XML software understands and uses. WSDL was designed to use SOAP as the message transport.

The WSDL documents for SOAP-based Sabre APIs are simplified Sabre XML schemas. These WSDL documents support the document-oriented style of SOAP binding.

The following tools are recommended for consuming SOAP-based Sabre APIs via WSDL:

- Apache Axis for Java clients
- Microsoft .NET Framework and Microsoft Visual Studio when developing C# and Visual Basic clients

**Generating Proxy Classes and Consuming Services**

The WSDL and schema documents are used at build time to create proxy classes. The WSDL document creates proxy classes that are then used to build client code, and the schemas determine the format of the messages. When consuming SOAP-based Sabre APIs via WSDL, in other words at run-time, the application developer must direct their development tool to the location of the WSDL document. The WSDL document, in turn, references the appropriate XML schemas.

Whenever a WSDL or schema document undergoes any modification, and the application developer wants their client to use the modifications, they must regenerate the proxy classes at build time to use the changes.

**Run-Time References to WSDL Documents**

The WSDL and schema documents are used at build time to generate proxy classes. Whenever a schema or WSDL document undergoes a modification, whether major or minor, and an application developer wants their client to use the modifications, they must regenerate the proxy classes.

When consuming SOAP-based Sabre APIs via WSDL, in other words at run-time, the application developer must direct their development tool to the location of the WSDL document. The URL for the WSDL documents on the Sabre Dev Studio can be found by signing into the site, searching for the name of the Web service, and then selecting the “Resources” link located at the bottom of each API overview page.

**Apache Axis**

Apache Axis is an implementation of SOAP proposed by the Apache Software Foundation. With Axis Framework, application developers can develop clients with Java, and clients consume SOAP-based Sabre APIs via WSDL. The Axis binary file, JAR files, and reference guide are required and available on the Apache Software Foundation’s Web site at [http://ws.apache.org/axis/](http://ws.apache.org/axis/).
Microsoft .NET Framework

With the .NET Framework, application developers can develop in any language available with the framework, such as C++, C#, or Visual Basic. Application developers can use a combination of languages and operating platforms to generate proxy code via WSDLs using .NET tools.

The SDK has the programs and files needed to develop clients, including the wsdl.exe and intermediate disassembler programs, and .NET Framework includes all the system tools and files needed for run-time operation.

For more information about WSDL and .NET, visit the following Microsoft URL: https://msdn.microsoft.com/en-us/vstudio/aa496123.

Validating XML Payloads

It is recommended that application developers validate their XML documents during development and testing, but real-time validation in production is not required. Doing so may affect the performance of clients. If the client application is consuming APIs without WSDL, developers can validate their XML payload documents locally. To do local validation, download the latest schemas that correspond to the Sabre APIs being leveraged.

If application developers need to do run-time validation of XML documents, they can point to the URL that corresponds to the desired version of the schema. The URLs for the WSDL and schema documents are available on the Sabre Dev Studio.
Versioning of Sabre XML Schema and WSDL Documents

This versioning strategy applies to the SOAP-based TPF Connector Sabre APIs, which obtain content from the legacy Sabre system.

An artifact, in the context of Web services, is anything that assists in the discovery and use of a service. Some examples of Sabre APIs artifacts include, but are not limited to, the WSDL and schema documents, design XML documents, sample payloads, and action codes. Each of these artifacts exists as a separate entity.

Artifacts are not generally shared among Web services, although some minor exceptions exist, such as the common schemas that are used across multiple Sabre APIs.

Metadata is data about a SOAP-based Sabre API. Examples of metadata for a SOAP-based Sabre API include the name of the API, the SOAP action associated with the particular service, etc…

It is important to be able to identify and obtain the artifacts for a specific version of specific SOAP-based Sabre APIs so that application developers can discover, consume, and troubleshoot them. To help, discussions follow on versioning and file naming standards for schema and WSDL documents, numbering system for Sabre APIs and documents, naming conventions for documents, and naming standards for the corresponding URLs.

Versioning and File Naming Standards

The SOAP-based Sabre APIs simultaneously support up to five versions of a particular API and its corresponding WSDL and schema documents. Therefore, multiple versions of a specific API and its corresponding set of WSDL and schema documents coexist for many of the SOAP-based Sabre APIs. The version is incremented whenever enhancements or corrections are made to the request or response messages.

The most recent versions of the schema and WSDL documents for a service that are released in production are maintained along with the most recent versions of the corresponding SOAP-based Sabre API. When corrections are necessary, a new version of the SOAP-based Sabre API and its corresponding WSDL and schema documents is created, and the changes are made to the artifact requiring the correction, either the WSDL or schema document, or the API itself.

Version Numbering System for Sabre APIs and Documents

The numbering system affects SOAP-based Sabre API versions, the file names of WSDL and schema documents, and the URLs where the WSDL and schema documents reside. With the exception of the initial version, the WSDL and schema document versions match the version numbers of their corresponding Sabre APIs. The request and response design XML
documentation also follows the same model.

The three-part version number is applied to the file names of the documents as well as the SOAP-based Sabre APIs themselves.

The format of the version number is 1.0.1, where:

\[ 1.0.1 = \text{is the version number, and the second digit is incremented} \]
Naming Conventions for WSDL and Schema Documents

Each of the Sabre APIs has a set of documents and naming conventions that are aligned with the numbering system. Most of the file names in the document sets contain a root that is the “base” action code of the API request and response, with RQ or RS omitted from the base.

For example, the action code of the request for the SOAP-based IgnoreTransactionLLSRQ Sabre API call is IgnoreTransactionLLSRQ. For the response, the action code is IgnoreTransactionLLSRS. Therefore, the base action code, with the RQ or RS omitted, is IgnoreTransactionLLS.

The examples in the table illustrate the file naming patterns for the SOAP-based IgnoreTransactionLLSRQ Sabre API.

The file names of the initial versions of WSDL and schema documents are without version numbers. The file names of additional, new versions include the three-part version number.

<table>
<thead>
<tr>
<th>Document Type</th>
<th>File Naming Convention</th>
<th>Examples of File Names for Initial Versions</th>
<th>Examples of File Names for Version 2.0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSDL document</td>
<td>Base action code + version + RQ + wsdl file extension</td>
<td>IgnoreTransactionLLSRQ.wsdl</td>
<td>IgnoreTransactionLLS2.0.0RQ.wsdl</td>
</tr>
<tr>
<td>Common schema</td>
<td>Base action code + version + RQ + RS + xsd file extension</td>
<td>IgnoreTransactionLLSRQRS.xsd</td>
<td>IgnoreTransactionLLS2.0.0RQRS.xsd</td>
</tr>
<tr>
<td>Request schema</td>
<td>Base action code + version + RQ + xsd file extension</td>
<td>IgnoreTransactionLLSRQ.xsd</td>
<td>IgnoreTransactionLLS2.0.0RQ.xsd</td>
</tr>
<tr>
<td>Response schema</td>
<td>Base action code + version + RS + xsd file extension</td>
<td>IgnoreTransactionLLSRS.xsd</td>
<td>IgnoreTransactionLLS2.0.0RS.xsd</td>
</tr>
</tbody>
</table>

Using File Names to Identify WSDL and Schema Document Versions

Version identifiers are present in the WSDL and schema documents.

WSDL Documents

The following line is present in all versions of WSDL documents. If the file name does not have a version number, it indicates the WSDL document is the initial version.

```xml
<import namespace="http://webservices.sabre.com/sabreXML/2011/10" schemaLocation="IgnoreTransactionLLS2.0.0RQRS"/>
```
Intermediate Schemas

The combination of include/schemaLocation point to the request and response schemas. Schemas with a version number in their file names have been upgraded beyond the initial version, which omits a version number in the file.

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <include schemaLocation="IgnoreTransactionLLS2.0.0RQ"/>
  <include schemaLocation="IgnoreTransactionLLS2.0.0RS"/>
</schema>
```

Obtaining WSDL, Schema, Design, and Other Service Documents

All documents for all SOAP-based Sabre APIs are available Sabre Dev Studio, either by viewing them in a browser window or by downloading them. To obtain the documents, application developers need a user name and password, which is provided when their user accounts are set up.

The WSDL and schema documents are also available by directly accessing them via a URL. To access them directly, application developers must become familiar with the URL and file naming patterns of the documents. A description of the URL naming standards and specific instructions on how to identify the WSDL and schema documents appear in the section of this document titled, “Identifying Documents for SOAP-Based Sabre APIs.”
Chapter 3: SOAP Formats/Requirements

Chapter three illustrates the sequence and format of the SOAP-based Sabre API calls used to successfully connect to and consume the SOAP-based Sabre APIs.

SOAP Message Overview

The SOAP Message with Attachments specification has two MIME parts: the header container, which is the SOAP envelope, and the payload container, which is where the payload is placed. For simplicity, this document refers to these two MIME parts as the SOAP envelope and payload.

SOAP Envelopes

The ebXML-based MessageHeader inside the SOAP envelope contains routing information for the message as well as other important information, such as the …/ConversationId, the …/Service, and the …/Action.

Payloads

The payload is the business or application content of the message. It corresponds to the request for the service being called. The payload is based on approved Sabre XML vocabularies for clients that consume SOAP-based Sabre APIs.

Sabre XML messages support one payload per envelope. Depending upon how the client consumes SOAP-based Sabre APIs, the payload is either sent as an attachment or included inline inside of the SOAP envelope.
For Java clients, the payload is a MIME part following the SOAP with Attachments Specification. While it is preferable to send the message as an attachment, it is also possible to format the payload inside the SOAP envelope when using Java.

For clients that consume Sabre APIs via WSDL, including clients that are developed with Microsoft .NET Framework or Apache Axis, the messages must conform to the WSDL standard by including the payload inside the SOAP envelope.

The Sabre XML schemas define the required formats for the content in the message payloads, including the extended elements and attributes that are defined for use with the Sabre system and other Sabre applications. (These are child elements of the TPA_Extensions nodes.)

**Note:** Each SOAP-based Sabre API has unique individual, API-specific values for the SOAP envelopes and payloads. For the valid list of elements and attributes associated with a particular SOAP-based Sabre API, consult the design documents located on the Sabre Dev Studio. The schemas associated with each individual SOAP-based Sabre API provide the formats and constraints for the data elements themselves.

**XML Request and Response Message Pairs**

Each SOAP-based Sabre API consists of an XML-based request and an XML-based response. The request is the message that a client sends to the appropriate Sabre system or application for processing, and the response is the message that the SOAP-based Sabre API returns to the client for consumption.

The basic types of functionality available in these messages are as follows:

- **Read functionality.** These types of messages find information and retrieve it for display. APIs with read functionality are for viewing data, such as fare displays, vehicle rates and rules, air schedules, and availability.

- **Write functionality.** These messages create or modify records in the Sabre system, such as PNRs. APIs which are based on write functionality create or add to something in the Sabre system.
Message Structure

The messages for Sabre APIs conform to the following specifications:

- The ebXML of the SOAP envelope conforms to SOAP with Attachments.
- The content of the payload attachments conforms to Sabre XML.

The structure of the messages is based on Internet standards such as HTTP, HTTPS, and the MIME mail extensions. HTTPS is the communications protocol.

The SOAP with Attachments protocol is used to format the messages. The preferred format has the payload as an attachment, as shown in Figure 2. HTTPS is the transport protocol.

Figure 2. Structure of an ebXML Message with a Payload Attachment
The SOAP Messages with Attachments specification is a multipart message with two MIME parts: the header container and payload container.

The SOAP message consists of the following elements:

- **SOAP header** – This is the mechanism to add features to the SOAP message, including header elements that are specific to ebXML.

- **SOAP body** – This is the container for the control data of the message service handler and information about the payload parts of the message. If the payload is sent as an attachment, the ebXML `<eb:Manifest>` element references the attached payload in the SOAP body.

- **Header container**
  The header container has a SOAP envelope, which is an XML document.

- **Payload container**
  The payload container is the application payload. It is formatted as Sabre XML. The content is either the business logic or data without business logic.

Instead of sending the payload as an attachment, it can be included inside the SOAP wrapper, replacing `eb:Manifest` inside the SOAP envelope. This is shown in Figure 3. If WSDL is used to format the messages, the payload is included inside the SOAP wrapper.

**Figure 3. Structure of an ebXML Message with the Payload Inside the SOAP Body**
SOAP Message Sequence and Format

When clients consume SOAP-based Sabre APIs, they use two types of messages: session management messages and travel content-related messages. This topic reviews the message formats for use in a conversational style connection. The messages are presented in their required sending sequence. For detailed requirements about formatting the data elements in the SOAP envelope, please refer to the section of this document titled, “SOAP Envelope Field Size Quick Reference.”

The names of the message pairs for each Sabre API end with RQ and RS, where “RQ” represents the request, and “RS” represents the response.

Some nodes and requirements in the SOAP messages are the same for all SOAP-based Sabre APIs, while other requirements are specific to the specific SOAP-based API itself. The SOAP-based session management APIs also have some unique nodes and formats in the SOAP envelopes. For all API-specific values, please consult the descriptions/developer notes available on the Sabre Dev Studio.

The SOAP-based Sabre APIs are built to conform to several standards/specifications, including ebXML and WS-Security. Therefore, the SOAP envelopes contain namespaces, elements, and attributes that these standards and specifications reference and require. For the additional information related to the standards and specifications please refer to the section of this document titled, “Standards and Specifications.”

Some fields have maximum lengths. Any data values exceeding the maximum number of characters results in an error which is returned to the client, preventing the client from creating a connection. For information related to the maximum field size of these data elements please refer to the section of this document titled, “SOAP Envelope Field Size Quick Reference.”

SessionCreateRQ Request Message

Consumers of all types of SOAP-based Sabre APIs use the same SOAP-based SessionCreateRQ Sabre API call to open connections to the SOAP-based Sabre APIs infrastructure.
Example 1. SOAP-Based SessionCreateRQ Envelope

(001) <?xml version='1.0' encoding='UTF-8'?>
(003) xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
(004) xmlns:xlink="http://www.w3.org/1999/xlink"
(005) xmlns:xsd="http://www.w3.org/1999/XMLSchema">
(006) <SOAP-ENV:Header>
(007) <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="2.0">
(008) <eb:ConversationId>ABC123@clientURL.com</eb:ConversationId>
(009) <eb:From>
(010) <eb:PartyId type="urn:x12.org:IO5:01">clientURL</eb:PartyId>
(011) </eb:From>
(012) <eb:To>
(013) <eb:PartyId type="urn:x12.org:IO5:01">webservices.sabre.com</eb:PartyId>
(014) </eb:To>
(015) <eb:CPAlId>yourIPCC</eb:CPAlId>
(016) <eb:Service eb:type="sabreXML">Session</eb:Service>
(017) <eb:Action>SessionCreateRQ</eb:Action>
(018) <eb:MessageData>
(019) <eb:MessageId>mid:20031209-133003-2333</eb:MessageId>
(020) <eb:Timestamp>2003-12-09T11:15:12Z</eb:Timestamp>
(021) <eb:TimeToLive>2003-12-09T11:15:12Z</eb:TimeToLive>
(022) </eb:MessageData>
(023) </eb:MessageHeader>
(025) xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/12/utility">
(026) <wsse:UsernameToken>
(027) <wsse:Username>USERNAME</wsse:Username>
(028) <wsse:Password>PASSWORD</wsse:Password>
(029) <Organization>yourIPCC</Organization>
(030) <Domain>DEFAULT</Domain>
(031) </wsse:UsernameToken>
(032) </wsse:Security>
(033) </SOAP-ENV:Header>
(034) <SOAP-ENV:Body>
(035) <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="2.0">
(036) <eb:Reference xmlns:xlink="http://www.w3.org/1999/xlink"
(xlink:href="cid:SessionCreateRQ" xlink:type="simple"/>
(037) </eb:Manifest>
(038) </SOAP-ENV:Body>
(039) </SOAP-ENV:Envelope>

Example 2. SOAP-Based SessionCreateRQ Payload

(040) <SessionCreateRQ>
(041) <POS>
(042) <Source PseudoCityCode="yourIPCC"/>
(043) </POS>
(044) </SessionCreateRQ>
SOAP-Based SessionCreateRQ Envelope

Format the SOAP envelopes and payloads for the requests as shown in Examples 1 and 2, respectively.

The client application does the following for each connection:

- Generates a globally unique value for eb:ConversationId (line 008)
- Generates a value for eb:MessageId (line 019)
- Generates values for eb:Timestamp (lines 020 and 021)
- Includes the appropriate values for eb:From and eb:To (lines 009–014)
- Includes the required value for eb:CPAId (line 015). This is the same value as <Organization>.
- Includes the service specific values for eb:Service, eb:type (line 016), and eb:Action (line 017)
- Includes security credentials in the wsse:Security node (lines 024–032)
- (Payloads sent as attachments) Sets the reference to the payload attachment in the xlink:href attribute of the eb:Reference element (line 036)
- (Payloads included in SOAP envelopes) Substitutes the payload for eb:Manifest in the first MIME part

SOAP-Based SessionCreateRQ Payload

The client creates the payload, either as an attachment or included in the SOAP body.

- In the MIME Header, include the value for the content ID. This must match the value of xlink:href in eb:Reference in the SOAP envelope.
- Specifies the document root element. It is recommended that this value match the value for content ID in the MIME Header and eb:Reference/xlink:href (line 036).
- Passes the value for Source/PseudoCityCode (line 042). This is the same value sent with eb:CPAId and Organization in the SOAP envelope.
SOAP-Based SessionCreateRS Response

Example 3. SessionCreateRS SOAP Envelope with wsse:BinarySecurityToken

(001) <?xml version='1.0' encoding='UTF-8'?>
(003)   xmlns:xlink="http://www.w3.org/1999/xlink">
(004)   <SOAP-ENV:Header>
(005)     <eb:MessageHeader xmlns:eb="http://www.ebxml.org/namespaces/messageHeader" eb:version="2.0" SOAP-ENV:mustUnderstand="1">
(006)       <eb:ConversationId>ABC123@clientURL.com</eb:ConversationId>
(007)       <eb:From>
(008)         <eb:PartyId>webservices.sabre.com</eb:PartyId>
(009)       </eb:From>
(010)       <eb:To>
(011)         <eb:PartyId>clientURL</eb:PartyId>
(012)       </eb:To>
(013)     </eb:From>
(014)     <eb:To>
(015)       <eb:PartyId>yourIPCC</eb:PartyId>
(016)     </eb:To>
(017)     <eb:Service eb:type="sabreXML">SessionCreateRS</eb:Service>
(018)     <eb:Action>SessionCreateRS</eb:Action>
(019)     <eb:MessageData>
(020)       <eb:MessageId>mid:20031209-12545-1369@webservices.sabre.com</eb:MessageId>
(021)       <eb:Timestamp>2003-12-09T11:15:13Z</eb:Timestamp>
(022)       <RefToMessageId>mid:20031209-133003-2333@clientURL</RefToMessageId>
(023)       <eb:MessageData>
(025)         <wsse:BinarySecurityToken valueType="String">
(026)           EncodingType="wsse:Base64Binary">
(027)             Shared/IDL:IceSess/SessMgr:1.0/IDL/Common/!
(028)               !ICESMS/RESA!ICESMSLB/RES.LB!-
(029)                 4845652307057192441!339520!0</wsse:BinarySecurityToken>
(030)         </wsse:Security>
(031)       </eb:MessageHeader>
(032)     </eb:Manifest>
(033)   </SOAP-ENV:Body>
(034) </SOAP-ENV:Envelope>
Example 4. SOAP-Based SessionCreateRS Payload

(035) `<SessionCreateRS xmlns="http://www.opentravel.org/OTA/2002/11" version="1" status="Approved">`
(036) `<ConversationId>`ABC123@clientURL.com</ConversationId>`
(037) `</SessionCreateRS>`

**SOAP-Based SessionCreateRS Response Format**

For the format of the response, see examples 3 and 4.

Note the following in the SessionCreateRS response:

- The infrastructure returns a unique message ID in `<eb:RefToMessageId>`. This is a reference to the message ID of the corresponding request (line 020).
- The payloads of the session request messages do not have an `xmlns` attribute with the document root element, but this attribute is returned in the payload of the responses (line 034).
- The `eb:version` attribute returns a number, but this version is independent of the versioning strategy for SOAP-based TPF Connector or open systems-based Sabre APIs (line 034).

**Consuming the SOAP-Based SessionCreateRQ Sabre API Call**

The client sends the SOAP-based SessionCreateRQ Sabre API call to the endpoint for consuming SOAP-based Sabre APIs over HTTPS. For complete information about the URLs and environments, please refer to the section of this document titled, "SOAP-Based Sabre APIs Environments."

The SOAP-based Sabre APIs infrastructure receives and authenticates the request, and creates a connection. The infrastructure then authorizes the security credentials. If required, it also allocates a Sabre session upon authorization.

The infrastructure returns a unique, encrypted security token to the requester on the client side in `wsse:BinarySecurityToken` in the SOAP envelope of the SOAP-based SessionCreateRS Sabre API call. It also returns the same conversation ID and a reference to the message ID in the request.

The connection ID consists of the returned security token and the conversation ID. Its return means the connection to the SOAP-based Sabre APIs infrastructure is alive and a SOAP-based Sabre APIs session (also called a TA) is allocated.

For every connection it creates, the client parses the `eb:ConversationId` and the entire `wsse:Security` node with `wsse:BinarySecurityToken` and stores them for subsequent use in
requests for travel content that use the connection and Sabre session. This makes it possible to reuse the connection when a connection pool is implemented. For complete information about techniques for handling connectivity, please refer to the section of this document titled, “Connection Management.”

Note: Remember that when using a specific Sabre APIs connection and session, the following values must match the values that were used to open the connection via the SOAP-based SessionCreateRQ Sabre API call: eb:ConversationId, eb:CPAId (eb:Organization), and in the payload, PseudoCityCode.

The same value returned via wsse:BinarySecurityToken in the SOAP-based SessionCreateRS Sabre API call must be sent in all subsequent messages leveraging the same conversation.
Request Messages for Travel Content

All SOAP-based open systems and TPF Connector Sabre APIs adhere to the same model for the SOAP envelopes of the requests and responses, as shown in the following examples, but some tags and values in the SOAP envelopes are specific to the API, connection, and service provider. Examples that can be cited include the value for eb:Action, which is a unique, API-specific value, and the value for eb:ConversationId, which is unique to a Sabre APIs connection and session. The use of the eb:Timeout tag itself is implemented by the a service provider. Currently this tag is read only by the TPF Connector, and so it can be included in the SOAP envelopes of SOAP-based TPF Connector Sabre API calls.

The payload messages for all SOAP-based TPF Connector and open systems-based Sabre APIs follow similar models.

Some of the exceptions are noted as follows:

- Each service provider specifies how to use the Version and PseudoCityCode attributes.

The SOAP-based TPF Connector Sabre API, OTA_HotelAvailLLSRQ, is used in Examples 5 and 6. For the service-specific payload messages and formats, please refer to the service documents published on the Sabre Dev Studio.
Example 5. SOAP Envelope of a Request for Travel Content

(001) <?xml version="1.0" encoding="UTF-8"?>
(003)   xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
(004)   xmlns:xlink="http://www.w3.org/1999/xlink"
(005)   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
(006)   <SOAP-ENV:Header>
(007)     <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="2.0">
(008)       <eb:ConversationId>ABC123@clientURL.com</eb:ConversationId>
(009)       <eb:From>
(010)         <eb:PartyId type="urn:x12.org:IO5:01">clientURL</eb:PartyId>
(011)       </eb:From>
(012)       <eb:To>
(013)         <eb:PartyId type="urn:x12.org:IO5:01">webservices.sabre.com</eb:PartyId>
(014)       </eb:To>
(015)       <eb:CPAId>yourIPCC</eb:CPAId>
(016)       <eb:Service eb:type="sabreXML">OTA_HotelAvailLLSRQ</eb:Service>
(017)       <eb:Action>OTA_HotelAvailLLSRQ</eb:Action>
(018)       <eb:MessageData>
(019)         <eb:MessageId>mid:20031209-133003-2334@clientURL</eb:MessageId>
(020)         <eb:Timestamp>2003-12-09T11:15:14Z</eb:Timestamp>
(021)         <eb:TimeToLive>2003-12-09T11:16:12Z</eb:TimeToLive>
(022)         <eb:Timeout>50</eb:Timeout>
(023)       </eb:MessageData>
(024)     </eb:MessageHeader>
(026)       <wsse:BinarySecurityToken valueType="String">
(027)         EncodingType="wsse:Base64Binary">SShared/IDL:IceSess\SessMgr:1.0.IDL/Common/!!ICESMS/RESA!ICESMSLB/RES.LB!-484565230705719244133952010</wsse:BinarySecurityToken>
(028)     </wsse:Security>
(029)     </SOAP-ENV:Header>
(030)     </SOAP-ENV:Body>
(031)     <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="2.0">
(032)       <eb:Reference xmlns:xlink="http://www.w3.org/1999/xlink">
(033)         xlink:href="cid:OTA_HotelAvailRQ"
(034)         xlink:type="simple"/>
(035)     </eb:Manifest>
(036)   </SOAP-ENV:Body>
(037) </SOAP-ENV:Envelope>
Example 6. Payload of a Request for Travel Content

(038) <?xml version="1.0" encoding="UTF-8"?>
(039) <OTA_HotelAvailRQ xmlns="http://webservices.sabre.com/sabreXML/2003/07"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
Version="2003A.TsabreXML1.4.1">
(040) <POS>
(041)  <Source PseudoCityCode="yourPCC"/>
(042) </POS>
(043) <AvailRequestSegments>
(044)  <AvailRequestSegment>
(045)   <StayDateRange Start="2006-11-22T00:00:00" End="2006-11-25T00:00:00"/>
(046)   <RoomStayCandidates>
(047)    <RoomStayCandidate>
(048)     <GuestCounts>
(049)      <GuestCount Count="2"/>
(050)    </GuestCounts>
(051)   </RoomStayCandidate>
(052)   </RoomStayCandidates>
(053)   <HotelSearchCriteria>
(054)    <Criterion>
(055)     <HotelRef HotelCityCode="DFW" ChainCode="MC"/>
(056)    </Criterion>
(057)   </HotelSearchCriteria>
(058)  </AvailRequestSegment>
(059) </AvailRequestSegments>
(060) </OTA_HotelAvailRQ>

Request SOAP Envelopes

Format the SOAP envelopes and payloads for the request as shown in Examples 5 and 6, respectively, using API-specific data values and formats. For detailed common requirements, please refer to the section of this document titled, “SOAP Envelope Field Size Quick Reference.”

The client includes the following:

- The value for eb:ConversationId which is extracted from the connection being used
- The same value for eb:CPAId that was used to open the connection (line 015)
- Values for <eb:MessageId> and eb:Timestamp, and optionally, eb:Timeout (lines 019–020)
- Optionally, the generated value for eb:Timeout. This value must be less than the system default value on the API. Currently, only the TPF Connector-based and Orchestrated Sabre APIs have implemented this feature. If the value is greater than the default value associated with the API, the TPF Connector or Orchestrated APIs will ignore it and use the default value instead. (line 022)
- If sending a time-out value, exclude eb:TimeToLive because these are mutually exclusive. Note, also, that TimeToLive is not supported. (line
021)

For more information about the service time-outs, see “eb:Timeout.”

• Appropriate values for eb:From and <eb:To> (lines 009–014), eb:CPAId (line 015), eb:Service, eb:type (line 016), and eb:Action (line 017)

• The wsse:Security node, which includes wsse:BinarySecurityToken, extracted from the SessionCreateRQ response that opened the Sabre APIs connection being used (lines 025–028)

• (Payloads sent as attachments) The reference to the payload attachment in the xlink:href attribute of the eb:Reference element (line 032)

• (Payloads included in SOAP envelopes) The payload in place of the eb:Manifest element in the first MIME part

Request Payloads

The client includes the following:

• In the MIMEHeader, the value for the content ID. This must match the value of xlink:href in eb:Reference (line 032)

• The document root element. It is recommended that this value match the value for content ID in the MIME Header and eb:Reference /xlink:href. (line 039)

• The value for the xmlns attribute of the document root element. Application developers need to refer to the developer notes for the Web service being used. (line 039)

• A value for the Version attribute that is applicable to the version of the Web service your client is consuming. Obtain the applicable versions and correct format in the API documentation on the Sabre Dev Studio. (line 039)

• The value for the Source/PseudoCityCode. This value must match the values sent with eb:CPAId and Organization in the SOAP envelope of the SessionCreateRQ message that opened the connection. (line 041)

The values for the following three must be the same:

• In all payloads, the IPCC in POS/Source/PseudoCityCode

• In all SOAP envelopes, eb:CPAId

• In the SOAP envelope of SessionCreateRQ, the Organization element

Remember that for all messages sent in a given connection, the value for PseudoCityCode and eb:CPAId must match the value in the SOAP-based SessionCreateRQ Sabre API call that was used to create the SOAP-based Sabre API connection being used.

For the valid version attribute values, MDR subsets, and document root elements, please
consult the design documents, service descriptions, and developer notes located on the Sabre Dev Studio.

**Response Messages with Travel Content**

Responses of all SOAP-based TPF Connector Sabre APIs conform to the following format, shown in examples 7 and 8.

**Example 7. SOAP Envelope of a Response for Travel Content**

```xml
<xml version="1.0" encoding="UTF-8">
  <soap-env:Header>
    <eb:MessageHeader xmlns:eb="http://www.ebxml.org/namespaces/messageHeader" eb:version="2.0" soap-env:mustUnderstand="1">
      <eb:From>
        <eb:PartyId eb:type="urn:x12.org:IO5:01">webservices.sabre.com</eb:PartyId>
      </eb:From>
      <eb:To>
        <eb:PartyId eb:type="urn:x12.org:IO5:01">clientURL</eb:PartyId>
      </eb:To>
      <eb:CPAId>yourIPCC</eb:CPAId>
      <eb:ConversationId>ABC123@clientURL.com</eb:ConversationId>
      <eb:Service eb:type="sabreXML">Hotel Availability</eb:Service>
      <eb:Action>OTA_HotelAvailLLSRS</eb:Action>
    </eb:MessageHeader>
      <wsse:BinarySecurityToken valueType="String" EncodingType="wsse:Base64Binary">Shared/IDL:IceSess\SessMgr:1\0.IDL/Common\\ICESMS\RESA\ICESMSLB\RES.LB!-4845652307057192441339520\0</wsse:BinarySecurityToken>
    </wsse:Security>
  </soap-env:Header>
  <soap-env:Body>
    <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="2.0">
    </eb:Manifest>
  </soap-env:Body>
</soap-env:Envelope>
```

**Example 8. Payload of a Response for Travel Content**
<?xml version="1.0" encoding="UTF-8"?>
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Version="2003A.TsabreXML1.4.1">
  <Success/>
  <RoomStays MoreIndicator="Y">
    <RoomStay>
      <RoomRates>
        <RoomRate RPH="001" RoomTypeCode="STD" RatePlanCode="RAC"/>
        <RoomRate RPH="001" RoomTypeCode="C1D" RatePlanCode="COR"/>
        <RoomRate RPH="001" RoomTypeCode="STD" RatePlanCode="GRR"/>
        <RoomRate RPH="001" RoomTypeCode="A1K,N1K,B2Q,N2Q,C1D,NS1" RatePlanCode="GRT"/>
        <RoomRate RPH="011" RoomTypeCode="STD" RatePlanCode="RAC"/>
        <RoomRate RPH="011" RoomTypeCode="STD" RatePlanCode="BBA"/>
        <RoomRate RPH="011" RoomTypeCode="STD" RatePlanCode="COR"/>
        <RoomRate RPH="011" RoomTypeCode="STD" RatePlanCode="GRR"/>
      </RoomRates>
      <BasicPropertyInfo ChainCode="HI" HotelCode="51645" HotelName="HOLIDAY INN EX STES DFW" HotelCityCode="DFW" AreaID="003NW">
        <TPA_Extensions>
          <Line RPH="001"/>
          <Distance Ind="M"/>
          <CurrencyCode>USD</CurrencyCode>
          <MinRate Amount="94.00" CurrencyCode="USD" DecimalPlaces="2"/>
          <MaxRate Amount="159.00" CurrencyCode="USD" DecimalPlaces="2"/>
          <DirectConnect>
            <DCSelfParticipant Ind="true"/>
            <DCAvailParticipant Ind="true"/>
            <UnAvail Ind="false"/>
            <RequestFail Ind="false"/>
          </DirectConnect>
          <LocationDescription Code="G">
            <Text>GRAPEVINE</Text>
          </LocationDescription>
          <Position Latitude="32.921900" Longitude="-97.080400"/>
          <Address>
            <TPA_Extensions>
              <AddressLine>309 STATE HWY 114 WEST</AddressLine>
              <AddressLine>GRAPEVINE TX 76051</AddressLine>
            </TPA_Extensions>
            <Address>
              <ContactNumbers>
                <ContactNumber PhoneNumber="817-442-5919"/>
                <FaxNumber PhoneNumber="817-442-5960"/>
              </ContactNumbers>
            </Address>
          </Address>
        </TPA_Extensions>
      </BasicPropertyInfo>
      <BasicPropertyInfo ChainCode="HI" HotelCode="53766" HotelName="HOLIDAY INN ADDISON" HotelCityCode="DFW" AreaID="013E">
        <TPA_Extensions>
          <Line RPH="011"/>
        </TPA_Extensions>
      </BasicPropertyInfo>
    </RoomStay>
  </RoomStays>
</OTA_HotelAvailRS>
(082)  <Distance Ind="M"/>
(083)  <CurrencyCode>USD</CurrencyCode>
(084)  <MinRate Amount="117.99" CurrencyCode="USD" DecimalPlaces="2"/>
(085)  <MaxRate Amount="139.00" CurrencyCode="USD" DecimalPlaces="2"/>
(086)  <DirectConnect>
(087)    <DCSellParticipant Ind="true"/>
(088)    <DCAvailParticipant Ind="true"/>
(089)    <UnAvail Ind="false"/>
(090)    <RequestFail Ind="false"/>
(091)  </DirectConnect>
(092)  <LocationDescription Code="G">
(093)    <Text>ADDISON TX</Text>
(094)  </LocationDescription>
(095)  </TPA_Extensions>
(096)  <Position Latitude="32.958500" Longitude="-96.827000"/>
(097)  <Address>
(098)    <TPA_Extensions>
(099)      <AddressLine>4960 ARAPAHO ROAD</AddressLine>
(100)      <AddressLine>ADDISON TX 75001</AddressLine>
(101)    </TPA_Extensions>
(102)  </Address>
(103)  <ContactNumbers>
(104)    <ContactNumber PhoneNumber="1-972-490-1212"/>
(105)    <TPA_Extensions>
(106)      <FaxNumber PhoneNumber="1-972-233-4283"/>
(107)    </TPA_Extensions>
(108)  </ContactNumbers>
(109)  </BasicPropertyInfo>
(110)  </RoomStay>
(111) </RoomStays>
(112) <TPA_Extensions>
(113) <HostCommand>ARS01S093HOTDFW/22NOV-25NOV2/MC</HostCommand>
(114) </TPA_Extensions>
(115)</OTA_HotelAvailRS>
Consuming a Travel-Based Service

Please note the following in the response:

**SOAP Envelope**

- The SOAP-based Sabre APIs infrastructure returns a unique message ID and a reference to the message ID of the corresponding request in `<eb:RefToMessageId>` (lines 016–017).
- The security token is returned in `<wsse:BinarySecurityToken>` (line 022).

**Payload**

- For SOAP-based TPF Connector Sabre APIs, the Version attribute of the document root element returns the service version requested (line 034).
- For SOAP-based TPF Connector Sabre APIs, the business application returns the Sabre system command used to format the request in `<HostCommand>` (line 113).
SessionCloseRQ Message

The model for the SOAP-based SessionCloseRQ Sabre API call, which is required to close Sabre APIs connections, is shown in examples 9 and 10.

Example 9. SOAP-Based SessionCloseRQ Envelope

(001) <?xml version='1.0' encoding='UTF-8'?>
(003) xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
(004) xmlns:xlink="http://www.w3.org/1999/xlink"
(005) xmlns:xsd="http://www.w3.org/1999/XMLSchema">
(006)   <SOAP-ENV:Header>
(007)       <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="2.0">
(008)         <eb:ConversationId>ABC123@clientURL.com</eb:ConversationId>
(009)         <eb:From>
(010)           <eb:PartyId eb:type="urn:x12.org:IO5:01">clientURL</eb:PartyId>
(011)         </eb:From>
(012)         <eb:To>
(013)           <eb:PartyId type="urn:x12.org:IO5:01">webservices.sabre.com</eb:PartyId>
(014)          </eb:To>
(015)       </eb:From>
(016)       <eb:CPAId>yourIPCC</eb:CPAId>
(017)     </eb:MessageHeader>
(020)     </wsse:Security>
(021)   </SOAP-ENV:Body>
(022) </SOAP-ENV:Envelope>
Example 10. SOAP-Based SessionCloseRQ Payload

(036) <SessionCloseRQ>
(037)   <POS>
(038)     <Source PseudoCityCode="yourIPCC"/>
(039)   </POS>
(040) </SessionCloseRQ>

SOAP-Based SessionCloseRQ Envelope

Format the SOAP envelopes and payloads for the requests as shown in examples 9 and 10, respectively.

Note: For any values not specifically called out or described in the reference section, format the messages as shown. For detailed common requirements, see Appendix A.

Your client does the following for each connection:

- Passes the conversation ID of the connection to close for <eb:ConversationId> (line 008)
- Generates a value for <eb:MessageId> and eb:Timestamp (lines 019–020)
- Includes the appropriate values for <eb:From> and eb:To (lines 009–014)
- Includes the required value for eb:CPAld (line 015). This is the same value as <Organization> in the SessionCreateRQ used to open the connection.
- Includes the service specific values for eb:Service, eb:type (line 016), and <eb:Action> (line 017)
- Passes the security token of the connection to close in wsse:Security@wsse:BinarySecurityToken (lines 024–027)
- (Payloads sent as attachments) Sets the reference to the payload attachment in the xlink:href attribute of the <eb:Reference element (line 032)
- (Payloads included in SOAP envelopes) Substitutes the payload for <eb:Manifest in the first MIME part

SOAP-Based SessionCloseRQ Payload

The client creates the payload, either as an attachment or included in the SOAP body.

The client does the following:

- In the MIME Header, includes the value for the content ID. This must match the value of xlink:href in eb:Reference in the SOAP envelope.
- Specifies the document root element. It is recommended that this value match the value for content ID in the MIME Header and eb:Reference / xlink:href (line
036).

- Passes the value for Source/PseudoCityCode (line 042). The is the same value sent with eb:CPAId and Organization in the SOAP envelope.

**Note:** In all request messages using a given connection, the values for following must be the same:

- In payloads, the IPCC in POS/Source/PseudoCityCode
- In SOAP envelopes, eb:CPAId
- In SOAP envelope of SessionCreateRQ, the Organization element
SOAP-Based SessionCloseRS Message

The SessionCloseRQ Sabre API call terminates both the Sabre APIs connection and its associated Sabre APIs session, and renders the security token invalid.

Example 11. SOAP-Based SessionCloseRS Envelope

```xml
(001) <?xml version="1.0" encoding="UTF-8" ?>
(003)  <soap-env:Header>
(004)   <eb:MessageHeader eb:version="1.0" soap-env:mustUnderstand="1" 
   xmlns:eb="http://www.ebxml.org/namespaces/messageHeader">
(005)    <eb:From>
(006)     <eb:PartyId eb:type="URI">webservices.sabre.com</eb:PartyId>
(007)    </eb:From>
(008)    <eb:To>
(009)     <eb:PartyId eb:type="URI">clientURL</eb:PartyId>
(010)    </eb:To>
(011)    <eb:CPAId>yourIPCC</eb:CPAId>
(012)    <eb:ConversationId>ABC123@clientURL.com</eb:ConversationId>
(013)    <eb:Service eb:type="sabreXML">Session</eb:Service>
(014)    <eb:Action>SessionCloseRS</eb:Action>
(015)    <eb:MessageData>
(016)     <eb:MessageId>mid:20030707-12545-1370@webservices.sabre.com</eb:MessageId>
(017)     <eb:Timestamp>2006-06-23T15:29:09</eb:Timestamp>
(018)     <RefToMessageId>mid:20031209-133003-2335@clientURL</RefToMessageId>
(019)    </eb:MessageData>
(020)  </eb:MessageHeader>
   wsse:BinarySecurityToken>
(023)    <wsse:Security>
(024) </wsse:Security>
(025) </soap-env:Header>
(026) </soap-env:Body>
(027) <SessionCloseRS status="Approved" version="1" xmlns="http:// 
   www.opentravel.org/OTA/2002/11"/>
(028) </soap-env:Body>
(029) </soap-env:Envelope>
```

Example 12. SOAP-Based SessionCloseRS Payload

```xml
(030) <?xml version="1.0" encoding="UTF-8" ?>
(031) <SessionCloseRS xmlns="http://www.opentravel.org/OTA/2002/11"version="1" status="Approved" />
```
SOAP-Based SessionCloseRQ Response Format

If the connection is closed successfully, the SOAP envelope and payload messages are returned. For an example of the response payload, see example 12.

Please note the following in the responses:

- The SOAP-based Sabre APIs infrastructure returns a unique message ID with a reference to the message ID of the corresponding request in <eb:RefToMessageId>. (lines 016–017)
- The payloads of the session request messages do not have an xmlns attribute with the document root element, but this attribute is returned in the payload of the responses.
- The eb:version attribute returns a number, but this independent of the versioning standards of SOAP-based TPF Connector Sabre APIs. (line 030)
- Only the root element and attributes are returned when a connection is closed properly. (line 030)

Consuming the SessionCloseRQ Sabre API Call

When a SOAP-based Sabre APIs connection is closed successfully, the following happens:

- The associated Sabre session is released.
- The security token is invalidated.
- The SOAP-based SessionCloseRS MessageHeader is returned to the requester.
- If a Sabre session or TA was allocated, the content in the Sabre work area/AAA is discarded and the Sabre session is returned to the session pool.
Payloads Formatted Inside SOAP Envelopes

For Java clients that consume Web services without WSDL, sending the payload as an attachment to the SOAP envelope is preferred. If a particular development tool does not support attachments, it is also possible to send the payload inside the envelope.

If the client consumes Sabre APIs via WSDL, it must include the payload inside the body of the first MIME part or SOAP envelope. This is shown in example 13.

To include the payload inside the SOAP envelope, do the following:

1. Modify an existing message using the SOAP with Attachments protocol.
2. Provide requirements that are specified by Sabre APIs for the envelopes and payloads.
3. Insert the payload of the second MIME part into the first MIME part. Remove the eb:Manifest node from the SOAP envelope, and insert the payload. Lines 28 through 49 represent the payload.

Example 13. Message Payload Inside SOAP Envelope Body

```xml
(002)   xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
(003)   xmlns:xlink=http://www.w3.org/1999/xlink xmlns:xsd="http://www.w3.org/1999/XMLSchema">
(004)   <SOAP-ENV:Header>
(005)     <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="2.0">
(006)       <eb:ConversationId>ABC123@clientURL.com</eb:ConversationId>
(007)       <eb:From>
(008)         <eb:PartyId type="urn:x12.org:IO5:01">clientURL</eb:PartyId>
(009)       </eb:From>
(010)       <eb:To>
(011)         <eb:PartyId type="urn:x12.org:IO5:01">webservices.sabre.com</eb:PartyId>
(012)       </eb:To>
(013)       <eb:CPAId>yourIPCC</eb:CPAId>
(014)       <eb:Service eb:type="OTA">Hotel</eb:Service> (015)
(016)       <eb:Action>OTA_HotelDescriptionRQ</eb:Action>
(017)       <eb:MessageData>
(018)         <eb:MessageId>mid:20031209-133003-2333@clientURL</eb:MessageId>
(019)         <eb:Timestamp>2003-12-09T11:15:12Z</eb:Timestamp>
(020)         <eb:Timeout>55</eb:Timeout>
(021)       </eb:MessageData>
(022) </eb:MessageHeader>
(024) <wsse:BinarySecurityToken valueType="String">
```

<xml>

<wsse:Security>
  <wsse:BinarySecurityToken EncodingType="wsse:Base64Binary">
    Shared/IDL:IceSess\SessMgr:1.0.IDL/Common/!ICESMS\RESC!ICESMSLB\RES.LB!-4954987477210575357!252506!0</wsse:BinarySecurityToken>
  </wsse:Security>
</xml>

<SOAP-ENV:Header>
    <POS>
      <Source PseudoCityCode="yourPCC"/>
    </POS>
    <AvailRequestSegments>
      <AvailRequestSegment>
        <StayDateRange Start="2003-10-29" End="2003-10-30"/>
        <RoomStayCandidates>
          <RoomStayCandidate>
            <GuestCounts>
              <GuestCount Count="2"/>
            </GuestCounts>
          </RoomStayCandidate>
        </RoomStayCandidates>
        <HotelSearchCriteria>
          <Criterion>
            <HotelRef HotelCode="62532"/>
          </Criterion>
        </HotelSearchCriteria>
      </AvailRequestSegment>
    </AvailRequestSegments>
  </OTA_HotelAvailRQ>
</SOAP-ENV:Header>
</soap-env:Envelope>
Chapter 4: Connection Management

Chapter four discusses connections and connection strategies related to consuming SOAP-based Sabre APIs.

SOAP-Based Sabre APIs Connections

Connections are open channels to the SOAP-based Sabre APIs infrastructure.

When a client requests a connection with Sabre APIs and the client is authenticated and authorized, an open channel to Sabre APIs is created. If a Sabre APIs session is required, it is allocated at the same time.

The distinction between the terms “connection” and “session” is purely semantic. A client application requests a connection to the SOAP-based Sabre APIs infrastructure, and upon success, a SOAP-based Sabre APIs session is created simultaneously with a business application or data center within Sabre. A connection is on the client side, and a session is on the Sabre side, as illustrated in Figure 5. The time-out value for a connection and session are synchronized, occurring simultaneously.
Figure 5. Connection versus Session

A connection is not a client side shopping cart and it does not maintain state in the AAA (referred to as the Sabre work area/AAA) of the Sabre host system.

**Connection Management Messages**

In addition to using OpenTravel specifications for Sabre XML request and response schemas, Sabre XML has added messages for managing SOAP-based Sabre API connections.

The SOAP-based SessionCreateRQ and SessionCloseRQ Sabre API calls open and close connections explicitly. These Sabre APIs use the SessionCreateRQ/RS message pair and the SessionCloseRQ/RS message pair, respectively.

Another session management service, OTA_PingRQ, is used to keep the SOAP-based Sabre API connections alive.

More information about these APIs also appears in the API description and XML design documentation located on the Sabre Dev Studio.

**Connecting to Sabre APIs**

There is one way to connect to the SOAP-based Sabre APIs. The general steps are for the client to send the SOAP-based SessionCreateRQ Sabre API call to request a connection that includes the client’s security credentials, a conversation ID, along with several other required values, let the gateway authenticate and authorize the security credentials, and receive a security token in the response. The return of the security token means a connection has been created successfully.
A summary of the process to connect is presented as follows:

**Request 1**

The SOAP message for the SOAP-based SessionCreateRQ Sabre API is created on the client side.

- Create the SOAP envelope in the required format for SOAP-based Sabre APIs. Include the required values for the SessionCreateRQ Envelope. Generate the value for eb:ConversationId, and include the values for eb:CPAId and your security credentials in wsse:Security node. Ensure the value for eb:Action for this request is SessionCreateRQ.
- Create the payload, either as an attachment or incorporated into the SOAP body.
- Send the SOAP-based SessionCreateRQ Sabre API call to the endpoint for consuming SOAP-based Sabre APIs over HTTPS. The client can connect to the Production URL or a URL representing one of the certification or development systems. For complete information about the URLs and environments, please refer to the section of this document titled, “SOAP-Based Sabre APIs Environments.”

**Response 1**

- The SOAP-based Sabre APIs infrastructure receives the request, authenticates it, and creates a connection. The infrastructure then authorizes access to the business application or system within Sabre based on the security credentials presented. Upon authorization, it allocates a Sabre session if required. (A Sabre session is another name for a TA; Sabre session is used in this documentation. Sabre sessions are discussed later in this chapter.)
- The infrastructure returns a unique, encrypted security token to the client side in wsse:Security@wsse:BinarySecurityToken in the SOAP envelope of the SessionCreateRS response. It also returns the same conversation ID and a reference to the message ID that was in the request.
- The connection ID consists of the returned security token and the conversation ID. Its return means the connection to the SOAP-based Sabre APIs infrastructure is alive and a Sabre session is allocated.
- The client extracts and stores the eb:ConversationId and the entire wsse:security@wsse:BinarySecurityToken node for inclusion in subsequent workflows and requests that use this connection.
- When sending subsequent Sabre API calls for travel content, the connection ID is needed for all transactions with the SOAP-based Sabre APIs infrastructure that use a specific connection, whether the client maintains state or not.
Closing Connections

When the client application needs to close a SOAP-based Sabre APIs connection, it must include the connection ID of the connection that it wants to close in the SOAP-based SessionCloseRQ Sabre API call. A summary of the process is presented as follows.

Request 1

The SOAP message for the SOAP-based SessionCloseRQ Sabre API is created on the client side.

- Create the SOAP envelope in the required format for the SOAP-based Sabre APIs. Include the required values for the SessionCloseRQ SOAP envelope. It is especially important to include the values for eb:ConversationId, eb:CPAId, and the security token of the connection to be closed. These values were sent in the SessionCreateRQ request and returned in SessionCreateRS response. Ensure the value for eb:Action for this request is SessionCloseRQ.

- Create the payload, either as an attachment or incorporated into the SOAP body.

- Send the SOAP-based SessionCloseRQ Sabre API call to the endpoint for the Sabre APIs environment where the connection lives. (For complete information about the URLs and environments, please refer to the section of this document titled, “SOAP-Based Sabre APIs Environments.”)

Response 1

- The SOAP-based Sabre APIs infrastructure receives the request. The infrastructure closes the connection and returns the previously allocated Sabre session to the session pool. The Sabre work area/AAA is cleared, and the security token is rendered invalid. The MessageHeader of SessionCloseRS message is returned to the client.
Relationship Between Connections and Sessions

As stated previously, when a requester’s security credentials are authorized, a SOAP-based Sabre APIs session is allocated along with the connection. The type of session depends on the configuration of the user ID that was used to open the connection.

Sessions with SOAP-based TPF Connector Sabre APIs

The SOAP-based TPF Connector Sabre APIs obtain their content and functionality from the legacy Sabre host system, therefore, the security credentials and user IDs of subscribers who consume SOAP-based TPF Connector Sabre APIs are specifically configured to create sessions within the legacy Sabre host system.

A Sabre host session is a specific type of session. This type of session is associated with a particular LNIATA residing in the native, legacy TPF-based Sabre systems (also referred to as PSS). The user IDs of the Sabre system require and use LNIATAs or TAs. They are assigned a finite quantity of TAs in a TAM pool for each IPCC they have. (The TAM pool is referred to as a session pool in this discussion and document.)

Allocation of Sabre APIs Sessions

Whenever security credentials that require a Sabre session open a connection, the SOAP-based Sabre APIs infrastructure creates a new connection and allocates a Sabre session from the subscriber’s session/TAM pool. The Sabre session becomes active and is no longer available in the session pool until the connection is closed or the connection/session time out.

The Sabre session and connection are synchronized, sharing the same time-out values.

Shopping Cart Functionality and the Sabre Work Area/AAA

A Sabre session has an active AAA (the AAA is referred to as the Sabre work area in this discussion and document). The Sabre work area/AAA provides shopping cart functionality on the client side. When the client calls SOAP-based TPF Connector Sabre APIs, content from the Sabre host system is temporarily placed in the work area. The client can use the host content in the Sabre work area/AAA in a stateful or stateless way. Some SOAP-based TPF Connector Sabre APIs rely on content placed in the work area by previous service calls in the same session, while other services do not have dependencies on services to place content in this work area.

As long as a client uses the security token and conversation ID from a specific connection and there is activity, the connection remains alive, the Sabre session is active, and content in the Sabre work area/AAA is retained.

To store transactions in the Sabre work area/AAA in a specific Sabre session, the client must use the SOAP-based Sabre API designed to end the transaction when the workflow is completed, i.e. EndTransactionLLSRQ.

When reusing a connection, the client is strongly advised to clear the Sabre work area/AAA
before sending messages in a new workflow. The SOAP-based IgnoreTransactionLLSRQ Sabre API can be used to clear the Sabre work area/AAA. This prevents mingling content from the new workflow with content from the previous use of the Sabre session.

**Note:** If the client crashes or experiences a network outage while a Sabre session is active, the content that was retrieved during the session remains in the work area until it times out. If the client or network is brought online before the time-out period expires, the content from the Sabre session remains. Moreover, if the new client instance re-uses a connection ID that was active before the system outage, the content for the Sabre session remains in the Sabre work area/AAA because the connection was not closed explicitly. By not specifically clearing the work area, the client risks mingling content from the re-used, recovered connection ID and associated Sabre session with your new workflow.

**Release of Sabre Sessions**

When a client or connection manager successfully closes a connection using the SOAP-based SessionCloseRQ Sabre API, the Sabre APIs connection is terminated and the security token is rendered invalid. The content in the Sabre work area/AAA is discarded, and the Sabre session (or TA) is released and returned to the session pool.

If the client lets unneeded connections time out instead of closing them properly with the SOAP-based SessionCloseRQ Sabre API call, it is possible that all connections and sessions in the session pool will be in use and unavailable until they time out.

Letting sessions time out on their own puts client applications in a situation where they will not have any connections available for log in, causing them to have to wait until the connections time out before they can log in.

If all Sabre sessions in the session pool are allocated, the client will receive an error message when it tries to log in and open another session.

**Sunday System Housekeeping**

The Sabre system maintenance program known as NORMOAA runs every Sunday morning between 00:15 and 00:20 Central time. NORMOAA clears all the Sabre API sessions.

There is a window just before NORMOAA runs during which clients with active sessions receive a warning message in response to any command. The warning would look like this if the client sent a request via the SOAP-based SabreCommandLLSRQ Sabre API call:

```xml
<SabreCommandLLSRQ
xmlns="http://webservices.sabre.com/sabreXML/2003/07"
Version="2003A.TsabreXML1.6.1">

<Response><![CDATA[SYSTEM HOUSKPING REQUIRES AAA TO BE CLEARED
RE ENTER LAST INPUT THEN COMPLETE OR]]>
```
END TRANSACTIONS IN ALL AREAS

ENTER SOALL WITHIN 04 MINUTES AND THEN

SIGN BACK IN TO CONTINUE WORKING]]>]<Response>

</SabreCommandLLRS>

The warning would look like this if the client sent most other SOAP-based TPF Connector Sabre API requests:


<TPA_Extensions>

<HostCommand>†††A††RS01S093†JX PNR</HostCommand>

</TPA_Extensions>

<Errors>

>Error ErrorCode="SessionFailure-103" Severity="High" ErrorMessage="Parameter not supported">

>ErrorInfo>

<Message>SYSTEM HOUSKPING REQUIRES AAA TO BE CLEARED
RE-ENTER LAST INPUT THEN COMPLETE OR
END TRANSACTIONS IN ALL AREAS
ENTER SOALL WITHIN 04 MINUTES AND THEN
SIGN BACK IN TO CONTINUE WORKING</Message>

</ErrorInfo>

</Error>

</Errors>

</TravellItineraryRS>

The “WITHIN 04 MINUTES” string will vary based on how many minutes are left until NORMOAA runs. If a client receives this response when performing stateless transactions
availability, shopping, etc.) they should simply close and reopen the session. If they are in a stateful transaction, i.e. building a PNR the client should immediately end transaction, close and reopen the session, and retrieve the PNR to continue. If NORMOAA runs before the PNR is closed all changes since the last end transaction will be discarded. Data in the work area before a PNR has been created will be lost entirely.

To refresh all active sessions in use at 00:15 Central time on Sunday the client should send the SOAP-based SessionCloseRQ Sabre API call followed by the SOAP-based SessionCreateRQ Sabre API call. The client does not need to receive the warning message before refreshing the session. Sabre recommends that clients who maintain a pool of open sessions close and reopen them after 00:15 Central time on Sunday’s as part of routine maintenance.

If pooled sessions are not refreshed in this way the active binary security tokens will be expired by NORMOAA and the client will receive a USG_INVALID_SECURITY_TOKEN error. By anticipating NORMOAA and performing a routine refresh client applications will avoid this inconvenience.

SOAP-Based Open Systems Sabre APIs

For subscribers who use SOAP-based open systems Sabre APIs, a session is created for use, as required by business applications and systems of other service providers within Sabre. This session is not necessarily a Sabre host session or TA.

Allocation of User Names to Connections and Sessions

As stated in the requirements, subscribing organizations receive one non-administrative user ID for every 50 Sabre sessions in the IPCC’s session pool. The purpose of these user IDs is to log in and connect to Sabre APIs.

When client applications create SOAP-based Sabre APIs connections, it is recommended for them to rotate user IDs, in round-robin fashion, using a different user ID for each session to provide for failover. An example is as follows:

If a user ID or password becomes unusable for any reason, such as the password is compromised or the ID is corrupted, the client application can continue to create new connections with the uncorrupted user IDs. The sessions with the uncorrupted user IDs remain in use.

To do this, the client must set up every EPR (user ID) the same way.
Time-Outs on SOAP-Based Sabre APIs Connections

A Sabre APIs connection remains active until either of the following occurs:

- The SOAP-based SessionCloseRQ Sabre API messages are exchanged.
- The period of permitted inactivity has been exceeded for the connection and it times out.

Each Sabre APIs connection has a time-out value associated with it. The default timeout value is 15 minutes. The default is set when security credentials are created for client use. (For more information, contact your Sabre account representative.)

**Note:** It is very important for consuming clients and connection managers to know the time-out values associated with their security credentials used for Sabre APIs.

To prevent an established SOAP-based Sabre APIs connection and associated Sabre session from timing out, a client can send any Sabre API call. Sending the OTA_PingRQ service with a valid conversation ID and security token is recommended for this purpose. The SOAP-based OTA_PingRQ Sabre API call has no effect on content in the Sabre work area/AAA aside from keeping the session alive.

It is not advisable to let connections time out. It is the responsibility of the client to either close Sabre APIs connections explicitly with the SOAP-based SessionCloseRQ Sabre API call before the time-out values are reached or to keep their connections alive while they are needed. If activity has not occurred within the pre-determined time-out limit, SOAP-based Sabre APIs connections are not guaranteed to be alive.

User-Defined Time-Outs on SOAP-based TPF Connector Sabre APIs

A SOAP-based Sabre API time-out is not to be confused with a system time-out on a connection or session. The service time-out is a time-out on the SOAP-based Sabre API transaction. Each individual, SOAP-based Sabre API has a system-defined time-out value which a user cannot override. The SOAP-based TPF Connector Sabre APIs accept user-defined time-outs that are less than or equal to the system default time-out value on the API. Currently, all SOAP-based TPF Connector Sabre APIs have a default time-out of 60 seconds, and these values are published on the Sabre Dev Studio.

The client application can decrease the default time-out on any individual SOAP-based TPF Connector Sabre API by passing a value that is less than the default in the SOAP envelope in `eb:Timeout`, as shown below.

```
<eb:Timeout>40</eb:Timeout> <!-- 6 character maximum. This value is in seconds. -->
```
If the client application sends a value greater than the default time-out, it is ignored. If the client application includes eb:Timeout in the SOAP envelope for any of the non-SOAP-based TPF Connector Sabre APIs, other providers may ignore it.

**Connectivity Handling Approaches**

The following solutions for handling connections using Sabre APIs are discussed:

- Basic connections – This solution creates a conversation for one time use.
- Connection managers and connection pools – This solution stores and retrieves open connections maintained in a pool.

**Basic Connections**

Basic connections are the simplest approach for connecting to the SOAP-based Sabre APIs. A basic connection is similar to a conversation. The client application starts a conversation (open a connection with the SOAP-based SessionCreateRQ Sabre API), exchanges requests for content and receives the responses (send and receive Sabre APIs messages in the form of SOAP-based TPF Connector or open systems-based Sabre APIs), and then ends the conversation (close the connection with the SessionCloseRQ Sabre API). The client to connection ratio is 1:1, in other words, one client equals one connection. This is illustrated in Figure 7.

![Figure 7. Basic Connection](image)

When a client application needs a connection to the SOAP-based Sabre APIs infrastructure to send a business workflow, it opens a new connection. With this solution, the client retains and resends the connection ID in all SOAP-based Sabre APIs requests in a business workflow, but the client does not store the connection ID for use beyond the current connection. The client can temporarily store the connection ID in memory or elsewhere until it is done using the connection. When the client opens a new connection, it stores the new security token, overwriting the previous one. The conversation ID can be reused in a new connection.
The client can actually send multiple workflows before closing the connection. The point of the basic connection is for a single client to open one connection, to send one or more workflows using the same connection ID, and to close the connection when the workflows are completed. This simultaneously terminates the Sabre APIs session allocated with the connection.

An example of the flow using a single, basic connection sending multiple workflows follows.

**Request 1**

- The client creates the SOAP message for the SOAP-based SessionCreateRQ Sabre API in the required format with the required values, and sends it to the endpoint for consuming Sabre APIs over HTTPS.

**Response 1**

- The SOAP-based Sabre APIs infrastructure authenticates and authorizes the client, and creates the connection. Upon authorization, a Sabre APIs session is also allocated from the subscriber’s session pool, as required.

- In the SOAP envelope of the SOAP-based SessionCreateRS Sabre API call, a unique, encrypted security token is returned to the client in wsse:Security@wsse:BinarySecurityToken and the conversation ID is returned.

**Request 2**

- The client sends the first message in a business workflow, requesting travel content.

- In the SOAP envelope, the client extracts the values for eb:ConversationId and wsse:BinarySecurityToken that were returned in the SessionCreateRS response message, and includes them in the request.

- The client formats the payload as described in the section of this document titled, “Request Messages for Travel Content.”

- The client requests a specific SOAP-based Sabre API version via the root Version attribute, and includes other service-specific elements and values. The client includes the IPCC for the PseudoCityCode attribute, which is the same value as eb:CPAId and Organization in wsse:Security in the SessionCreateRQ SOAP envelope.

  **Note:** For the specific values and valid data elements please consult the design, schema, and developer notes on the Sabre Dev Studio.

**Response 2**

- The service provider’s business application within Sabre retrieves the requested content and returns it in the response payload. The security token and conversation ID in the request are returned.
The client parses the content it wants from the response payload along with the security token and conversation ID, which it stores for use in all messages in the workflow.

**Request 3**

- The client sends the remaining requests for travel content in the workflow, formatting the SOAP messages as in Request 2, including the extracted security token and conversation ID.

**Response 3**

- The business application retrieves the requested content and returns it in the response payload and SOAP message as described previously in Response 2.
- When the client has parsed all content it wants from the payload and is done with the workflow, it ends the transaction.

**Request 4**

- The client sends the SOAP-based EndTransactionLLSRQ Sabre API call to save the transaction and PNR that are temporarily in the Sabre work area/AAA of the Sabre system.

**Response 4**

- The SOAP-based EndTransactionLLSRQ Sabre API call returns a record locator for the PNR to the client.

**Request 5**

- (Optional) The client sends messages in a second workflow, formatting the messages for travel content the same way as the first travel workflow. Because this is a single client using a single connection, the client passes the same conversation ID, security token, and value for eb:CPAId used to open the connection in all requests.

**Response 5**

- The service provider’s business application obtains the requested content and returns it in the response payloads.

**Request 6**

- The client sends the SOAP-based EndTransactionLLSRQ Sabre API call to save the transaction and PNR in the Sabre system.

**Response 6**

- The SOAP-based EndTransactionLLSRQ Sabre API call returns a
record locator to the client.

Request 7

- The client requests termination of the connection by sending the SOAP-based SessionCloseRQ Sabre API call. The SOAP envelope includes the same values for eb:ConversationId, wsse:BinarySecurityToken, and eb:CPAId used to open the connection.

Response 7

- The Sabre APIs infrastructure ends the session and closes the connection simultaneously. It also renders the security token invalid. The SOAP-based SessionCloseRS Sabre API call is returned to the client.

When to Use Basic Connections

If the need for connections is low in volume or if the client application is performing batch processing, this solution is suggested. Low volume is defined by several hundred connections per hour, that is, fewer than 0.25 TPS or an average of 900 individual Sabre APIs calls per hour during peak times.

Advantages and Disadvantages

The advantages of implementing basic connections are low cost and simple architecture. The drawbacks are little or no session recovery, no failover, and limited scalability.

Connection Pools

Implementing a pooling design for caching and managing connections is recommended. Connection pooling is a widely-used practice for managing connections effectively.

A connection manager opens and maintains multiple concurrent connections based on projected volumes and business model. These connections are stored in a connection pool, a repository of multiple open connections which are kept alive and ready when clients need to send travel workflows. The connection pool has multiple open connections to Sabre APIs. With a connection pool, applications can have multiple clients. A many-to-one ratio of connections to clients exists; generally more open connections than clients.

Designing and implementing a connection manager is more complicated than using the basic connection approach.

The connection pool is one component of a connection manager. With connection pools, a connection manager creates the connections it needs by sending multiple SessionCreateRQ requests and storing the connection IDs. Again, the connection ID includes the conversation ID and security token. With a pool, the connection manager also needs to store a client ID as a reference to the client instance using the connection, a time stamp, and connection status. The connection manager persists the connection IDs so that clients can reuse them.

When needed, a client obtains an available connection from the pool to send Sabre API requests that make up a business workflow. As connections are needed, a client retrieves
an available connection from the pool, and passes the connection ID in all messages in the workflow it sends to Sabre APIs. When the client is done, the connection ID is returned to the pool for reuse. The connection manager refreshes the open connections in the pool to prevent them from timing out.

A connection manager has thresholds defined for high and low volume traffic, and tries to maintain the clients needed during the high and low volumes. Then when traffic volumes are low, the connection manager closes some of the connections.

An example of a workflow that obtains an open connection from a connection pool follows, using the SOAP-based TPF Connector Sabre APIs for the travel messages. The details of special values to pass are not included in this example. To view this information, please refer to Appendix A as well as the documentation on the Sabre Dev Studio.

The connection manager sends multiple SessionCreateRQ requests to create Sabre API connections for the connection pool on the client side.

**Request 1**

- The connection manager is initialized. It opens multiple connections per the threshold defined at initialization. It uses the SessionCreateRQ Sabre API in the required format with the required values, and sends them to the endpoint for consuming Sabre APIs over HTTPS.

- In particular, the SOAP envelopes of all requests include a unique client-generated value for eb:ConversationId, the assigned value for eb:CPAId, and the security credentials for consuming Sabre APIs in the wsse:Security node as follows: wsse:UsernameToken, wsse:Password, Organization, and Domain.

**Response 1**

- The Sabre APIs infrastructure authenticates and authorizes access based on the security credentials in the request

- For user IDs that require Sabre host access, allocates one Sabre session per connection

- For every request, returns a unique security token in the BinarySecurityToken element in the SOAP envelope of each SessionCreateRS response messages

- Returns the same conversation ID to each request

- Note: Remember that when a client uses a specific Sabre APIs connection and Sabre session, the following values must match the values that were used to open the connection with SessionCreateRQ:

  - eb:ConversationId, eb:CPAId (eb:Organization), and in the payload, PseudoCityCode. The same value returned in wsse:BinarySecurityToken in SessionCreateRS must be sent in all messages using the connection.

  - The connection manager stores the connection IDs in the connection pool. The
connection IDs are in the pool, waiting for a client to request one. The connection manager extracts and stores the conversation ID and security token from the SessionCreateRS response. It also stores the time stamp and creates a client ID.

- The client requests a connection ID from the connection pool. When the client needs to send a travel workflow, the client requests an open connection from the connection pool.

- The connection manager clears the Sabre work area/AAA before handing over the connection ID to the client. The connection manager sends the IgnoreTransactionLLSRQ Sabre API call, which discards any content that remains from a previous Sabre session that used the connection ID. Remember that the connection ID consists of the security token and conversation ID used to create the connection. While it is using the connection, the client stores the connection ID for use in all requests in the workflow.

- The client exchanges messages that represent a travel workflow. The client includes connection ID information in all request messages in this workflow.

- An example of a travel workflow is an exchange of messages that search for air availability, request an air segment, and then find lower fares.

- While using the connection, the client sends the request messages one at a time, waiting for a response before sending the next request.

- When finished with a SOAP-based TPF Connector Sabre APIs workflow, the client stores the transactions in the Sabre system by sending the appropriate Sabre API, in this case, EndTransactionLLSRQ.

- The client has ended the workflow and returns the connection ID to the pool for reuse by another client.

- A new client requests a connection from the pool. In concurrence with the first client using a connection ID, a second client can also request an open connection from the pool, send a workflow, and return the connection ID when finished with the workflow in similar fashion. As many clients as your business model needs and your capacity planning will allow can independently retrieve open connections from the pool and send workflows.

- The connection manager clears the work area before it hands a connection ID to a client. When the connection manager retrieves an existing connection from the pool, it retrieves the connection ID and clears the Sabre work area/AAA of any content that lingering from the previous Sabre session by sending the appropriate SOAP-based TPF Connector Sabre API, in this case IgnoreTransactionLLSRQ. Depending on the business model, application developers can clear the Sabre work area/AAA when a workflow is completed or just before beginning a new one.

- The connection manager refreshes the open connections in the pool to prevent time-outs. The connection manager keeps the connections open or alive by time stamping them with the OTA_PingRQ Sabre API.

- The connection manager closes excessive connections. When traffic volume is low
and fewer connections are needed, the connection manager closes some connections to maintain the minimum threshold it has defined.

- The connection manager obtains the connection IDs of the connections to close by using the conversation ID and security token used to open the connections.

- When the SessionCloseRQ Sabre API call is successfully consumed, all of the internal resources held by the connection/individual session are released, and the current quantity of active Sabre API sessions is decremented. The Sabre session becomes inactive and is returned to the subscriber’s session pool.

The connection manager stores the connection IDs and other connection information in the connection pool in a separate database or file. The use of a connection pool creates persistent connections and allows for reuse of connections as needed. When a workflow is complete, the client returns the connection ID to the connection pool, requesting a connection again when the need to send a workflow arises. This can be any free and available connection in the pool.

When calling SOAP-based TPF Connector Sabre APIs, which again, obtain their content from the Sabre host system, the client or connection manager has the responsibility of clearing the data in the Sabre work area/AAA that lingers from a previous session.

**When to Use Connection Pooling**

When there is a need for multiple clients, and the quantity of connections needed exceeds the quantity of clients available, this form of management is recommended. For a steady volume of 1 to 2 transactions per second, this technique is suitable. If the client’s business process needs multithreaded processes, it will need multiple, open connections. That is the only way to send simultaneous API calls.

**Advantages and Disadvantages**

Advantages of a connection pool are the ability to have multiple clients and make simultaneous Sabre APIs calls, while reducing the overhead of excessive requests to open and close connections. This saves time and resources by reusing connections instead of creating them every time the client application needs to retrieve travel content. The disadvantages are that additional hardware is required for the connection manager, and the architecture is not as simple or inexpensive to implement and maintain as the basic connection solution.

**Connection Managers**

A connection manager, the most complex solution, is also the most reliable architecture. The connection manager includes the following:

- A strategy for connection management
- The opening and closing of connections
- A connection pool
• The storage of connection IDs and updates about the status
• Load balancing
• Failover and connection recovery

A connection pool is a component of a connection manager. The connection manager opens and maintains multiple, concurrent connections, and persists the connection IDs, enabling multiple clients to request open connections and reuse them. A connection manager goes beyond connection pooling by using load balancing. The design can eliminate points of failure by adding redundancy and storing the connection IDs on a separate box so that they can be recovered, making failover automatic and recovery possible. An example of a simple implementation with some built-in redundancy is shown in Figure 9.

![Connection Manager Architecture with Limited Redundancy](image)

**Figure 9. Connection Manager Architecture with Limited Redundancy**

As shown in Figure 9, multiple clients are routed through a load balancer to the connection manager to request connections. The connection manager is housed on two boxes, eliminating a point of failure. The connection information is also stored separately so that the connection IDs can be recovered in the event of failure.

The implementation of a connection manager helps ensure that an adequate quantity of available connections is available when needed, without over-allocating your resources, that is, the quantity of Sabre APIs connections and Sabre sessions in your session pool.

**When to Use This**

If the client application’s environment is high volume, implementation of a connection manager with a level of redundancy needed is essential. If the client application cannot afford to have down time, a solution with full redundancy is recommended. Multiple simultaneous connections are also needed for multi-threaded processes.

**Advantages and Disadvantages**
This solution has the highest rate of reliability, automated failover, and fast session recovery. If the connection manager is fully redundant, it has no single point of failure, and connections are used efficiently, saving time and resources while eliminating overhead. This architecture is also highly scalable.

To effectively consume Sabre APIs, efficient management of connections is essential. Efficient connection management has the following benefits:

- Accelerates or expedites response times
- Minimizes errors
- Facilitates recovery from failures on the consumer or business application side

In addition to being complex to design and implement, other disadvantages are greater cost and the need for additional hardware and systems administration.

**Responsibilities and Duties of a Connection Manager**

The duties and responsibilities of a connection manager are explained as follows.

- **Define a specific and configurable quantity of open connections**
  This pool of open connections is designed to grow or shrink to a predefined threshold to accommodate the volume of traffic requesting connections from the pool.

  The number of connections also depends on the subscriber's TAM pool size. If the client has multiple IPCCs, each IPCC is allocated a quantity of session in its specific session pool. (In legacy systems, a session pool was referred to as a TAM pool.)

  When the connection manager opens a connection, not only is one of the connections being used, but one of the Sabre sessions in the session pool is also allocated and in use.

  The quantity of TAs available is based on information provided to the Sabre account representative up front, and is defined in the Sabre APIs contract. This information is used by capacity planning to determine the required allocation of Sabre TAs. (If additional resources are justified, please contact your Sabre account representative.)

- **Create connections to Sabre APIs**
  The connection manager begins its business logic by initializing the connection pool. This entails the creation of the predetermined and configurable number of connections by invoking the SessionCreateRQ Sabre API.

- **Throttle the quantity of open connections**
  The connection manager ensures the availability of a minimum number of Sabre API connections during the lifetime of the application.

  The connection manager is responsible for throttling the number of open connections to accommodate fluctuations in traffic volumes. The connection
manager adjusts the quantity of connections during the day to handle peak and low traffic volumes.

This throttle should be distributed across the different instances of a client application, if they do not use a common pool between the application instances.

If the connection manager determines that fewer live connections are needed, the connection manager closes some connections. If more connections are needed, the connection manager opens more. At any point, the minimal threshold of connections should not be exceeded.

• The connection manager is a proxy between the client and the network instead of a separate repository of connection information.

The job of the connection manager is to monitor activity and refresh sessions efficiently, therefore, transactions must pass through the connection manager.

• Manage the connections by storing them in a pool

  • The connection manager stores and tracks the connection information for each of the live connections.

    The connection information can be stored in a centralized database, memory, or another form of storage during application run-time. Basic connection information to store includes the security token, conversation ID, time stamp, connection status, that is, whether the connection is free or in use, and client ID, in other words, which client is using it.

    The connection manager caches and stores the connection information, and updates the information with a new time stamp and the status.

    All clients must be able to obtain the connections repeatedly with a given Web services connection.

  • The connection manager keeps connections alive, ensuring that the minimal quantity of Sabre APIs connections is available for use when needed. The connection manager must know the time-out value assigned to the security credentials used to open the connections. To prevent the connections from timing out, the connection manager refreshes the connections by sending the OTA_PingRQ Sabre API call.

• Destroy connections

The connection manager terminates connections when the threshold for low volume traffic is reached, the connection is no longer usable, or when connections need to be cleaned up. The connection manager closes the connections by invoking the SessionCloseRQ Sabre API call. Remember that this Sabre API also terminates the allocated Sabre session, and returns the Sabre session to the session pool. All of data in the Sabre work area/AAA is discarded.

• Clean up connections

The connection manager cleans up all live connections before the application is closed, shut down, or restarted. This makes the connections available to
other instances of the client that use the same security credentials.

The connection manager uses the SessionCloseRQ Sabre API call to terminate the connections. The connection manager also closes connections on a regular basis to refresh the pool and reinitialize it. Sabre APIs connections are no longer usable when they time out; they have to be created again.

- **Clean up Sabre sessions**
  
  When a client requests a connection from the pool, the connection manager clears the content in the Sabre work area/AAA before giving the connection ID to the client. This relieves the client of clearing the work area before sending a workflow. Some workflows may be designed to clear the work area after the last transaction in a workflow. Doing this depends on the applications’ business process logic.

- **Handle connection-related errors**
  
  The connection manager handles exceptions and time-outs that occur during the life cycle of all Sabre APIs connections. These errors can be API-specific or connection-specific. Depending on the error received, the result may be the termination of the current connection and the creation of a new one.

**Connection Manager Implementation**

To effectively manage SOAP-based Sabre API connections, basic connection information must be stored to enable clients to retrieve connections from a pool as needed, and to let the connection manager track the connections and keep them alive. Each entry inside the pool has the format shown in Figure 10.

### Storage of Connection Information

<table>
<thead>
<tr>
<th>Security Token</th>
<th>Conversation ID</th>
<th>Time Stamp</th>
<th>Connection Status</th>
<th>Client ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>This field…</td>
<td>Is used as follows…</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security Token</td>
<td>The security token returned with the creation of the connection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time Stamp</td>
<td>A time and date value that determines when the connection was last used. It also enables the Status field to be updated when the connection requires validation.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conversation ID</td>
<td>The conversation ID used to create the connection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This field...</td>
<td>Is used as follows...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connection Status</td>
<td>A value showing whether a connection is free or in use</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client ID</td>
<td>Identifies the client that is using the connection to associate the client instance with a particular connection ID in the pool</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When the box with the connection pool is started and the pool is opened, connection manager sends the SOAP-based SessionCreateRQ Sabre API calls the number of times equal to the quantity of connections it is configured to initialize.

Once the connections have been created, the connection manager is ready to handle requests from the clients in need of connections. All Sabre APIs requests must obtain a valid connection from the pool. This can be implemented in the connection manager as follows:

1. A client instance requests an available or free connection ID from the connection pool.
2. The connection manager sets the connection ID status flag to “in use.”
3. The connection manager sends the SOAP-based IgnoreTransactionLLSRQ Sabre API call to clear the Sabre work area/AAA of content lingering from previous use of the connection ID.
4. The connection manager provides the connection ID to the client.
5. The client sends messages representing the travel workflow using the connection ID.
6. When the client is done and has stored content it wants, the client returns the connection ID to the pool.
7. The connection manager sets the connection ID status flag to “available” or “free.”
8. The connection manager updates the time stamp of the connection ID.

During some configurable interval, the connection manager inspects the status indicator of all connections in the pool. All free connections with a time stamp older than the time-out value are either revalidated or closed. This decision is made on the basis of the current connection pool size.

The recommended keep-alive value should be less than the connection time-out value. The connection keep alive is used only to maintain a minimum number of connections in the pool. If the application has low traffic or use, it is advisable to close the connections until the minimum threshold limit in the pool is reached. However, if the application is running with less than the minimum threshold, keeping connections alive using the OTA_PingRQ Sabre API call is recommended.

As part of its clean-up activities, the connection manager must terminate all active connections in the pool. This can be a fixed, automated process or a manual maintenance activity. Even when the client is restarted or stopped, it is necessary to invoke the
SessionCloseRQ Sabre API call. This makes all of the connections in the pool available to the client when the client starts up.

All system-related errors and time-outs result in the closing of the connection and the creation of a new connection. This minimizes the number of orphan connections in the client.

**Session Recovery and Failover**

To enable session recovery and failover, the connection manager stores the connection ID with the corresponding client ID on a different machine from the connection manager and connection pool. To recover connections and active Sabre sessions, the client is restarted with the connection IDs that are stored on the other machine.

To eliminate points of failure, the system architecture should be redundant. Clients can choose which components to duplicate, or can replicate all components, as shown in Figure 11. This enables failover, load balancing, and recovery. Business needs dictate how clients design their architecture.

![Figure 11. Connection Manager Architecture with Full Redundancy](image-url)
Implementation Scenarios

Some sample scenarios are presented in this topic.

Scenario 1

When the machine with the connection pool is started and the connection pool is opened, the connection manager sends the SessionCreateRQ Sabre API calls the number of times equal to the quantity of connections it is configured to initialize. Let us say this quantity is 20. The connection manager stores the connection IDs of the open connections that are in the pool, monitoring use of the connections. When the threshold of connections in use is reached (say the threshold is 16), the connection manager opens 10 more connections. Sixteen connections is close to the threshold of 20, and to avoid running out of open connections, the connection manager is configured to open 10 more. When 30 connections are open, and only 11 are in use, the connection manager is configured to close 10 connections. 30 are open, 11 in use, and 19 are unused. Because 11 are fewer than the threshold of 16 open connections, there is no need to have 19 unused connections, therefore 10 are closed.

Scenario 2

This scenario has a client whose IPCC has been allocated a pool of 100 sessions. A 15 minute time-out value has been assigned to the connections and sessions.

At any point, the connection manager wants to maintain 10 connections in the pool. During times when traffic is very low, the client requires a maximum of 20 sessions. This is the low threshold value for the connection pool. During times of high traffic, the client can increase the pool size on demand as traffic increases.

Because the time-out is 15 minutes, the client refreshes or clears connections in the pool every 13 minutes. If there are less than 20 unused connections, the client refreshes the connections by sending the OTA_PingRQ Sabre API call. If the pool has more than 20 unused connections, the client calls the SessionCloseRQ Sabre API call until the threshold of 20 unused connections is reached.

When the client shuts down, it closes all connections in the pool.

The pool size has a maximum value of less than 100 connections. This minimizes the errors the client receives about unavailable resources from the Sabre APIs infrastructure.

Sabre session creation and removal must be minimized by the reuse of connections. Careful study of the scenarios depicted on the connection manager sequence diagram in Figure 12 shows that several Sabre APIs were invoked before the connection was returned to the pool. The connection manager will guarantee that the connection remains active between Sabre session retrieval and removal calls.
Figure 12. Sequence Diagram for a Connection Manager
Chapter 5: Business and Application Logic

Chapter five contains topics about travel workflows and the implementation of business and application logic in the client.

Maintaining Session State

In addition to designing clients to manage connections, application developers must include business logic to obtain the content desired, and manipulate the business application that provides the content by way of requests.

The Sabre system has some functionality that is stateful and other functionality that is stateless. The AAA (referred to as the Sabre work area throughout this document) is designed for state maintenance in the Sabre host system. The Sabre work area/AAA provides shopping cart functionality. The content retrieved via a Sabre format or a related SOAP-based TPF Connector Sabre API is stored in the work area until it is specifically cleared out, closed, or a time out occurs.

A Sabre session has a LNIATA or terminal address (TA). (The TA is referred to as a Sabre session throughout this document). The user IDs of SOAP-based TPF Connector Sabre APIs subscribers require Sabre host access, therefore, a Sabre session is allocated when the connection to Sabre APIs is authorized.

When a client connects to the SOAP-based Sabre APIs infrastructure a Sabre session is allocated, the Sabre work area/AAA is also initialized. This lets the client talk to the work area.

While all SOAP-based Sabre APIs are stateless, many of the functions associated with the SOAP-based Sabre APIs are stateful. The term stateful, according to the Webopedia Web site, is “the last-known or current status of an application or a process. The terms maintaining state and/or managing state refer to keeping track of the condition of the process.”

A SOAP-based Sabre APIs session that sends and receives any of the stateful Sabre API functions can maintain the last-known or current content in the Sabre work area/AAA.

A client can consume SOAP-based TPF Connector Sabre APIs in a stateless or stateful way,
depending on the specific SOAP-based TPF Connector Sabre APIs being consumed. It is
the client that decides whether to use content retrieved from a previous SOAP-based Sabre
API call or not. The design of the client and the workflow, and the sequencing of the SOAP-
based Sabre API calls let the client retrieve content from previous services.

Take the SOAP-based TraveltineraryReadLLSRQ Sabre API call as an example. The
command upon which this SOAP-based Sabre API call is based is a stateless Sabre system
command. This SOAP-based Sabre API call retrieves a passenger name record, i.e a PNR,
and places it in the work area.

An example of a stateful function in the Sabre system is the request to search for and display
air availability. The Sabre system remembers the flights that it displays via this Sabre format.
The SOAP-based TPF Connector Sabre API call that requests air availability,
OTA_AirAvailLLSRQ, is stateless, but a client can use other Sabre APIs to obtain further
information about selected flights placed in the work area by the previous SOAP-based
OTA_AirAvailLLSRQ Sabre API call. The client references the flights it wants in another
SOAP-based Sabre API call to obtain fare rules, search for lower fares, etc. It is the client
that is using the Sabre APIs in a stateful way.

The Sabre system tracks and maintains the content or state of the Sabre work area/AAA
when proper techniques are implemented.

The proper techniques for maintaining state in a Sabre session are as follows.

• Obtain the conversation ID and security token associated with the Sabre session.
  If this is a new connection, extract the wsse:Security node with the
  BinarySecurityToken from the SessionCreateRS response message. In both
cases, include the conversation ID and security token in all requests using the
  session. Sending the same conversation ID and security token with every
  request message in the session maintains state in the Sabre work area/AAA.

• When a client reuses a connection and its associated Sabre session, it must
  ensure the work area is cleared before sending a new workflow. A connection
  manager can also do this.

• When finished with a workflow, the client stores the content by invoking the SOAP-
  based EndTransactionLLSRQ Sabre API call.

When SOAP-based Sabre APIs representing stateful functions are called, the content from
all requests in a specific Sabre session is stored in the Sabre work area/AAA. The Sabre
work area/AAA can be thought of as the session state buffer. The content in this buffer can
be displayed, created, updated, and removed in a single Sabre session. If clients have
shopping cart functionality, the shopping cart reflects the content in the Sabre work
area/AAA. The client can parse content from each of the responses, and again, before it
ends the transaction.

The sequencing of the messages, referred to as orchestration, is especially important for
workflows that use stateful functions. This is because stateful functions can create and
maintain content in the Sabre work area/AAA. Sending a message in the wrong sequence
can overwrite the content in this work area. Application developers must be aware of the
content that is being created and stored there at all times. Ending a Sabre session properly
with the SOAP-based EndTransactionLLSRQ Sabre API call saves the content in the Sabre system and records it in the PNR. When client applications start their workflow by retrieving and reusing a connection from the pool, the content from the previously-used session associated with the connection remains in the Sabre work area/AAA. Therefore, application developers must design their client to clear the Sabre work area/AAA before sending the messages that represent a new workflow.

SOAP-based Sabre APIs functions are either stateless or stateful. The effect of each Sabre API call on state in a specific Sabre session is dependent on the Sabre API call itself. Some SOAP-based Sabre API calls only require a valid security token, while other SOAP-based Sabre API calls depend on the content placed in the Sabre work area/AAA by other Sabre API calls so they can perform their functions.

The SOAP-based Sabre APIs functions that are stateless can perform their functions independently of other Sabre APIs by sending a valid security token. The responses of stateless functions do not have references to content in other responses, for example, when the SOAP-based OTA_AirAvailLLSRQ Sabre API call is sent, the Sabre system processes it and returns a response. The SOAP-based OTA_AirAvailLLSRQ Sabre API call does not depend on any content placed in the Sabre work area/AAA by other Sabre APIs, but it leaves content in the work area that can affect subsequent commands, for example, 1*.

The descriptions of SOAP-based Sabre APIs located on the Sabre Dev Studio note when calls depend on content retrieved from a previous SOAP-based Sabre API call.

Stateful functions depend on content that is placed in the Sabre work area/AAA from responses to other SOAP-based Sabre API calls. To complete a transaction, additional SOAP-based Sabre API calls may need to be sent after a particular SOAP-based Sabre API call.

The modification of a PNR is presented as an example:

1. Assuming a PNR exists, the client first reads or displays it in the Sabre work area/AAA. The SOAP-based TravellItineraryReadLLSRQ Sabre API call request accomplishes this by loading the content from the PNR into the active Sabre work area/AAA.

2. Next, the client sends the SOAP-based TravellItineraryModifyInfoLLSRQ Sabre API call with updated content. The SOAP-based TravellItineraryModifyInfoLLSRQ Sabre API call is a stateful function, and depends on the SOAP-based TravellItineraryReadLLSRQ Sabre API call to load the content into the work area. When the content is loaded, the SOAP-based TravellItineraryModifyInfoLLSRQ Sabre API call can modify the content. It parses or extracts any data it needs from the response.

3. Finally, the client sends the SOAP-based EndTransactionLLSRQ Sabre API call to complete the PNR, and to receive and end the PNR record. This stores the updated record in the Sabre system.

Parsing and Storing Content

To store the transactions in the Sabre work area/AAA that occur in a given workflow, the
client application needs to send the SOAP-based EndTransactionLLSRQ Sabre API call. This stores the changes in the PNR and assigns a record locator. If needed, the client application can also parse and store other information it receives in any of the responses.

**Clearing Content in the Sabre Work Area/AAA**

When a connection manager is implemented in a client, SOAP-based Sabre APIs connections and the Sabre sessions that are allocated with the connections are reused. Multiple business workflows can also be sent in a single session.

When the connection needs to be retrieved from the pool for reuse, the client must ensure that the Sabre work area/AAA is cleared or empty before sending a new workflow. Clearing the Sabre work area/AAA eliminates the possibility of content from a previous Sabre session remaining and becoming intermingled with content in the Sabre session that follows.

After retrieving a connection from the connection pool, the client application can clear the Sabre work area/AAA by invoking the SOAP-based IgnoreTransactionLLSRQ Sabre API call.

This flow assumes that a new SOAP-based Sabre APIs connection has already been created and is in the connection pool. The client retrieves an existing connection from the pool, along with the conversation ID and security token. The client clears the Sabre work area/AAA with the SOAP-based IgnoreTransactionLLSRQ Sabre API call. A business workflow that consists of one or more travel-based Sabre APIs is sent. At the end of this workflow, the client sends the SOAP-based EndTransactionLLSRQ Sabre API call to save the transaction and record locator in the Sabre system. The client then returns the session along with the conversation ID and security token to the connection pool for reuse.

When another connection is needed, the client retrieves one from the pool. The client again sends the SOAP-based IgnoreTransactionLLSRQ Sabre API call to clear the content from the previous workflow in the Sabre work area/AAA.

The client sends another business workflow. At the end of the workflow, the application sends the SOAP-based EndTransactionLLSRQ Sabre API call to save the transaction and record locator in the Sabre system.

As long as connections are needed, the client continues to retrieve existing sessions from the pool and send workflows, clearing the Sabre work area/AAA before sending each new workflow. If the traffic volume is low and it is necessary to close some existing connections, the client sends the SOAP-based SessionCloseRQ Sabre API call to close the unneeded connections. The SOAP-based SessionCloseRQ Sabre API call releases the Sabre session and connection associated with the conversation ID and security token, invalidates the security token, and clears the Sabre work area/AAA.

Regardless of the workflow and commands sent, all content in the Sabre work area/AAA is removed when one of the following occurs:

- The client logs out with the SOAP-based SessionCloseRQ Sabre API call
- The Sabre API connection and session time out
• The SOAP-based IgnoreTransactionLLSRQ Sabre API call is sent - This Sabre API call clears everything associated with a PNR, but leaves other content, such as availability displays.
SOAP-Based Sabre APIs Workflows

The flexible design of the SOAP-based Sabre APIs allows application developers to create travel workflows any way that they want. When designing client applications and travel workflows, the application developer needs to select the Sabre APIs whose payload messages represent the content that they want to request and retrieve, and determine their sending sequence. To determine which of the SOAP-based Sabre APIs to incorporate into a client, application developers should review the content in the XML payload requests and responses to decide which data elements to send, and which elements to parse and integrate into the client application. For a list of the content in the payloads, application developers can consult the request and response design documents located on the Sabre Dev Studio. One pair of request and response design documents and a description document is provided for every SOAP-based Sabre API, and can be obtained via the Sabre Dev Studio.

There are choices for sending workflows with SOAP-based Sabre APIs, and the needs of the business determine which are most suitable. Some of the ways that can be used to manage workflows are as follows:

- Open a new connection to send a single or multiple workflows in a Sabre session, and then close the connection. This is the basic connection strategy.

- Reuse connections to send workflows using more than one client. Each client reuses the connections and sessions. This strategy uses connection pools and a connection manager.

A number of strategies must be considered with this approach:

- Clearing the Sabre work area/AAA after the completion of each workflow
- Clearing the Sabre work area/AAA at the beginning of the workflow
- Clearing the Sabre work area/AAA at the beginning of the workflow, retaining the content in the Sabre work area/AAA, and sending another workflow using the same Sabre session

Note that clearing the work area after completing each workflow is not as reliable a way to clear the content as clearing the work area at the beginning of client workflows.

Regardless of how application developers implement workflows, when the client application is finished with each workflow, it can save or ignore the transaction and return the connection to the connection pool for reuse. For complete information about SOAP-based Sabre APIs connection strategies and connection pools, please refer to the section of this document titled, "Connection Management."

Single Workflow Using a Single Sabre Session

When a single session contains more than one workflow, application developers must ensure that the work area is clear before sending the next workflow. When the workflow is complete, the client application can save the content in the Sabre system by ending the transaction with
the SOAP-based EndTransactionLLSRQ Sabre API call or can clear the Sabre work area/AAA with the SOAP-based IgnoreTransactionLLSRQ Sabre API call.

An example of a simple travel workflow that uses one SOAP-based Sabre API call is one that leverages the SOAP-based OTA_AirFlifoLLSRQ Sabre API call. In this workflow, the SOAP-based OTA_AirFlifoLLSRQ/RS API calls are exchanged to retrieve information about a specific flight and display the results. In this example, the client does not store any transactions or state in the Sabre system when it consumes the API. The client gathers the content it needs, which is the latest flight information, and provides it to an end consumer or other process.

Multiple Workflows Re-Using a Sabre Connection

To send more than one workflow with a single SOAP-based Sabre APIs session, the client application must first obtain a connection from the connection pool, followed by the exchange of messages representing the travel workflow. The client application completes the workflow by exchanging messages that store the transaction in the Sabre system.

The next choice is to send messages that either clear the work area or represent another travel workflow that uses some content remaining in the work area from the previous workflow. The needs of the business and end users dictate how to design these workflows, and whether it is necessary to clear the work area or retain the content before sending subsequent workflows that use the same Sabre session, before returning it to the pool for reuse.

When finished with the workflows, the client application can save the transaction in the Sabre system and return the connection to the connection pool for reuse. This automatically frees up the SOAP-based Sabre session and returns it to your session pool for reuse. Remember that a SOAP-based Sabre APIs connection and a Sabre session are synchronized. They are allocated and released simultaneously. One caveat is that the content remains in the Sabre work area/AAA after a Sabre session is released, until the connection is closed or times out.

Client applications can repeat this process until they want to terminate the SOAP-based Sabre API connection or the connection is no longer usable.

Single and Multi-threaded Workflows

The client application can also pass the conversation ID and security token among threads, processes, and machines. For example, by replicating and managing connections correctly, you can use a set of machines to provide failover features to a system.

**Note:** Do not share the connection, Sabre session, security token, or conversation ID among multiple threads.

**Single threads**

To minimize the quantity of Sabre sessions used in the session pool in a single process, clients must pass the same conversation ID and security token for the connection they are using in all messages in the thread.
**Multithreaded processes**

For multithreaded workflows or processes, use a separate SOAP-based Sabre APIs connection, each with its own conversation ID and security token, in each thread. If a client is generating multiple, simultaneous requests, either by multiple end consumers or other means, use one connection for each of the requesters.

**Minimizing Scans**

Clients are billed for scan charges whenever their client makes requests against the production Sabre system.

This applies to the following URL:

- [https://webservices3.sabre.com/websvc](https://webservices3.sabre.com/websvc)

When clients consume SOAP-based Sabre APIs, three types of scan charges may be applied: basic, fare, and search. Basic type scans cost less than fare or search type scans.

Application developers may want to take into account the type of scans that are associated with each SOAP-based Sabre API call. This may help with client design and user interaction by limiting the more expensive searches and caching responses.

As an example, application developers may want to impose constraints on the types of searches end consumers can do and the quantity of searches they can perform. Letting end consumers search for all availability and fares without specific dates is more expensive than searches based on specific dates.

Developers may also want to cache responses whenever it makes sense, but they must also be aware of limitations on the data retrieved from the Sabre system, such as the length of time for which data is valid.

The SOAP-based Sabre APIs does not limit the flexibility when it comes to creating meaningful workflows for organizations and customers. However, there are efficient ways to build client applications and workflows to minimize scans, which helps to minimize costs.

Reusing connections to send workflows reduces scans because client applications do not send the SOAP-based SessionCreateRQ and SessionCloseRQ Sabre API calls between each workflow. Sending the IgnoreTransactionLLSRQ Sabre API call before beginning a new workflow may possibly incur fewer transactions than clearing the work area after every workflow. Reducing scans and designing workflows for efficiency not only reduces costs, but it also improves total client response times.
Chapter 6: SOAP-Based Sabre APIs

Environments

Chapter six describes the systems and environments that are available for developing clients, testing, and consuming the SOAP-based Sabre APIs.

We provide several environments for consuming the SOAP-based Sabre APIs for the following general purposes: client development, customer acceptance testing, and production. The following table and topics fully describe these environments.

Caution

When a client or solution books travel arrangements utilizing a URL that resolves to the back-end production system, the transactions are recorded in the live, production Sabre system, and real-time inventory is decremented. This applies to the following URL:

- https://webservices3.sabre.com/websvc

Please be sure to cancel any production environment bookings created for test purposes. If these bookings are not canceled, you and possibly your customers will be billed by suppliers or other vendors for all associated fees.

<table>
<thead>
<tr>
<th>Use and Availability</th>
<th>Development</th>
<th>Customer acceptance</th>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points to the Production Sabre APIs application and Sabre system</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Points to the TSTS test Sabre host and associated systems</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Feature</td>
<td>Test Result</td>
<td>Production Result</td>
<td>Test Result 2</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>-------------</td>
<td>------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Points to the CERT test Sabre host and associated systems</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Test PNRs must be cancelled</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Sabre scan charges apply</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Production Sabre sessions (TAs) are shared across multiple SOAP-based Sabre APIs environments</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>URL and targeted system is available 24 x 7 x 365</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Chapter 7: Common Error Responses/Corrective Actions

In some situations SOAP-based Sabre APIs may respond with an error. Errors are returned via the .../soap-env:Fault structure which contains two key elements .../faultcode and .../StackTrace. .../faultcode provides a coarse-grained error categorization, and .../StackTrace provides additional details. This pattern focuses on utilizing the .../StackTrace element to properly diagnose and respond to Sabre API-related error responses.

The following table contains a list of the most common error messages that a SOAP-based Sabre APIs customer is likely to encounter. Please note that this is not a comprehensive list of SOAP-based Sabre API-related errors. However, the errors omitted from this list are internal, system-related errors that generally cannot be generated or corrected through client action.

<table>
<thead>
<tr>
<th>Error String Included in .../StackTrace</th>
<th>Cause</th>
<th>Corrective Action</th>
</tr>
</thead>
</table>
| SC_SERVICE_UNAVAILABLE                   | Internal | 1. The client application needs to stop processing for at least 500 milliseconds before attempting to retry the message.  
2. If step 1 does not resolve the issue, please contact Web services support at webservices.support@sabre.com. |
| USG_AUTHENTICATION_FAILED                | Invalid credentials presented in the SOAP-based SessionCreateRQ API call | 1. The client application needs to stop processing and the application's credentials need to be checked and corrected.  
2. If step 1 does not resolve the issue, please contact Web services support at webservices.support@sabre.com for additional assistance. |
<p>| USG_AUTHENTICATION_NOT_BAD                | Credentials in the request rather than a Binary Security | 1. The client application needs to stop processing and the application's |</p>
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Issue Description</th>
<th>Resolution</th>
</tr>
</thead>
</table>
| ALLOWED    | Token             | 1. SOAP envelope needs to be checked and corrected.  
|            |                   | 2. If step 1 does not resolve the issue, please contact Web services support at webservices.support@sabre.com for additional assistance. |
| USG_AUTHORI   | Not permitted to access the requested service | 1. The client application needs to stop sending the service call that is generating this error.  
|ZATION_FAILE|                   | 2. Please contact Web services support at webservices.support@sabre.com for additional assistance. |
| USG_CONNECTOR_IS_BUSY | Internal limit of concurrent requests for a given service family has been reached | 1. The client application needs to stop processing for at least 500 milliseconds before attempting to retry the message.  
|            |                   | 2. If step 1 does not resolve the issue, please contact Web services support at webservices.support@sabre.com. |
| USG_CONVERSATION_ID_REQUIRED | eb:ConversationId element is missing or null in SOAP header | 1. The client application needs to stop processing and the application’s SOAP envelope needs to be checked and corrected.  
|            |                   | 2. If step 1 does not resolve the issue, please contact Web services support at webservices.support@sabre.com for additional assistance. |
| USG_COULD_NOT_COMPLETE_REQUEST | Internal | 1. The client application needs to stop processing for at least 500 milliseconds before attempting to retry the message.  
<p>|            |                   | 2. If step 1 does not resolve the issue, please contact Web services support at <a href="mailto:webservices.support@sabre.com">webservices.support@sabre.com</a>. |
| USG_IOP_OBJECT_NOT_EXIST | Internal | 1. The client application needs to stop processing for at least 500 milliseconds before |</p>
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Steps</th>
</tr>
</thead>
</table>
| USG_IIOPT_TRANSIENT_EXCEPTION | Internal                                          | 1. The client application needs to stop processing for at least 500 milliseconds before attempting to retry the message.  
2. If step 1 does not resolve the issue, please contact Web services support at webservices.support@sabre.com. |
| USG_INVALID_ACTION            | Incorrect API operation in the <eb:Action> node of the SOAP envelope | 1. The client application needs to stop processing and the application’s SOAP envelope needs to be checked and corrected.  
2. If step 1 does not resolve the issue, please contact Web services support at webservices.support@sabre.com for additional assistance. |
| USG_INVALID_EBXML             | Required ebXML element or attribute is missing or incorrectly formed in SOAP envelope | 1. The client application needs to stop processing and the application’s SOAP envelope needs to be checked and corrected.  
2. If step 1 does not resolve the issue, please contact Web services support at webservices.support@sabre.com for additional assistance. |
| USG_INVALID_SECURITY_TOKEN    | Session has been closed or has expired and the Binary Security Token is invalid | 1. The client application needs to stop utilizing this session, and open a new one.  
2. If step 1 does not resolve the issue, please contact Web services support at webservices.support@sabre.com for additional assistance. |
<p>| USG_INVALID_SESSION           | Occurs when client attempts to use a session in the | 1. The client application needs to stop processing, and the application’s workflow needs to be |</p>
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
<th>Steps</th>
</tr>
</thead>
</table>
| USG_IS_BUSY             | Internal limit of concurrent requests for some combination of PCC/IP/API name has been reached | 1. The client application needs to stop processing for at least 500 milliseconds before attempting to retry the message.  
2. If step 1 does not resolve the issue, please contact Web services support at webservices.support@sabre.com. |
| USG_NO_RESPONSE_FROM_JMSRECEIVER_IN_TIME | Internal                                                             | 1. The client application needs to stop processing for at least 500 milliseconds before attempting to retry the message.  
2. If step 1 does not resolve the issue, please contact Web services support at webservices.support@sabre.com. |
| USG_PASSWORD_CHANGE_REQUIRED | Password change is required                                               | 1. The client application needs to stop processing, and the client's CREATE agent needs to reset the application's password.  
2. If step 1 does not resolve the issue, please contact Web services support at webservices.support@sabre.com. |
| USG_PASSWORD_NOTFOUND   | <wsse:Password> node is missing or null in the SOAP-based SessionCreateRQ Sabre API call | 1. The client application needs to stop processing and the application's SOAP envelope needs to be checked and corrected.  
2. If step 1 does not resolve the issue, please contact Web services support at webservices.support@sabre.com for additional assistance. |
| USG_RESOURCE_UNAVAILABLE | Client has exhausted                                                   | 1. The client application needs to stop attempting                                                            |
| LE | available sessions (TAM pool) | to open new sessions and close unused, existing sessions.
2. If step 1 does not resolve the issue, please contact Web services support at webservices.support@sabre.com. |
| USG_SECURITY_ICE_ERROR | Internal | 1. The client application needs to stop processing for at least 500 milliseconds before attempting to retry the message.
2. If step 1 does not resolve the issue, please contact Web services support at webservices.support@sabre.com. |
| USG_SERVICE_IS_BUSY | Internal limit of concurrent requests for a given service has been reached | 1. The client application needs to stop processing for at least 500 milliseconds before attempting to retry the message.
2. If step 1 does not resolve the issue, please contact Web services support at webservices.support@sabre.com. |
| USG_SERVICE_PROVIDER_ERROR | Internal | 1. The client application needs to stop processing for at least 500 milliseconds before attempting to retry the message.
2. If step 1 does not resolve the issue, please contact Web services support at webservices.support@sabre.com. |
Errors Related to SOAP-Based Sabre API Versions

Request payloads must include a Version attribute in the correct format. If the request is successfully processed, the requested version number is returned with the document root element in the response payload. If this requirement is not accurately fulfilled, error messages are returned in the <Errors><Error> nodes as follows:

<table>
<thead>
<tr>
<th>Error</th>
<th>Cause</th>
<th>Corrective Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>errors.INVALID_VERSION</td>
<td>Returned if the requestor specifies an invalid version a specific SOAP-based Sabre API call</td>
<td>Send a valid version for the specific SOAP-based Sabre API call</td>
</tr>
<tr>
<td>errors.MISSING_VERSION</td>
<td>Returned if a version is not present</td>
<td>Include the following in the document root element of the payload:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The Version attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• A valid version number in the correct format</td>
</tr>
</tbody>
</table>
Appendix A: SOAP Envelope Field Size Quick Reference

All service providers support the fields shown in the following table:

For SOAP-based Sabre API-specific SOAP envelope values, please consult the documentation related to each SOAP-based Sabre API located on the Sabre Dev Studio.

<table>
<thead>
<tr>
<th>SOAP Field Name</th>
<th>Definition</th>
<th>Maximum Field Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>eb:ConversationId</td>
<td>Identifies the set of related messages that make up a connection and its associated session</td>
<td>255</td>
</tr>
<tr>
<td>eb:CPAId</td>
<td>Identifies the point of sale location participating in the connection</td>
<td>20</td>
</tr>
<tr>
<td>eb:MessageId</td>
<td>A globally unique identifier for each message</td>
<td>255</td>
</tr>
<tr>
<td>eb:RefToMessageId</td>
<td>RefToMessageId has a cardinality of zero or one, and when present, it MUST contain the MessageId value of an earlier ebXML message to which this message relates</td>
<td>255</td>
</tr>
<tr>
<td>eb:PartyId</td>
<td>Identifies the party that originated the message</td>
<td>255</td>
</tr>
<tr>
<td>eb:PartyId/@eb:type</td>
<td>Attribute type in the element &lt;eb:PartyId&gt;</td>
<td>255</td>
</tr>
<tr>
<td>eb:Service</td>
<td>Identifies the Sabre API call that acts on the message</td>
<td>128</td>
</tr>
<tr>
<td>eb:Service/@eb:type</td>
<td>Attribute type in the element &lt;eb:Service &gt;</td>
<td>255</td>
</tr>
<tr>
<td>eb:Timestamp</td>
<td>Represents the time that the message header was created conforming to a date-time</td>
<td>30</td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>eb:TimeToLive</td>
<td>The time, expressed as UTC, by which a message should be delivered. This is not supported. It is mutually exclusive with eb:Timeout.</td>
<td>25</td>
</tr>
<tr>
<td>eb:Timeout</td>
<td>The time-out value, expressed in seconds (must be less than 60 seconds) Applies to SOAP-based TPF Connector Sabre APIs. It is mutually exclusive with eb:TimeToLive.</td>
<td>6</td>
</tr>
<tr>
<td>eb:Action</td>
<td>Identifies the action that acts on the Sabre API call</td>
<td>48</td>
</tr>
<tr>
<td>wsse:Username</td>
<td>Specifies a username</td>
<td>20</td>
</tr>
<tr>
<td>wsse:Password</td>
<td>Specifies the password associated with the particular username</td>
<td>30</td>
</tr>
<tr>
<td>wsse:NewPassword</td>
<td>NewPassword is not currently supported, but upon implementation it will allow for passing a new password into the system</td>
<td>30</td>
</tr>
<tr>
<td>Organization</td>
<td>A Sabre extension that specifies the point of sale location associated with the particular username and password</td>
<td>20</td>
</tr>
<tr>
<td>Domain</td>
<td>A Sabre extension that specifies a domain location (Sabre partition) that is associated with the particular username, password, and organization</td>
<td>20</td>
</tr>
<tr>
<td>wsse:BinarySecurityToken</td>
<td>The security token that is used in conjunction with the eb:ConversationId to allow messages to be exchanged</td>
<td>200</td>
</tr>
</tbody>
</table>
Appendix B: Identifying Documents for SOAP-Based TPF Connector Sabre APIs

Appendix B describes the naming pattern for the URLs that point to the WSDL documents of SOAP-based TPF Connector Sabre APIs, and how to display WSDL and schemas on a URL using the naming pattern.

Each of the SOAP-based Sabre APIs have their own set of WSDL, schema, and design documents.

All documents and tools, such as the Java test client and release notes, are available on the Sabre Dev Studio.

After logging in, customers can search for a Sabre API and select the version that they are interested in learning about. Customers can view the documents in a browser or download them. To obtain the URL for the WSDL and schema documents, display them in a browser.

The WSDL and schema documents are also available by directly accessing them via a URL instead of logging into the Sabre Dev Studio. To access them directly, outside the Sabre Dev Studio, customers must become familiar with the URL and file naming patterns of the documents.
SOAP-based TPF Connector WSDL and Schema URLs

It is important to be able to identify the artifacts that correspond to the SOAP-based Sabre API version that customers are consuming. This enables customers to discover, consume, and troubleshoot the SOAP-based Sabre APIs.

The file names of the schema and WSDL documents are part of the URL where these files reside. When customers search for and select a Sabre API on the Sabre Dev Studio Web site, they can also display the WSDL and schema documents. They appear in a browser from which you can view or download them.

As stated previously, the three-part version number is applied to file names of upgraded WSDL and schema documents, and consequently, to the URLs. Most of the URLs for the XML schema and WSDL document set conform to the pattern illustrated below.

URL Pattern:

- http://webservices.sabre.com/wsdl/tpfc/OTA_AirPriceLLS2.0.0RQ.wsdl

In this example:

- “http://webservices.sabre.com” represents the root URL.
- “/wsdl” indicates the directory where the WSDL and schema files are stored.
- “/tpfc/” represents the business application utilized to fulfill the request.
- “OTA_AirPriceLLS2.0.0RQ.wsdl” represents the file name.

Example SOAP-Based Sabre APIs PROD URLs:

- http://webservices.sabre.com/wsdl/tpfc/OTA_AirPriceLLS2.0.0RQ.wsdl
- http://webservices.sabre.com/wsdl/tpfc/OTA_AirPriceLLS2.0.0RQ.xsd
- http://webservices.sabre.com/wsdl/tpfc/OTA_AirPriceLLS2.0.0RQRS.xsd
- http://webservices.sabre.com/wsdl/tpfc/OTA_AirPriceLLS2.0.0RS.xsd

Example SOAP-Based Sabre APIs CERT URLs:

- http://wsdl-crt.cert.sabre.com/wsdl/tpfc/OTA_AirPriceLLS2.0.0RQ.wsdl
- http://wsdl-crt.cert.sabre.com/wsdl/tpfc/OTA_AirPriceLLS2.0.0RQ.xsd
- http://wsdl-crt.cert.sabre.com/wsdl/tpfc/OTA_AirPriceLLS2.0.0RQRS.xsd
Finding WSDL and Schema Documents via a URL

Once customers become familiar with the file naming patterns and URIs for the WSDL and schema documents, they can access the URIs directly from a browser window.

To view or download the documents from a URI, customers need to enter the complete URL of the document they want to view in a browser. They can copy and paste the URI into their development tools, or download the schemas to validate payloads locally.

The following example shows customers how to display the schema and WSDL documents for the 2.0.0 version of the OTA_AirAvailLLSRQ Sabre API call.

1. Enter the base URL, along with /wsdl. The URL should look like the following example: http://webservices.sabre.com/wsdl/

2. Add a slash character /, and append the abbreviation for the business application to the URL. It should look like the following example: http://webservices.sabre.com/wsdl/tpfc

3. Add a slash character /, and add the base action code of the service plus .wsdl to the URL. The URL should look like the following example: http://webservices.sabre.com/wsdl/tpfc/OTA_AirAvailLLS2.0.0RQ.wsdl

4. The URL is complete. Display the WSDL document associated with the URL in the preceding step.

5. In the WSDL document, look for the following line: <import namespace="http://webservices.sabre.com/sabreXML/2011/10" location="OTA_AirAvailLLS2.0.0RQRS.xsd"/>

6. Copy OTA_AirAvailLLSRQRS.xsd from this line and substitute it for OTA_AirAvailLLS2.0.0RQ.wsdl in the URL. It should look like the following example: http://webservices.sabre.com/wsdl/tpfc/OTA_AirAvailLLS2.0.0RQRS.xsd

7. Display the document associated with the URL in the preceding step.

8. The OTA_AirAvailRQRS.xsd schema has two lines that refer to the request and response schema files. Look for the lines that are shown below: <include schemaLocation="OTA_AirAvailLLS2.0.0RQ.xsd"/> <include schemaLocation="OTA_AirAvailLLS2.0.0RS.xsd"/>

9. Copy OTA_AirAvailLLS2.0.0RQRS.xsd from the appropriate include line. Replace OTA_AirAvailLLS2.0.0RQRS.xsd with OTA_AirAvailLLS2.0.0RQ.xsd in the URL. It should look like the following example: http://webservices.sabre.com/wsdl/tpfc/OTA_AirAvailLLS2.0.0RQ.xsd

10. Display the XML request schema associated with the URL in the preceding step.
11. Display the schema for the header and SOAP wrapper again by entering the following URL or by clicking the “Back” button on the browser:
   http://webservices.sabre.com/wsdl/tpfc/OTA_AirAvailLLS2.0.0RQRS.xsd

12. In the OTA_AirAvail2.0.0RQRS.xsd schema, look for the line shown below: <include schemaLocation="OTA_AirAvailLLS2.0.0RS.xsd"/>

13. Copy OTA_AirAvailLLSRS.xsd from this line. Replace OTA_AirAvailLLSRQRS.xsd with OTA_AirAvailLLSRS.xsd in the URL. It should look like the following example:
   http://webservices.sabre.com/wsdl/tpfc/OTA_AirAvailLLS2.0.0RS.xsd

14. Display the XML response schema associated with the URL in the preceding step.
Appendix C: Sample Travel Workflows

Appendix C describes several, common travel workflows.

Sample Air Workflow

The following air workflow is broken down into two paths, the shopping path, and the booking path.

During the shopping path the client application is shopping for flights on behalf of an end user.

There are several air shopping services available, i.e. the SOAP-based BargainFinderMaxRQ Sabre API call, or the SOAP-based OTA_AirLowFareSearchLLSRQ Sabre API call. These services contain many qualifiers for further filtering/tailoring the result set; however at a minimum the client simply needs to specify a market, a date, and a class of service. When these services respond they provide air itineraries in accordance with the options that were requested. The client application needs to allow the end user to select an itinerary, which the client application will then use on the booking path.

During the booking path the client application is taking the air itinerary that the end user selected during the shopping path, and combining it with several other pieces of passenger-related information in order to create a passenger name record (PNR).

It is oftentimes far simpler to add the passenger-related information into the PNR prior to selling the air itinerary. To add passenger-related information into the PNR client applications can leverage the SOAP-based PassengerDetailsRQ Sabre API call. The PassengerDetailsRQ Sabre API call allows client applications to add agency address information, customer numbers, email addresses, passenger names, passenger types, phone numbers, received from information, remarks, retention segments, SSRs, and ticket time limit-related information. The service also allows client applications to specify to queue place the record to a particular queue, or to simply end and retrieve the record.

At a minimum the client application needs to provide agency address information, passenger names, phone numbers, and ticket time limit-related information prior to selling an air itinerary. With this information in place the client application can then book the air itinerary.

To book an air itinerary client applications can leverage the SOAP-based EnhancedAirBookRQ Sabre API call.
The SOAP-based EnhancedAirBookRQ Sabre API call is used to book air itineraries, and it can also be used to price the air itineraries. To book an air itinerary clients simply need to specify the flight numbers, the airline codes, the dates/times, the classes of service, as well as the markets associated with the segments. All of this information is returned in the shopping service responses. To price the air itinerary clients simply need to specify the number of passengers, along with the pertinent passenger types.

Once this step is complete the client application simply needs to receive and end the record. The SOAP-based PassengerDetailsRQ Sabre API call can be used to receive and end the record.

Workflow summary:

1. Shop (BargainFinderMaxRQ, OTA_AirLowFareSearchLLSRQ)
2. Book (PassengerDetailsRQ, EnhancedAirBookRQ)
3. Finalize the record (PassengerDetailsRQ)

Complete details on each these SOAP-based Sabre APIs, as well as XML-based workflows are available on the Sabre Dev Studio.

**Sample Vehicle Workflow**

The following vehicle workflow is broken down into two paths, the shopping path, and the booking path.

During the shopping path the client application is shopping for vehicle reservations on behalf of an end user.

The SOAP-based OTA_VehAvailRateLLSRQ Sabre API call is used to search for vehicles/rates. This Sabre API call contains many qualifiers for further filtering/tailoring the result set; however at a minimum the client simply needs to specify a city code, the reservation dates/times, as well the number of vehicles to be reserved. When this Sabre API responds it provides vehicles/rates in accordance with the options that were requested.

To view more in depth rate-related information clients can utilize the SOAP-based VehRateRulesLLSRQ Sabre API call.

To view in depth rental location-related information clients can utilize the SOAP-based OTA_VehLocDetailLLSRQ Sabre API call.

The client application needs to allow the end user to select a vehicle/rate, which the client application will then use to fulfill the subsequent booking steps. At a minimum the client application needs to keep track of the vehicle rental chain code, the city code associated with the rental location, the reservation dates/times, and the vehicle type code, which are all returned in the SOAP-based OTA_VehAvailRateLLSRQ Sabre API response.

During the booking path the client application is taking the vehicle/rate that the end user selected during the shopping path, and combining it with several other pieces of passenger-related information in order to create a passenger name record (PNR).
When booking a vehicle the passenger-related information must be added into the PNR prior to selling the vehicle segment. Client applications must add names prior to attempting to book the vehicle segment. If this information is not present in the PNR, the client application will receive an error response when attempting to book the vehicle segment. To add passenger-related information into the PNR client applications can leverage the SOAP-based PassengerDetailsRQ Sabre API call. The SOAP-based PassengerDetailsRQ Sabre API call allows client applications to add agency address information, customer numbers, email addresses, passenger names, passenger types, phone numbers, received from information, remarks, retention segments, SSRs, and ticket time limit-related information. The service also allows client applications to specify to queue place the record to a particular queue, or to simply end and retrieve the record.

The SOAP-based OTA_VehResLLSRQ Sabre API call is used to book a vehicle reservation. To book a vehicle reservation, clients simply need to specify the rental city code, the reservation dates/times, the vehicle type, as well the number of cars to be reserved that were gathered from the SOAP-based OTA_VehAvailRateLLSRQ Sabre API response.

Once this step is complete the client application simply needs to receive and end the record. The SOAP-based PassengerDetailsRQ Sabre API call can be used to receive and end the record.

Workflow summary:

1. Shop (OTA_VehAvailRateLLSRQ, VehRateRulesLLSRQ, OTA_VehLocDetailLLSRQ)
2. Book (PassengerDetailsRQ, OTA_VehResLLSRQ)
3. Finalize the record (PassengerDetailsRQ)

Complete details on each these SOAP-based Sabre APIs, as well as XML-based workflows are available on the Sabre Dev Studio.

**Sample Hotel Workflow**

The following hotel workflow is broken down into two paths, the shopping path, and the booking path.

During the shopping path the client application is shopping for hotels on behalf of an end user.

The SOAP-based OTA_HotelAvailLLSRQ Sabre API call is used to search for hotels. This service contains many qualifiers for further filtering/tailoring the result set; however at a minimum the client simply needs to specify a city code, a date, and the number of guests. When this service responds it provides hotels in accordance with the options that were requested.

Oftentimes after a user has selected a hotel they look at the hotel's property description. The SOAP-based HotelPropertyDescriptionLLSRQ Sabre API call is used to display hotel property description-related information. This response also contains rate-related information.

To view more in depth rate-related information clients can utilize the SOAP-based HotelRateDescriptionLLSRQ Sabre API call.
The client application needs to allow the end user to select a hotel, which the client application will then use to fulfill the subsequent booking steps. At a minimum the client application needs to keep track of the hotel chain code, the hotel property number, the stay dates, the number of guests, and the hotel rate code.

During the booking path the client application is taking the hotel that the end user selected during the shopping path, and combining it with several other pieces of passenger-related information in order to create a passenger name record (PNR).

When booking a hotel the passenger-related information must be added into the PNR prior to selling the hotel segment. Client applications must add names, as well as agency address-related information prior to attempting to book hotel segments. If this information is not present in the PNR, the client application will receive an error response when attempting to book the hotel segment. To add passenger-related information into the PNR client applications can leverage the SOAP-based PassengerDetailsRQ Sabre API call. The SOAP-based PassengerDetailsRQ Sabre API call allows client applications to add agency address information, customer numbers, email addresses, passenger names, passenger types, phone numbers, received from information, remarks, retention segments, SSRs, and ticket time limit-related information. The service also allows client applications to specify to queue place the record to a particular queue, or to simply end and retrieve the record.

The SOAP-based OTA_HotelResLLSRQ Sabre API call is used to book a hotel reservation. To book a hotel, clients simply need to specify the hotel chain code, the hotel property number, the stay dates, the number of guests, and the hotel rate code.

Once this step is complete the client application simply needs to receive and end the record. The SOAP-based PassengerDetailsRQ Sabre API call can be used to receive and end the record.

Workflow summary:

1. Shop (OTA_HotelAvailLLSRQ, HotelPropertyDescriptionLLSRQ, HotelRateDescriptionLLSRQ)
2. Book (PassengerDetailsRQ, OTA_HotelResLLSRQ)
3. Finalize the record (PassengerDetailsRQ)

Complete details on each these SOAP-based Sabre APIs, as well as XML-based workflows are available on the Sabre Dev Studio.
Glossary

AAA

An abbreviation for Agent Assembly Area. See Sabre work area/AAA.

artifact

As it relates to SOAP-based Sabre APIs, an artifact is anything that assists in the discovery and use of a service. Some examples of artifacts for SOAP-based Sabre APIs include the Sabre XML WSDL and schema documents, and action codes. Each of the artifacts exists as a separate entity, and artifacts are not shared among SOAP-based Sabre APIs.

Compare metadata.

basic connection

Basic connection is the simplest approach for handling connecting to SOAP-based Sabre APIs. It is similar to a conversation. You open a connection with the SOAP-based SessionCreateRQ Sabre API call, next you exchange requests for travel content and receive the responses using SOAP-based TPF Connector or open systems-based Sabre APIs, then you close the connection with the SOAP-based SessionCloseRQ Sabre API call. The client to connection ratio is one-to-one—you have one client and one connection.

binary security token

This document uses the term security token See security token.
**connection**

A connection is an open channel to the SOAP-based Sabre APIs infrastructure. After a client or other process is authenticated and authorized, an open connection to Sabre APIs is successfully created, and at the same time, a SOAP-based Sabre APIs session is allocated.

A connection is not a client side shopping cart and it does not maintain state in the Sabre work area/AAA of the Sabre host system.

The distinction between the terms “connection” and “session” is semantic. A client requests a connection to the SOAP-based Sabre APIs infrastructure, and upon successful connection, a SOAP-based Sabre APIs session is created simultaneously with a business application or data center within Sabre. A connection is on the client side, and a session is on the Sabre side. The connection and session are synchronized.

**connection ID**

A connection ID consists of the security token and conversation ID returned to the requester in the SOAP-based SessionCreateRS Sabre API call SOAP response envelope. Its return means the connection to the SOAP-based Sabre APIs infrastructure is alive and a Sabre APIs session is allocated. The connection ID is required for all transactions with the SOAP-based Sabre APIs infrastructure that are using the connection.

**connection manager**

The practice of managing SOAP-based Sabre APIs connections to ensure that connections are available without over-allocating resources. The client is responsible for implementing a connection manager.

The connection manager is an implementation of a strategy for handling multiple, concurrent connections. It has several components, such as a connection pool, and manages many tasks. Some of the tasks include opening and maintaining connections, persisting the connection IDs, and refreshing the connections. A connection manager has thresholds defined for high and low volume traffic and tries to maintain the connections needed to accommodate the fluctuations in traffic volumes. It also provides failover and recovery.

**connection pool**

A connection pool is a repository of multiple open connections whose connections are maintained and available for clients who need to use them to request travel content and send travel workflows. From the perspective of Sabre APIs, the connections in the pool are open channels to the Sabre APIs infrastructure. With a connection pool, you have more open connections than clients. The connection pool is one component of a connection manager. See also connection manager.

**constraint**

Specifies the data type of an element or attribute, such as a string or integer, and whether the values for an element or attribute are restricted and required to be present.
**conversation**

A term of the W3C and ebXML. It is the exchange of messages among trading partners. A conversation is the same thing as a basic Sabre APIs connection. See basic connection.

**ebXML**

Electronic Business Using Extensible Markup Language. ebXML is an enabling technology sponsored by UN/CEFACT and OASIS, and the OpenTravel specifications are based on OASIS and UN/CEFACT.

**extension**

Any addition, such as an element or attribute, to the OpenTravel specifications. Extensions let organizations use proprietary content that is not present in the OpenTravel specifications so that they can exchange content with their trading partners.

Many Sabre XML schemas incorporate extensions, enabling SOAP-based Sabre APIs to use proprietary content in the Sabre system and other Sabre applications.

See also Sabre XML.

**GDS**

global distribution system. The Sabre system is a GDS.

See PSS and Sabre global distribution system (Sabre GDS).

**IPCC**

Internet Pseudo City Code

**metadata**

All data or information about a Sabre API. Metadata for Sabre APIs includes, but is not limited to, service implementation date and version, name of the service, and previously required orchestration.

Compare artifact.

**open systems-based Sabre APIs**

SOAP-based Sabre APIs that obtain their content from a variety of open systems within Sabre via direct connections to those systems. The open systems services are sometimes referred to as direct services.

**OpenTravel**

OpenTravel provides standards for the travel industry. OpenTravel specifications provide a common reference point that eliminates duplication of common data elements. It separates data and reduces it to the data element level, making it possible for two parties to communicate individual data elements. The parties decide whether to use specific data elements and how many times. Suppliers use this
standard and the TCP/IP infrastructure of the Internet to communicate with numerous other organizations.

**Passenger Name Record**

See PNR.

**PCC**

Pseudo City Code

**PNR**

Passenger Name Record. The record stored in the Sabre system that contains information related to a passenger's trip. It is identified with a unique record locator.

**PSS**

Passenger Service System

**Sabre global distribution system (Sabre GDS)**

This document uses the term Sabre system. See Sabre system.

**Sabre session**

A Sabre session is a specific type of session. It is associated with a LNIATA in native Sabre systems (also referred to as a PSS). The user IDs of Sabre system subscribers require and use LNIATAs or Terminal Addresses (TAs).

A Sabre session is a session that is established with the Sabre “host” system when a Sabre APIs connection is opened with a user ID that requires use of the Sabre system. A Sabre session, also known as a TA, is allocated from the user’s session pool and becomes active. The connection and session are synchronized, and therefore, both the connection and Sabre session remain active until either a time-out occurs or the connection is closed explicitly.

**Sabre system**

The Sabre global distribution system, or Sabre “host” system. This is the system that stores travel inventory and itineraries, and is the source of travel-related content for SOAP-based TPF Connector Sabre APIs as well as some other open systems and applications.

**Sabre APIs**

All APIs provided by Sabre. Under the umbrella of Sabre APIs are SOAP-based TPF Connector, open systems-based Sabre APIs, and session management-based Sabre APIs.
See also open systems Sabre APIs, session management Sabre APIs, and SOAP-based TPF Connector Sabre APIs.

**Sabre APIs infrastructure**

The combined components which provide connections, security, logging, and route incoming requests to the appropriate service provider's business application, and route the responses to the requester upon receipt from the service provider. One component of the infrastructure, the SOAP-based Sabre APIs gateway, provides a single point of access using a standard communication path, SOAP, and promotes a standard interface for access to services using XML.

External access to SOAP-based Sabre APIs is through the SOAP-based Sabre APIs infrastructure.

**Sabre APIs session**

A session that is allocated when a client makes a connection to the SOAP-based Sabre APIs infrastructure. The session is synchronized with the connection, which is on the client side. The type of session that is allocated depends on the security credentials used to open the connection. Users of SOAP-based TPF Connector Sabre APIs are allocated a Sabre session when they connect to Sabre APIs, while users of open systems may be allocated another type of session.

See also Sabre session and connection.

**Sabre work area/AAA**

Refers to the buffer in the Sabre system where content is retained while a Sabre “host” session is active. Other terms for the Sabre work area include Agent Assembly Area and AAA. This document uses the term Sabre work area/AAA.

The Sabre work area/AAA provides shopping cart functionality on the client side.

When SOAP-based TPF Connector Sabre APIs are called, the content from requests in a specific Sabre session is temporarily stored in this work area. The client application can use the content in the Sabre work area/AAA in a stateful or stateless way. Some SOAP-based TPF Connector Sabre APIs rely on content placed in the work area by previous service calls in the same Sabre session, while other services do not have these dependencies.

**Sabre XML**

XML messages used by SOAP-based Sabre APIs that are formatted to include the proprietary data, elements, and attributes of the Sabre system and other applications within Sabre. Some of these messages are based on OpenTravel specifications and other messages are not.

Many of these data elements are added into the messages as child elements of the TPA_Extensions element.

The WSDL documents for SOAP-based Sabre APIs also fall under the umbrella of Sabre XML. They are modified for compatibility with Sabre APIs and various frameworks for
developing and consuming SOAP-based Sabre APIs via WSDL.

See also extension.

**Sabre XML specification**

Sabre XML specifications encompass the following: Sabre XML request and response schema documents for all services, WSDL documents for Sabre XML, the content of the payloads, any constraints on data, and messages for managing SOAP-based Sabre APIs connections. See also Sabre XML.

**SDS**

Sabre Data Source.

**security token**

The binary security token that is returned to a client in the SOAP-based SessionCreateRS Sabre API SOAP response envelope in the wsse:BinarySecurityToken element. It is returned after a client creates a connection to the SOAP-based Sabre APIs infrastructure and has been authenticated and authorized.

**service**

A discrete unit of data or content that consists of business logic or host command input and output. It is exposed via a common access infrastructure.

**session management-based Sabre APIs**

SOAP-based Sabre APIs that are designed to connect to and disconnect from the Sabre APIs infrastructure. The session management Sabre APIs are SessionCreateRQ, SessionCloseRQ, and OTA_PingRQ. These messages are also part of the Sabre XML specifications.

**session pool**

User IDs that are tied to Sabre host sessions are allocated a finite quantity of Sabre sessions for their use with the Sabre system. This collection of Sabre host sessions is referred to as a session pool or a TAM pool.

The use of SOAP-based TPF Connector Sabre APIs requires a user ID that also use a Sabre host session. User IDs that use open systems Sabre APIs do not use TAs or TAM pools.

When your client or connection manager successfully connects to Sabre APIs with the SOAP-based SessionCreateRQ Sabre API call, one of the Sabre sessions in your TAM pool is allocated and active. The Sabre session is no longer available in the pool until the connection is closed or the session times out.

Compare with connection pool and see also TAM pool.

**SOAP**
Simple Object Access Protocol. A mechanism for transporting the data from one network to another.

**state and stateful**

The Sabre system is stateful. The Sabre work area/AAA is designed for state maintenance in the Sabre system. The content is remembered or stored in the work area until specifically cleared out.

A client can consume SOAP-based TPF Connector Sabre APIs in a stateless or stateful way, depending on which of those Web services it calls. The client decides whether to use content it has retrieved from a previous service or not.

**subscriber**

A travel organization that is a contracted customer of Sabre and Sabre APIs. Sabre subscribers include businesses or other entities such as travel agencies, online travel providers, travel suppliers (including airlines), and travel software development organizations which are involved with travel marketing and travel distribution. Sabre subscribers must have a valid Sabre access agreement to use Sabre APIs.

**TA**

Terminal Address. See also Sabre session.

**TAM**

Terminal Address Management

**TAM pool**

In the Sabre system, a pool of TAs is allocated to subscribers whose user IDs require Sabre host sessions. This pool of TAs is generally referred to as a session pool or a TAM pool. See also session pool.

**TPA_Extensions**

OpenTravel provides standards for the travel industry, and it also provides the TPA_Extensions element for adding elements that do not exist in its specifications. Elements that are added to the OpenTravel specifications are referred to as extensions. The use of extensions allows Sabre APIs and other Sabre systems/applications to exchange proprietary content that is not present in the OpenTravel specifications with its trading partners. Because SOAP-based Sabre APIs use XML schemas that have extensions, Sabre XML includes messages with these extensions.

See also Sabre XML.

**TPF Connector Sabre APIs**

The collection of SOAP-based Sabre APIs which obtain their content from the Sabre system or PSS via a TPF-based application. The SOAP-based TPF Connector Sabre APIs are fine-grained, and generally, one service is equivalent to one Sabre system command. They use a Sabre session and the Sabre work area/AAA.
The letters LLS appear in the names and action codes of SOAP-based TPF Connector Sabre APIs, for example, OTA_AirAvailLLSRQ, which distinguishes them from open systems Sabre APIs. Open systems-based Sabre APIs obtain their content from other business applications within Sabre.

**UN/EDIFACT**

United Nations Electronic Data Interchange for Administration, Commerce, and Transport. A travel message distribution protocol that has syntax rules for exchanging data.

This protocol is based on sentence structure. The two parties who are exchanging data must agree on the set of messages they plan to exchange in a specific transaction, for example, the availability of flights and seats at 2:00 on Friday between point A and point B. OpenTravel and ebXML specifications are sponsored by UN/EDIFACT.

**Web service**

A software system that uses XML to define the format and data in messages. The messages are sent over the Internet.

**WSDL**

Web Services Description Language.

**XML**

Extensible Markup Language.