Sabre® APIs: Connection Management Best Practices and Strategies, December 31, 2014

© 2003-2014 Sabre. 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
3150 Sabre Drive, Southlake, TX 76092
Tel: 682 605 1000
www.sabre.com
## Contents

About This Document .................................................................................................................. 5

Types of User IDs and Sessions .................................................................................................. 5

Related Documents ...................................................................................................................... 6

Managing Connections with *Sabre APIs* .................................................................................. 7

*Sabre APIs* Connections ......................................................................................................... 7

Connecting to *Sabre APIs* ......................................................................................................... 8

Closing Connections .................................................................................................................... 10

Relationship Between Connections and Sessions .......................................................................... 10

Sessions with SOAP-based TPF Connector *Sabre APIs* ............................................................ 11

Allocation of *Sabre APIs* Sessions ........................................................................................... 11

Shopping Cart Functionality and the *Sabre Work Area* ............................................................ 11

Release of *Sabre* Sessions ......................................................................................................... 12

Sessions with Open Systems *Sabre APIs* ............................................................................... 12

*Sabre APIs* Time-Outs .............................................................................................................. 13

Approaches for Handling Connectivity ...................................................................................... 13

Basic Connections .................................................................................................................... 13

When to Use Basic Connections ............................................................................................... 16

Advantages and Disadvantages ................................................................................................. 16
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection Pools</td>
<td>16</td>
</tr>
<tr>
<td>Connection Managers</td>
<td>20</td>
</tr>
<tr>
<td>When to Use This</td>
<td>21</td>
</tr>
<tr>
<td>Advantages and Disadvantages</td>
<td>22</td>
</tr>
<tr>
<td>Responsibilities and Duties of a Connection Manager</td>
<td>22</td>
</tr>
<tr>
<td>Connection Manager Implementation</td>
<td>24</td>
</tr>
<tr>
<td>Session Recovery and Failover</td>
<td>26</td>
</tr>
<tr>
<td>Single and Multithreaded Workflows</td>
<td>27</td>
</tr>
<tr>
<td>Implementation Scenarios</td>
<td>27</td>
</tr>
<tr>
<td>Scenario 1</td>
<td>28</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>28</td>
</tr>
<tr>
<td>Glossary</td>
<td>30</td>
</tr>
</tbody>
</table>
About This Document

This document is for software developers who create clients that connect to the *Sabre® APIs* infrastructure to access business applications or other systems within *Sabre*. Once connected to the infrastructure, the client calls one or more *Sabre APIs*.

The goal of this document is to give you, the developer, information that will help you design a client that uses best practices for connection management. The result is the ability to maintain the level of available connections that meet the needs of your business without over allocating or exceeding your resources.

Before presenting connection strategies, it is important to understand the different types of user IDs and security credentials used to connect to and consume *Sabre APIs*, and when a connection and session are created.

Types of User IDs and Sessions

Two types of *Sabre APIs* sessions may be allocated when a connection to the *Sabre APIs* infrastructure is created—connections that allocate a *Sabre®* “host” session and connections that do not allocate *Sabre* sessions. (A *Sabre* session is also known as a TA. This document uses the term *Sabre session*.) The type of user ID you have determines the type of session that is allocated when your client connects to the *Sabre APIs* infrastructure.

- If the user ID uses the *Sabre®* global distribution system (*Sabre® system*), it also uses *Sabre* sessions (remember that *Sabre* sessions are synonymous with TAs). When a client creates a *Sabre APIs* connection, a *Sabre* session is allocated simultaneously by the infrastructure for this type of user ID. Most user IDs fall into this category. For subscribers with this type of user ID, a *Sabre APIs* connection and a *Sabre* session are treated the same by the infrastructure. For example, when a *Sabre APIs* connection is alive, the *Sabre* session is allocated and in use. When a client returns the connection to the connection pool, the *Sabre* session is also
returned to the session pool or TAM pool. (This document uses the term “session pool” for TAM pool.)

- If the user ID and security credentials do not use the Sabre system or a Sabre session, a Sabre APIs connection is created without allocating a Sabre session or TA.

- One of the elements in your security credentials for client login to the Sabre APIs infrastructure is the wsse:Username element. Your client passes a user ID in this element when logging in.

- If you are unsure about the type of user ID you have, please contact your Sabre account representative.

Note: Whether or not your client uses a Sabre session, the best practices for connection management and connection pooling presented in this document are applicable.

Related Documents

The following documents provide additional information. They are available on Sabre Dev Studio at https://developer.sabre.com/.

- Sabre APIs Guide to Accessing and Using Services

Note: This developer's guide is required reading. It has complete details about the SOAP message formats and requirements for connecting to the Sabre APIs infrastructure and consuming Sabre APIs. It also discusses business logic for working with the Sabre system, the AAA, and SOAP-based TPF Connector Sabre APIs, in addition to other information.

- Documentation for the session management services, which includes WSDL, request, and response schemas, and a service description:

  - SessionCreateRQ Service
  - SessionCloseRQ Service
  - SessionValidateRQ Service
Managing Connections with Sabre APIs

This Chapter discusses best practices and strategies for Sabre APIs connections.

Sabre APIs Connections

Connections are open channels to the 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. At the same time, a Sabre APIs session is created.

The distinction between the terms “connection” and “session” is purely semantic. A client application requests a connection to the Sabre APIs infrastructure, and upon success, a 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 1. The time-out value for a connection and session are synchronized, occurring simultaneously.
A connection is *not* a client side shopping cart and it does *not* maintain state in the AAA of the *Sabre* host system. (The AAA is referred to as the *Sabre* work area in this document.)

### Connecting to *Sabre APIs*

There is one way to connect to *Sabre APIs*. The general steps are to send the Web service that requests a connection with your security credentials, conversation ID, and other required values, let the infrastructure authenticate and authorize your security credentials, and receive a security token in the response. The return of the security token means a connection has been created successfully. This flow is depicted in Figure 2.
A summary of the process to connect is presented as follows. (For complete information about the formats and required values of the `SessionCreateRQ` message, see Chapter 2 in *Guide to Accessing and Consuming SOAP-Based Sabre APIs* on the *Getting Started* web page.)

**Request 1.** The SOAP message for the `SessionCreateRQ` Service is created on the client side.

1. Create the SOAP envelope in the required format for *Sabre APIs*. Include the required values for the `SessionCreateRQ` Envelope. Generate the value for `eb:ConversationId`, and include your values for `eb:CPAId` and security credentials in `wsse:Security` node. Ensure the value for `eb:Action` for this request is `SessionCreateRQ`.

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

3. Send the `SessionCreateRQ` request message to the endpoint for consuming *Sabre APIs* over HTTPS. You 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, see Chapter 6 in *Guide to Accessing and Consuming SOAP-Based Sabre APIs* on the *Getting Started* web page.)

**Response 1.** The *Sabre APIs* gateway 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. Upon authorization, it allocates a *Sabre* session. (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 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 Web service requests for travel content, the connection ID is needed for all transactions with the Sabre APIs infrastructure that use a specific connection, whether the client maintains state or not.

## Closing Connections

When you need to close a Sabre APIs connection, obtain the connection ID of the connection you want to close, and include it in the `SessionCloseRQ` message. A summary of the process is presented as follows.

**Request 1.** The SOAP message for the `SessionCloseRQ` Service is created on the client side. (For complete information about this SOAP message’s formats and required values, see Chapter 2 in Guide to Accessing and Consuming SOAP-Based Sabre APIs on the Getting Started web page.)

1. Create the SOAP envelope in the required format for 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 you want to close. These values were sent in the `SessionCreateRQ` request and returned in `SessionCreateRS` response. Ensure the value for `eb:Action` for this request is `SessionCloseRQ`.

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

3. Send the `SessionCloseRQ` request message to the endpoint for the Sabre APIs environment where the connection lives. (For complete information about the URLs and environments, see Chapter 6 in Guide to Accessing and Using Services on the Getting Started web page.)

**Response 1.** The Sabre APIs gateway receives the request. The infrastructure closes the connection and the allocated Sabre session is returned to the session pool. The Sabre work area 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 Sabre APIs session is allocated along with the connection. The type of session depends on the user ID in the security credentials 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 *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 with the *Sabre* host system. This enables them to use the *Sabre* system.

A *Sabre* host session is a specific type of session. (For simplicity, the *Sabre* host session is referred to as a *Sabre* session. Remember that a *Sabre* session is the same as a TA.) It is associated with a LNIATA in native *Sabre* systems (also referred to as PSS). The user IDs of *Sabre* system users 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 *Sabre APIs* infrastructure creates a new connection to *Sabre APIs* and allocates a *Sabre* session from the subscriber's session pool. The *Sabre* session becomes active and is no longer available in the session pool until the connection is closed or the connection and session time out.

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

*Sabre* sessions in a session pool are also shared across the testing and production environments of *Sabre APIs*. (For more information, see Table 5 in *Guide to Accessing and Consuming SOAP-Based Sabre APIs* on the Getting Started web page.)

**Shopping Cart Functionality and the *Sabre* Work Area**

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 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 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 is retained.
To store transactions in the *Sabre* work area in a specific *Sabre* session, the client must use the Web service designed to end the transaction when the workflow is completed. This Web service is EndTransactionLLSRQ.

When reusing a connection, you are strongly advised to clear the *Sabre* work area before sending messages in a new workflow. The IgnoreTransactionLLSRQ Service clears the *Sabre* work area. This prevents mingling content from the new workflow with content from the previous use of the *Sabre* session.

**Note:** If your client crashes or you experience 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 because the connection was not closed explicitly. By not specifically clearing the work area, you risk mingling content from the re-used, recovered connection ID and associated *Sabre* session with your new workflow.

**Release of *Sabre* Sessions**

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

If you let unneeded connections time out instead of closing them properly with SessionCloseRQ, it is possible that all connections and sessions in your pool will be in use and unavailable until they time out.

Letting sessions time out may put you in a situation where you will not have any connections available for log in, and your clients will need to wait until the connections time out before they can log in.

If all *Sabre* sessions in your session pool are allocated, your client receives an error when it tries to log in.

**Sessions with Open Systems *Sabre* APIs**

For subscribers who use open systems *Sabre* APIs, a session is created for use, as required by business applications and systems of other service providers within *Sabre*. 
**Sabre APIs Time-Outs**

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

- The SessionCloseRQ Service 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 time-out value is 15 minutes; for some situations, this value can be set as high as 30 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 *Sabre APIs* connection and associated *Sabre* session from timing out, a client can send any Web service. Sending the SessionValidateRQ Service with a valid conversation ID and security token is recommended for this purpose. The SessionValidateRQ Service has no effect on content in the *Sabre* work area.

It is not advisable to let connections time out. It is the responsibility of the client to either close *Sabre APIs* connections explicitly with SessionCloseRQ 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, *Sabre APIs* connections are not guaranteed to be alive.

**Approaches for Handling Connectivity**

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 *Sabre APIs*. A basic connection is similar to a conversation. You start a conversation (open a connection with the SessionCreateRQ Service), you exchange requests for content and receive the responses (send and receive *Sabre APIs* messages in the form of SOAP-based TPF Connector or open systems *Sabre APIs*), then you end the conversation (close the connection with the SessionCloseRQ Service). The client to connection ratio is 1:1, in other words, one client equals one connection. This is illustrated in Figure 3.
Figure 3. Basic Connection

When your client needs a connection to the Sabre APIs gateway to send a business workflow, it opens a new connection. With this solution, the client retains and resends the connection ID in all 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, 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, but sending multiple workflows, follows. (For complete details about the SOAP message formats for session management and travel-based Sabre APIs, see Chapter 2 in Guide to Accessing and Using Services on the Getting Started web page.)

**Request 1.** The client creates the SOAP message for the SessionCreateRQ Service in the required format with the required values, and sends it to the endpoint for consuming Sabre APIs over HTTPS.

**Response 1.** The 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.

In the SOAP envelope of the SessionCreateRS response, 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.
a. 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.

b. The client formats the payload as described in “Request Messages for Travel Content” in Chapter 2 of _Guide to Accessing and Using Services_.

c. The client requests a specific Web service version in the `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 `SessionCreateRQ` SOAP envelope.

**Note:** For the service-specific values and valid data elements in the payload of the Web services you want to send, consult the design, schema, and developer notes documents for those Web services on 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 `EndTransactionLLSRQ` Service to save the transaction and PNR that are temporarily in the _Sabre_ work area of the _Sabre_ system.

**Response 4.** _Sabre APIs_ return 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.

The client parses and stores any content it wants in the responses.
The client is done with the workflow.

**Request 6.** The client sends the EndTransactionLLSRQ Service to save the transaction and PNR in the *Sabre* system.

**Response 6.** *Sabre APIs* return a record locator for the PNR to the client.

The client is ready to close the connection to *Sabre APIs*.

**Request 7.** The client requests termination of the connection by sending the SessionCloseRQ Service. The SOAP envelope includes the same values for \texttt{eb:ConversationId}, \texttt{wsse:BinarySecurityToken}, and \texttt{eb:CPAId} used to open the connection.

**Response 7.** The *Sabre APIs* infrastructure ends the session and closes the connection simultaneously, and renders the security token invalid. The SessionCloseRS response message is returned to the client.

### When to Use Basic Connections

If your need for connections is low in volume or if you are doing 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 your 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 your 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, you have multiple clients. A many-to-one ratio of connections to clients exists; you generally have more open connections than clients. Multiple clients and a connection pool are illustrated in Figure 4.
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 Service 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. You persist the connection IDs so that clients can reuse them.

When needed, a client obtains an available connection from the pool to send the Sabre APIs service 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. See below: Connection Managers.

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. For this information, see Chapter 2 in Guide to Accessing and Using Services on the Getting Started web page.

1. The connection manager sends multiple SessionCreateRQ Service requests to create Sabre APIs 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 Service 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, 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 receives the request and does the following:

- 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:

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

2. 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.

3. 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.

4. The connection manager clears the *Sabre* work area before handing over the connection ID to the client. The connection manager sends the IgnoreTransactionLLSRQ Service, which discards any content that remains from a previous *Sabre* session that used the connection ID.

**Note:** 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.
5. The client exchanges Web services 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 Web service, in this case, EndTransactionLLSRQ.

(For information about storing content, clearing buffers, and other activities that manipulate content and the Sabre work area, see “Maintaining Session State in Your Client” in Chapter 5 of Guide to Accessing and Using Services on the Getting Started web page.)

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

7. 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.

8. 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 of any content that lingers from the previous Sabre session by sending the appropriate SOAP-based TPF Connector Sabre APIs, in this case IgnoreTransactionLLSRQ.

Depending on your business model, you can clear the work area when a workflow is completed or just before beginning a new one.

9. 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 SessionValidateRQ Service.

10. The connection manager closes excessive Web services 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 Service is consumed, all the internal resources held by the connection and session are released, and the current quantity of active Web services 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 that lingers from a previous session.

**When to Use Connection Pooling**

When you need multiple clients, and the quantity of connections you need exceeds the quantity of clients you have, this form of management is recommended. For a steady volume of 1 to 2 transactions per second, this technique is suitable. If your business process needs multithreaded processes, you need multiple, open connections. That is the only way to send simultaneous service 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. You save time and resources by reusing connections instead of creating them every time you need 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 5.

![Figure 5. Connection Manager Architecture with Limited Redundancy](image)

As shown in Figure 5, 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 your environment is high volume, implementation of a connection manager with the level of redundancy you need is essential. If you cannot afford to have down time, a solution with full redundancy is recommended. Multiple simultaneous connections are also needed for multithreaded 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

  You define a predetermined number of connections for your connection manager that are a low threshold of connections you need to initialize and accept low traffic. 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 you have multiple IPCCs, each IPCC is allocated a quantity of session in your session pool. (In legacy systems, a session pool was referred to as a TAM pool.)

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

  The quantity of TAs you have is based on information provided to your Sabre account representative, and is defined in your 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 Service.

- Throttle the quantity of open connections

The connection manager ensures the availability of a minimum number of *Sabre APIs*
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. See
also: *Implementation Scenarios* which explains this in greater detail.

- 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, the status of the connection, that is, whether the connection is
free or in use, and the client ID, in other words, which client is using it.

The connection manager caches and stores the connection information, and updates it 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 SessionValidateRQ message.

- 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 Service. Remember that this Web service also terminates the allocated Sabre session, and returns the Sabre session to the session pool. All data in the work area 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 Service 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 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 your business process.

- 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 service-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.

For errors returned by the Sabre APIs infrastructure, see Chapter 7 in Guide to Accessing and Using Services.

Connection Manager Implementation

You use the conversation ID and security token to manage your open connections with the Sabre APIs infrastructure, which again, if using TPFC/LNIATAs, are also synchronized with a Sabre session.

December 31, 2014
To effectively manage Web services connections, basic connection information must be stored so that clients can retrieve connections from a pool as needed and the connection manager can track the connections and keep them alive. Each entry inside the pool has the format shown in Figure 6.

![Connection Information Fields](image)

### Figure 6. Connection Information Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Use Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Token</td>
<td>The security token returned with the creation of the connection</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>
</tr>
<tr>
<td>Conversation ID</td>
<td>The conversation ID used to create the connection</td>
</tr>
<tr>
<td>Status</td>
<td>A value indicating whether a connection is free or in use by a client</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>
</tr>
</tbody>
</table>

When the box with the connection pool is started and the pool is opened, connection manager sends the SessionCreateRQ messages 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 begin service 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 IgnoreTransactionLLSRQ Service to clear the Sabre work area 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 SessionValidateRQ Service is recommended. This is illustrated more fully in Figure 7 on page 26.

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 Service. 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, your architecture should be redundant. You can choose which components to duplicate, or you can replicate all components, as shown in Figure 7. This enables failover, load balancing, and recovery. Your business needs dictate how to design your architecture.
Single and Multithreaded Workflows

You 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.

- **Single threads**

To minimize the quantity of Sabre sessions used in your 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 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.

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

Implementation Scenarios

Some sample scenarios are presented in this topic. You may discover other designs that suit your needs.
Scenario 1

When the machine with the connection pool is started and the connection pool is opened, the connection manager sends the SessionCreateRQ messages 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 SessionValidateRQ Service. If the pool has more than 20 unused connections, the client calls the SessionCloseRQ Service 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 8 (on the next page) 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 8. Sequence Diagram for a Connection Manager
Glossary

AAA

Pronounced “triple A”.

Abbreviation for Agent Assembly Area

See also: Sabre work area.

Agent Assembly Area

This document uses the term Sabre work area.

See also: Sabre work area.

basic connection

Basic connection is the simplest approach for handling connecting to Sabre APIs. It is similar to a conversation. You open a connection with the SessionCreateRQ Service, next you exchange requests for travel content and receive the responses using SOAP-based TPF Connector Sabre APIs or open systems Sabre APIs, then you close the connection with the SessionCloseRQ Service. 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 also: security token.

connection

A connection is an open channel to the 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 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 of the Sabre host system.
The distinction between the terms “connection” and “session” is semantic. A client requests a **connection** to the *Sabre APIs* infrastructure, and upon successful connection, a *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.

See also: [Sabre session](#) and [Sabre APIs session](#).

**connection ID**

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

**connection manager**

The practice of managing *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](#).

**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 also: [basic connection](#).
GDS

global distribution system. The *Sabre* system is a GDS.

See also: *PSS* and *Sabre global distribution system (Sabre GDS)*.

IPCC

Abbreviation for Internet Pseudo City Code

open systems Sabre APIs

*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.

PCC

Abbreviation for Pseudo City Code

PSS

Abbreviation for Passenger Service System

Sabre global distribution system (Sabre GDS)

This document uses the term *Sabre* system.

See also: *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.

This document uses the term *Sabre* session instead of TA.

See also: *connection* and *session pool*. 
**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 Web services provided by *Sabre*. Under the umbrella of *Sabre APIs* are SOAP-based TPF Connector *Sabre APIs*, open systems *Sabre APIs*, and session management Web services.

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 *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 *Sabre APIs* is through the *Sabre APIs* infrastructure.

**Sabre APIs session**

A session that is allocated when a client makes a connection to the *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**

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*.

The *Sabre* work area 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 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 dependencies.

**security token**

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

**session management *Sabre APIs***

Web services that are designed to connect to and disconnect from the *Sabre APIs* infrastructure. The session management services are SessionCreateRQ, SessionCloseRQ, and SessionValidateRQ. 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. The session pool is also known as 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 SessionCreateRQ Service, 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.

See also: [connection pool](#) and [TAM pool](#).

**state and stateful**

The *Sabre* system is stateful. The *Sabre* work area 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

Abbreviation for Terminal Address

See also: Sabre session.

TAM

Abbreviation for 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 TAM pool.

See also: session pool.

TPFC

Abbreviation for Transaction Processing Facility Connector

The SOAP-based TPF Connector Sabre APIs retrieves content from the Sabre system, or PSS

See also: PSS.

SOAP-based TPF Connector Sabre APIs

The offering of 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.

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 Web services obtain their content from other business applications within Sabre.

See also: session management Sabre APIs.
Transaction Processing Facility Connector

The Transaction Processing Facility *Sabre APIs* connector that retrieves content from the *Sabre* global distribution system, which is also referred to as the *Sabre* host system or PSS (Passenger Service System).