

*QuickBooks® SDK*

*Developer's Guide  
for QuickBooks Online Edition*

Version 4.0

SDK version 4.0, released November 2004. (c) 2004 Intuit Inc. All rights reserved.

QuickBooks and Intuit are registered trademarks of Intuit Inc. All other trademarks are the property of their respective owners and should be treated as such.

Acknowledgement: This product includes software developed by the Apache Software Foundation (<<http://www.apache.org>>) (c) 1999-2004 The Apache Software Foundation. All rights reserved.

Intuit Inc.  
P.O. Box 7850  
Mountain View, CA 94039-7850

For more information about the QuickBooks SDK and the SDK documentation, visit <http://developer.intuit.com/QuickBooksSDK/>.

## About This Guide

Who Should Read This Guide . . . . .	vii
What's New in This Guide . . . . .	vii
New Simplified Application Connection Support for QBOE . . . . .	viii
New Transaction Query Filter Support . . . . .	viii
Additional Support for QuickBooks SDK Objects and Operations . . . . .	viii
Before You Begin . . . . .	ix
What's in This Guide? . . . . .	x

## Chapter 1: Overview

Important Terminology Notes Regarding the QuickBooks User Interface . . . . .	1
What Is QuickBooks Online Edition? . . . . .	1
QuickBooks Online Edition vs. Other QuickBooks Products . . . . .	2
How Do the Integrated Applications Differ? . . . . .	3
What Types of Integrated Applications Are Supported? . . . . .	3
Desktop Applications . . . . .	3
Server Applications . . . . .	4
How Integrated Applications Access QuickBooks (Access Control) . . . . .	5
How Desktop Applications Access QBOE . . . . .	5
How Server Applications Access QBOE . . . . .	6
More About the Connection Ticket . . . . .	7
Connection Tickets and Session Tickets In Server Applications . . . . .	7
Why Multiple Authorizations per Application Are Useful . . . . .	8
Further Access Control for Server Applications . . . . .	9
Registering with Intuit Developer Network . . . . .	9
Special Registration and Certification for Server Applications . . . . .	9
How to Register . . . . .	9
Where to Go From Here . . . . .	9

## Chapter 2: Getting a Certificate for a Server Application

Using Certificates in Server Applications: Background Information . . . . .	11
Certificate Requirements for a Hosted Application . . . . .	11
About the Java keytool Utility and Keystore Files . . . . .	13
Process Overview: Certificate Provisioning . . . . .	13
Creating Your Application Key . . . . .	14
Generating a Certificate Request . . . . .	15
Registering Your Application . . . . .	15
Importing the Intuit CA Certificate . . . . .	15
Importing the Intuit Signed Certificate . . . . .	16

## Chapter 3: Registering Your Application

Registering a Desktop Application . . . . .	17
Registering a Server Application . . . . .	18

## Chapter 4: Integrating a Desktop Application

What You Need to Do . . . . .	21
Making Required Registry Entries . . . . .	22
Connecting to a QBOE Company . . . . .	23
In OpenConnection2 . . . . .	23
In BeginSession . . . . .	24
In CloseConnection . . . . .	24
Desktop Application Security Requirements . . . . .	24
Handling Errors . . . . .	24
Using Installation IDs . . . . .	25

## Chapter 5: Integrating a Server Application

Server Application Security Requirements . . . . .	27
Storing the Connection Ticket Securely . . . . .	27
Using Installation IDs. . . . .	28
Integration Task Overview . . . . .	28
Setting Up Encryption/Decryption Using the IDN Certificate . . . . .	29
Example: Implementing Encryption/Decryption in a Servlet. . . . .	29
Posting to the Data Exchange URL. . . . .	30
Supporting User Authorization (Getting Connection Tickets) . . . . .	30
Enabling User Authorization in Your Main Form . . . . .	30
Constructing the URL Required for Authorization . . . . .	30
Handling the Authorization Callback . . . . .	31
Getting the Session Ticket . . . . .	35
Where in the Server Application Should You Get Session Tickets?. . . . .	36
How Do You Get Session Tickets?. . . . .	36
Supplying Client Date/Time in the Required Format. . . . .	42
Supporting User Cancellation of Authorization. . . . .	42
Enabling User Cancellation in Your Main Form . . . . .	42
Constructing the URL Required for Cancellation . . . . .	42
Handling the Cancellation Callback . . . . .	43
About the SDKTest.java Sample . . . . .	43
Handling Errors . . . . .	45

## Appendix A: Status Codes Returned in Responses

## Appendix B: Supported qbXML Operations

List Object Support . . . . .	51
Transaction Object Support. . . . .	51
Data Extensions Support for Company Object. . . . .	52
List Query Support . . . . .	52
Transaction Query Support . . . . .	53
Other Query Support . . . . .	53
Unsupported Objects Related to Add Operations . . . . .	53

## Appendix C: Signon Messages and Responses XML

## Appendix D: Supported qbXML Operations for QuickBooks Simple start Edition

## Appendix E: Integrating Desktop Applications without Using the QBOE Connector

Application Interaction with QuickBooks Online Edition Sites . . . . .	63
Interactions with the QuickBooks Online Edition Authorization Site . . . . .	63
Interactions with the QuickBooks Online Edition Cancellation Site . . . . .	64
Interactions with the Session Authentication Site . . . . .	65
Normal Data Exchange with the QuickBooks Company . . . . .	67
Posting to the Data Exchange URL . . . . .	67
Desktop Application Security Requirements . . . . .	68
Storing the Connection Ticket Securely . . . . .	68
Using Installation IDs . . . . .	68
Integration Task Overview . . . . .	69
Supporting User Authorization (Getting Connection Tickets) . . . . .	69
VB Sample: Subscribe Button-Click Handler . . . . .	69
Getting a Session Ticket . . . . .	70
VB Sample: Getting a Session Ticket . . . . .	71
Building the SignonDesktopRq Element to Get a Session Ticket . . . . .	75
Parsing the SignonDesktopRs Response for the Session Ticket . . . . .	79
Constructing the SignonTicketRq Element for Data Exchange . . . . .	80
Sample XML for the SignonTicketRq Element . . . . .	81
VB Sample: Constructing the SignonTicketRq Element . . . . .	82
Supporting User Cancellation of Authorization . . . . .	85
VB Sample: Unsubscribe Button-Click Handler . . . . .	85
Supplying Client Date/Time in the Required Format . . . . .	86
Sending Users to QuickBooks Online Edition Sites . . . . .	86
Posting Data to a QuickBooks Online Edition Company . . . . .	87
Implementing "Logout" Functionality . . . . .	87
Handling Errors . . . . .	88
Using wincrypt to Store Connection Tickets . . . . .	89

## Index



# ABOUT THIS GUIDE

This *Developer's Guide* describes the integration of desktop and server applications with QuickBooks Online Edition (QBOE). The purpose of this guide is to provide the details you need to know in order to successfully post qbXML messages from your application to a QBOE company and receive responses.

However, this guide does not cover the content of the qbXML messages. Nor does this guide provide details on handling the responses, in particular, handling and recovering from errors. For this material, please refer to the *QuickBooks SDK Technical Overview*, the *QuickBooks SDK Concepts Manual*, and the *QuickBooks SDK Onscreen Reference*.

In this guide, the examples for integrated desktop applications are in Visual Basic. For integrated server applications, the examples are in Java.

## Who Should Read This Guide

---

This guide is for developers who are creating desktop or server applications that integrate with QBOE.

In order to create a desktop application, you should know a little about XML and how to assemble and post XML documents to a web URL (and handle responses) in your programming language of choice.

The knowledge requirements for server applications are more involved. You should be familiar with HTTPS and XML. Depending on your implementation, you may also need to know about Java servlets/JSP or similar technologies, since your server application will be processing data posted from users' web browsers and will also be interacting with the remote QBOE server. You need to be familiar with persistent storage (database) technologies because your application will need to use this to store, at a minimum, connection tickets and certain user information. You also need to know how to manage server and client certificates in your development environment.

## What's New in This Guide

---

### **IMPORTANT**

QBOE features can change between SDK releases. Accordingly, to keep up to date with current QBOE features, refer to the IDN developer network website at <http://developer.intuit.com>.

This section describes new features and changed behaviors for QBOE coincident with the release of QBSDK 4.0.

## New Simplified Application Connection Support for QBOE

Connection to QBOE from a desktop application is now much simpler due to a new redistributable plugin for the QBXMLRP2 request processor. This plugin is called the QBOE connector for integrated applications. The QBOE connector works with the QBXMLRP2 request processor to perform all of the work involved in obtaining and managing connection tickets and session tickets from QBOE. Consequently, it is now far simpler to write a desktop application that works with QBOE. For more information, see the chapter on integrating a desktop application later in this document.

In conjunction with the QBOE connector, both qbXML and QBFC provide a new Open Connection call that allows the application to specify which QuickBooks edition the application wishes to connect to. For more information, see the *Developer's Guide for QuickBooks*, the *Developer's Guide for the QBFC Library* and the *Onscreen Reference (OSR)* provided with the QuickBooks SDK.

## New Transaction Query Filter Support

Beginning with SDK 4.0, QBOE supports transaction filters that will enable you to set up your queries the way that you need to in order to break up large queries into more manageable chunks. These filters are listed and documented under each type of transaction query in the *Onscreen Reference*.

## Additional Support for QuickBooks SDK Objects and Operations

Additional functionality in the QBSDK is now supported for QBOE. (For documentation of these, see the *Onscreen Reference (OSR)* provided with the QuickBooks SDK.) The following support is now available in QBOE:

- Added the following support for SalesReceiptAdd, InvoiceAdd, and CreditMemoAdd
  - > Added the DiscountLineAdd aggregate
  - > Added the SalesTaxLineAdd aggregate
  - > Added the ShippingLineAdd aggregate
- Added the following aggregates in the SalesReceiptRet, InvoiceRet, and CreditMemoRet return objects:
  - > DiscountLineRet
  - > SalesTaxLineRet
  - > ShippingLineRet
- Added the IsTaxable Boolean type to the following:

- > InvoiceLineRet
- > InvoiceLineAdd
- > SalesReceiptLineRet
- > SalesReceiptLineAdd
- > CreditMemoLineRet
- > CreditMemoLineAdd
- Added support for APAccountRef in the following:
  - > BillRet
  - > BillAdd
  - > VendorCreditRet
  - > VendorCreditAdd

## Before You Begin

---

Before you read the rest of this guide, be sure you've read the *Technical Overview* for the QuickBooks Software Development Kit (SDK). This guide assumes you're already familiar with the introductory material and key concepts contained in the overview. Figure 1 shows the complete SDK documentation set and the other documents you'll consult as you develop your integrated application.

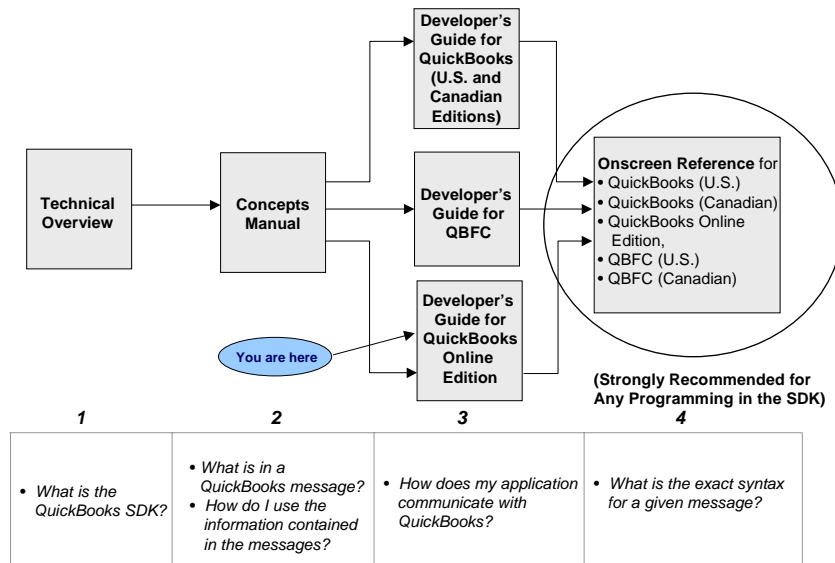


Figure 1 Documentation Roadmap

When you're finished with this manual, be sure to familiarize yourself with the material contained in the *Concepts Manual*. Also check out the *Onscreen Reference* for QuickBooks, which contains the qbXML syntax for each request and response message type.

In many cases, developers may be creating an application that integrates with several QuickBooks products. If you are developing for multiple QuickBooks products, be sure to read the *Developer's Guide* for each target QuickBooks product, as shown in Figure 1. (Note too that the *Concepts Manual* applies to integrating with any QuickBooks product.)

## What's in This Guide?

---

This *Developer's Guide* contains the following chapters:

Chapter 1, "Overview," describes the supported application architectures (desktop and server), the main concepts behind communication with QBOE (user authorization expressed in connection and session tickets), and an integration process overview.

Chapter 2, "Getting a Certificate for a Server Application," provides instructions on obtaining a certificate from IDN for your server application. Every server application must request and obtain a certificate from IDN.

Chapter 3, "Registering Your Application," provides instructions on registering your desktop or server applications with Intuit Developer Network (IDN). Every application must be registered with IDN.

Chapter 4, "Integrating a Desktop Application," provides instructions and samples for building a desktop application that successfully accesses QBOE.

Chapter 5, "Integrating a Server Application," provides instructions and samples for building a server application that successfully accesses QBOE.

Appendix A, "Status Codes Returned in Responses," lists the possible status codes that can be returned in the responses to qbXML requests posted to QBOE.

Appendix B, "Supported qbXML Operations," lists the qbXML operations that are supported by QBOE.

Appendix C, "Signon Messages and Responses XML," provides a sample XML document that shows the constructed signon message requests and responses.

Appendix E, "Integrating Desktop Applications without Using the QBOE Connector," shows how to manage the connection and session ticket interactions manually, including encrypting tickets. This is the older (and more difficult) method of integrating with QBOE but is still supported for developers who want more control over the forms displayed during the obtaining of the connection and session tickets.

# CHAPTER 1

## OVERVIEW

This chapter provides an overview on building applications that are integrated with QuickBooks Online Edition (QBOE). It covers the application architectures that are supported, the general structure of an integrated application, and the preliminary steps, such as registration, that you need to perform in order to enable your application to communicate with QBOE.

This chapter assumes that you are familiar with QuickBooks, and have already read the *QuickBooks SDK Technical Overview*, the *QuickBooks SDK Concepts Manual*, and are familiar with the *QuickBooks SDK Onscreen Reference*.

### **IMPORTANT**

This overview describes integrated desktop applications that use the new QBOE Connector, which virtually eliminates special QBOE coding requirements. For information on the alternate behavior of desktop applications using features prior to SDK 4.0, see Appendix E, "Integrating Desktop Applications without Using the QBOE Connector."

## Important Terminology Notes Regarding the QuickBooks User Interface

---

For the most part, this guide uses the same terminology as that found in the QBOE user interface. However, there are a few exceptions, shown as follows:

Table 1-1 Terminology Differences From the QBOE User Interface

<b>QBOE User Interface</b>	<b>This Guide</b>
login security	session authentication
login key	session ticket
connection key	connection ticket

The reason for the difference is partly due to the different audiences: the UI serves general users and this guide serves programmers. It is also partly due to the implementation insofar as you will be constructing XML documents with elements that are named `SessionTicket` and `ConnectionTicket`.

## What Is QuickBooks Online Edition?

---

QuickBooks Online Edition is a new way to manage small business finances. It is a completely new product built from the ground up to be accessed over the Internet from a web browser via a secure HTTPS connection (see Figure 1-1).

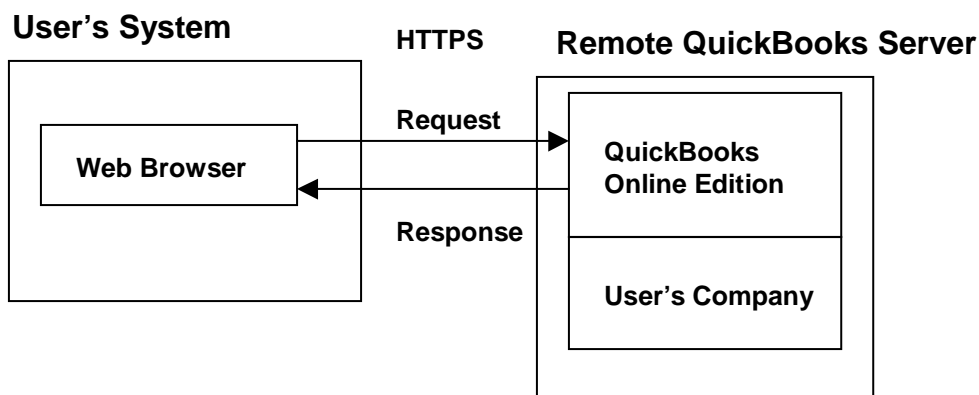


Figure 1-1 QuickBooks Online Edition Architecture

As shown in the figure, the end user accesses QBOE at the remote server via a web browser.

The primary advantages of this arrangement are the following:

- The ability to access business data at any time from any location
- The use of the latest QuickBooks version at the remote server, eliminating the need for the end user to upgrade

## QuickBooks Online Edition vs. Other QuickBooks Products

---

QuickBooks Online Edition is a completely new product. It is not directly interoperable with the other QuickBooks products, although company data from the other QuickBooks products can be migrated to it.

Although the products are different, the SDK provides similar functionality for both products. That is, applications integrated with QBOE can use very nearly the same set of qbXML messages that is used by applications integrated with the other SDK-supported QuickBooks products.

However, you should be aware that the SDK feature set available for QBOE is a subset of those features available for those other QuickBooks products. There are also some additional SDK differences: some supported features are supported only in a limited way, and some features are supported for QBOE only. See Appendix B of this guide for a list of SDK functionality supported for QBOE.

## How Do the Integrated Applications Differ?

---

Desktop applications integrated with other QuickBooks products and those integrated with the QBOE differ only in the SDK feature set available to each. They both communicate via a local QuickBooks request processor DLL. However, the QBOE-integrated desktop application does use an additional software component called the QBOE connector that transparently handles communication with QBOE (see Figure 1-2). Notice that QBOE-integrated server applications differ substantially from desktop applications, as described later in this chapter.

## What Types of Integrated Applications Are Supported?

---

QuickBooks Online Edition supports two application architectures:

- Desktop, where the integrated application is running and executing on the user's system (see Figure 1-2)
- Server, where the integrated application is running on a remote application server accessible via a web browser (see Figure 1-3) at the user's system

Each of these architectures is described in detail below.

### Desktop Applications

---

Figure 1-2 shows a desktop integrated application accessing QBOE.

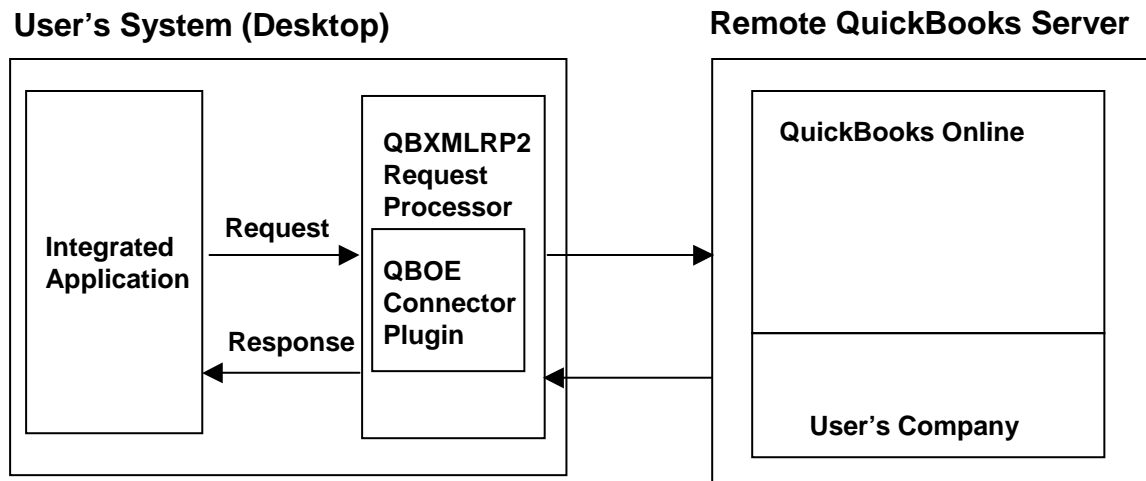


Figure 1-2 Integrated Desktop Application Architecture

Prior to SDK 4.0, QBOE-integrated desktop applications had to include special coding to work with QBOE to get and store connection tickets, issue HTTPS Get and Post method calls, and had to know the proper URLs for these activities. This presented a degree of difficulty for those porting applications or writing them new for QBOE.

Beginning with SDK 4.0, a redistributable software component called the QBOE connector is supplied that automatically handles all of this work. Now, a QBOE-integrated desktop application behaves in the same way as an application written for other QuickBooks editions and is coded in the same way, except for one parameter in the OpenConnection call. (Of course, QBOE continues to support a subset of SDK features.) The only other difference is that your application must register certain values in the Windows registry.

## Server Applications

---

Figure 1-3 shows the server architecture.

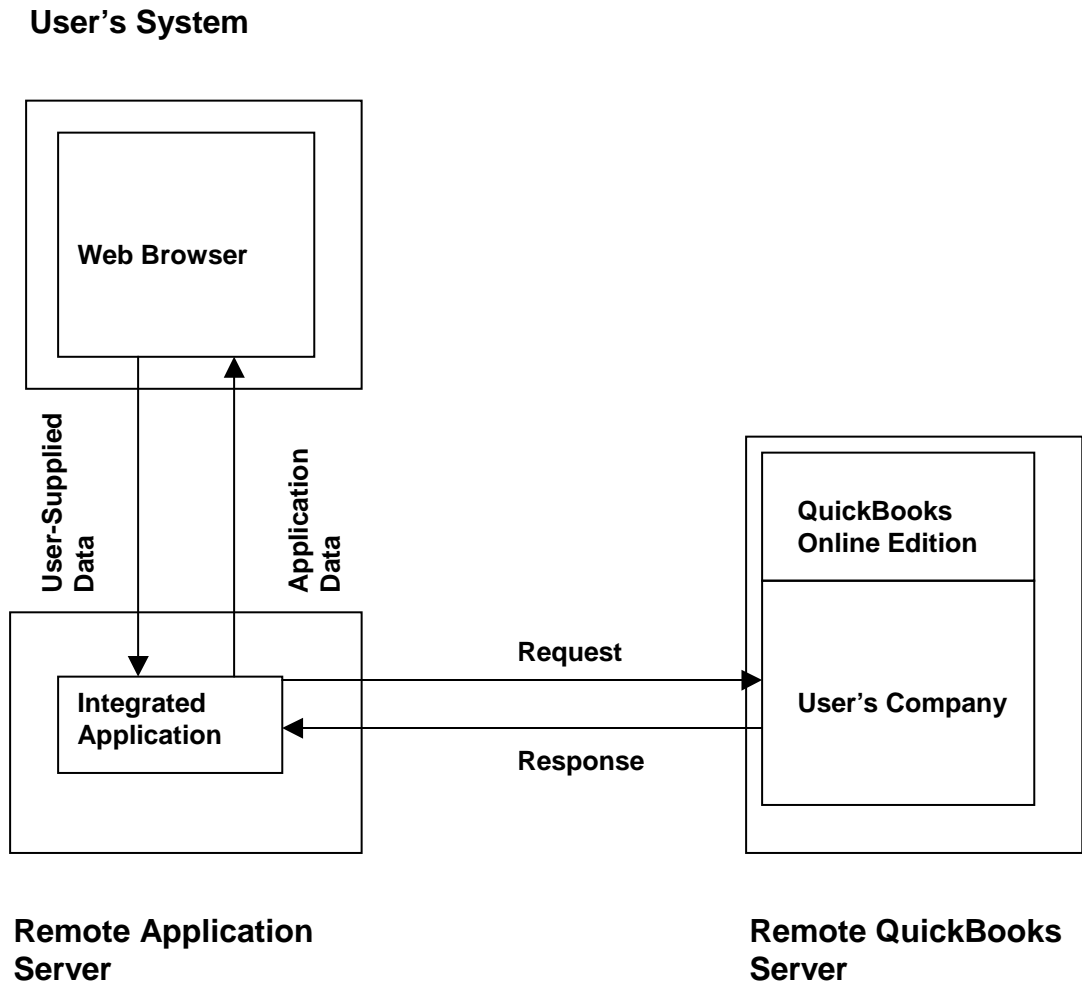


Figure 1-3 Integrated Server Application Architecture

The QBOE-integrated server applications do not use the QBOE connector, so the coding requirements for these remain the same as for prior releases. As is the case with desktop integrated applications, notice that the integrated server application does not necessarily require end-user interaction after authorization is granted by the administrative user.

# How Integrated Applications Access QuickBooks (Access Control)

The primary access control for both server and desktop integrated applications is the connection ticket, which represents the user's authorization of your application to connect to the user's company. This process varies depending on whether the application is a desktop or a server integrated application.

## How Desktop Applications Access QBOE

The connection ticket is obtained for your *desktop* application by the QBOE connector, if your application currently does not have a valid ticket. The QBOE connector sends the user to the QBOE authorization web page, where the user is prompted to specify the type of authorization and is prompted to copy the resulting connection ticket to a form temporarily displayed by the QBOE connector (see Figure 1-4).

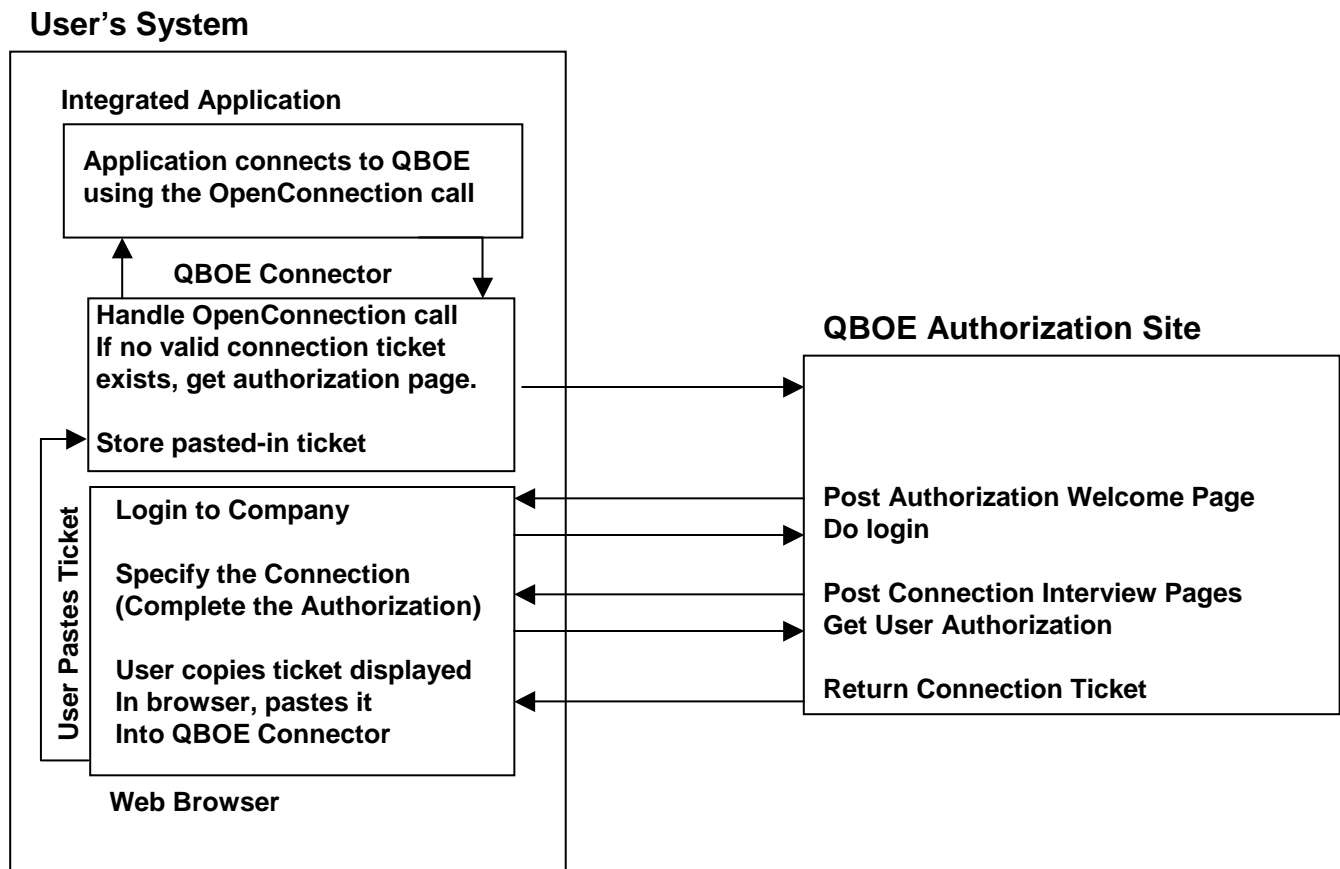


Figure 1-4 User Authorization in a Desktop-Integrated Application

Notice that your application does not know or need to know anything about the location of the QBOE server, connection tickets or session tickets, or about any user authorizations. Those items are handled for you by the QBOE connector.

## How Server Applications Access QBOE

Figure 1-5 shows how the connection ticket is obtained in a server application.

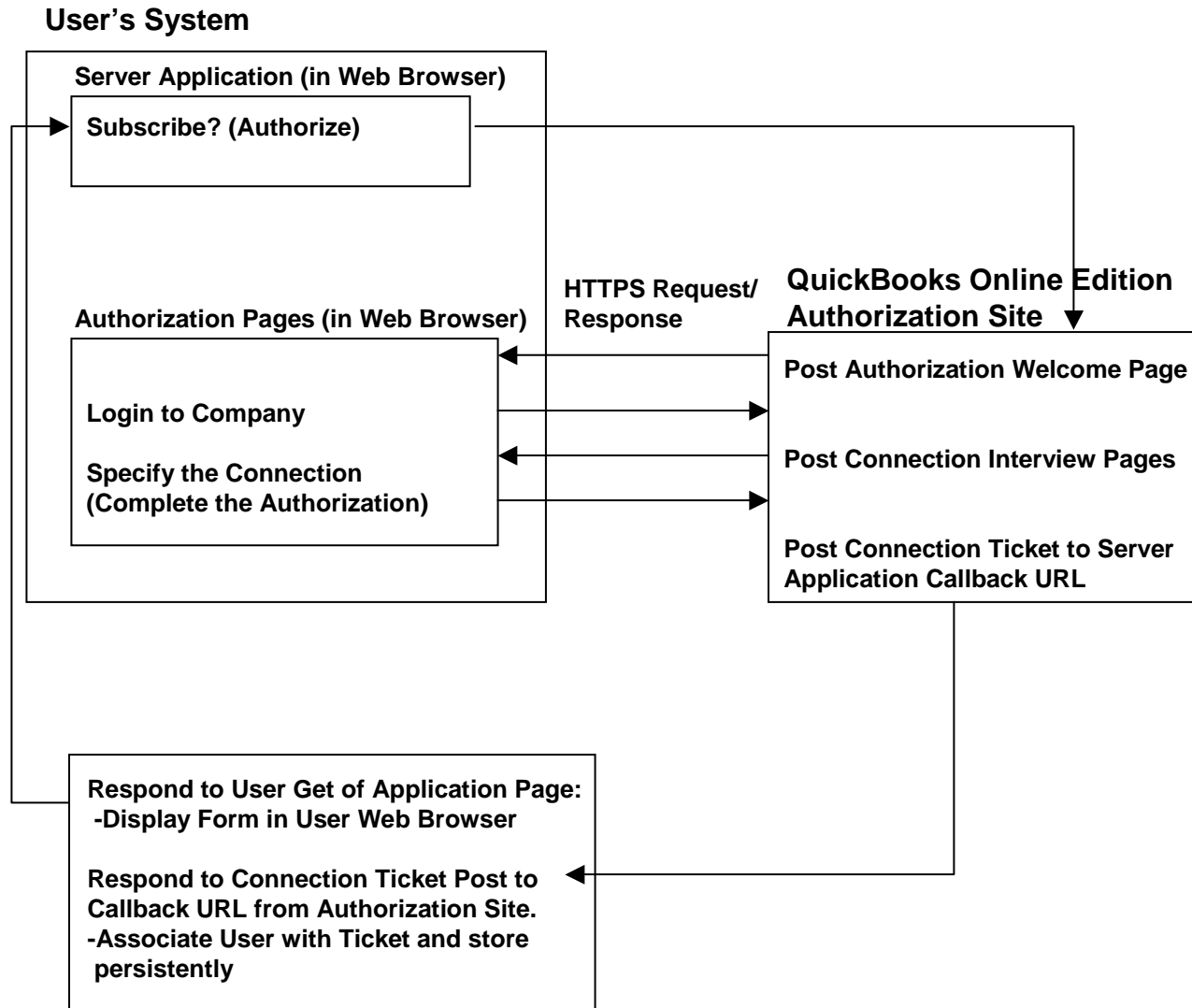


Figure 1-5 User Authorization in a Server-Integrated Application

As shown in Figure 1-5, for a *server* application, the user does not copy and paste the connection ticket into the application. Instead, the connection ticket is returned by QBOE directly to your server application via a callback URL.

An important thing to notice in both server and desktop applications is that once the user is sent to the authorization site, your application is aware of nothing but the connection ticket that gets returned. That is, your application does not know how the user is specifying the connection ticket.

Furthermore, notice that your application cannot determine directly from the returned connection ticket what kinds of restrictions have been placed on it by the administrative user. These restrictions make themselves known in certain error codes that are returned during application runtime, so your application needs to handle them. (This is described later in this guide in Chapter 4 and Chapter 5.)

## More About the Connection Ticket

---

During the connection creation interview at the QBOE authorization site, the administrative user can specify certain limitations for the connection ticket. The most important of these is whether to require additional security in the form of a forced QuickBooks logon each time your application begins a session with QuickBooks. In the authorization web pages, this is called login security, but in this document, it is called *session authentication*.

For an integrated *desktop* application, if the connection requires session authentication, then the QBOE connector will send the user to the QBOE authorization page at the start of a new session to obtain the session ticket and prompts the user to copy/paste the session ticket into the QBOE connector form. Again, your application is completely unaware of all of this. It is only aware that it gets a valid connection or not, after the QBOE connector and end user supply what is needed.

Another limitation that can be built into the connection ticket by the administrative user is *access rights restriction*, which is useful if the administrative user is creating connection tickets for other users with fewer access rights. The choices include full access to all QBOE areas, or only to customer/sales areas, or only vendor/purchaser areas, or access to both customer/sales and vendor/purchaser. In general the administrative user is encouraged to be as restrictive as possible when assigning access rights.

## Connection Tickets and Session Tickets In Server Applications

---

### **IMPORTANT**

Connection tickets and session tickets provide access to your user's QuickBooks company. Therefore they must be protected at all times. See Chapter 4 and Chapter 5 for specific security requirements.

When your integrated *server* application begins a session with a QBOE company, it could simply use the connection ticket for communication if the administrative user didn't choose session authentication. If session authentication was chosen during connection ticket specification, then your application must obtain and use a session ticket. (Obtaining a session ticket for a server application is described in detail in Chapter 5.)

### **IMPORTANT**

The easiest way to code your application is to use session tickets whether session authentication is in effect or not. Accordingly, this document follows that approach and shows the use of session tickets for all communication with QuickBooks.

Your server application cannot know in advance whether the user will specify a connection ticket requiring session authentication or one without session authentication. Therefore it must support both cases. The way your application does this is to perform a runtime check for a session authentication required error returned from the first post of the session.

The reason for this is that if session authentication *is* required by the connection ticket, the first post to QuickBooks will fail with an error code indicating that session authentication is required. Your application must handle this error by sending the user to the QuickBooks session authentication site, where the user performs a QuickBooks logon.

As is the case with connection tickets, for server applications the session ticket is returned directly to your server application via a callback URL.

Notice that if no session authentication is required for a particular connection ticket, the session ticket is automatically returned in the response message from the first post of the session. So if the check for the session authentication error is false (no error) then your server application must parse the response for that session ticket and use it. All of this is shown in detail in Chapter 5 of this guide.

### **IMPORTANT**

Once the integrated application has a valid connection ticket, it accesses the QuickBooks company *with no further QuickBooks logons* unless the administrative user specified "Login Security" (session authentication) at authorization time.

## Why Multiple Authorizations per Application Are Useful

It is possible for the administrative user to authorize an application several times, with a separate connection ticket returned and stored for each authorization. Your application code does not need to be aware of this as it is not affected by it. However, you might find it a useful piece of background information to know why an administrative user would do this.

QuickBooks activity logs are segregated by connection ticket. So by assigning different tickets to different users, you can provide for an audit trail showing which users did which activities in the company. This is especially important if users are geographically distributed and the administrative user wants to track activities by location or function.

However, session authentication must be in effect or this kind of activity tracking will not be enabled.

## Further Access Control for Server Applications

---

In addition to the connection ticket/session ticket and session authentication mechanisms described in the preceding paragraphs, server applications have an additional layer of security. All server applications must have a certificate issued by Intuit Developer Network. See Chapter 3 for more information on requesting certificates for server applications.

## Registering with Intuit Developer Network

---

Every application integrated with QBOE, whether desktop or server, must be registered via the Intuit Developer Network registration site. During the registration process, an application ID is assigned to your application, along with other required items. In addition, the contact name that you provide will enable Intuit to contact you in the event of problems with your application, as well as inform you of upcoming new releases, changes in the QuickBooks SDK, such as deprecation of various qbXML message sets, and so on.

## Special Registration and Certification for Server Applications

---

In addition to the registration described in the previous paragraph, server application developers must supply certain callback URLs and obtain certificates for their application. You must obtain a server certificate from a trusted authority such as Verisign or Thawte, and you must also obtain a certificate from Intuit.

The callback URLs are URLs that QuickBooks needs in order to inform your application of certain user-initiated events, such as the granting of authorization to your application to access the user company, or cancelling or changing that authorization.

Each server application requires its own certificate, and this certificate is authenticated by QBOE every time the server application sends a request.

## How to Register

---

See Chapter 3 in this guide about the application registration process. If you are registering a server application, you also need to perform the tasks in Chapter 2.

## Where to Go From Here

---

If you are developing desktop applications, you need to read implementation details provided in chapters 3 and 4, along with the list of supported SDK operations in Appendix B. If you are developing server applications, you need to read implementation details provided in chapters 2, 3, and 5, and the list of supported SDK operations provided in Appendix B.

For further information on the qbXML supported by QBOE, see the following documentation:

- *QuickBooks SDK Technical Overview*
- *QuickBooks SDK Concepts Manual*
- *QuickBooks SDK Online Reference*

## CHAPTER 2

# GETTING A CERTIFICATE FOR A SERVER APPLICATION

This chapter provides background information about the use of certificates in server applications and also detailed instructions describing how to request a certificate from IDN and what to do with the returned certificate.

### **IMPORTANT**

This chapter describes certificate provisioning in terms of the Java platform. However, you are not required to use Java to obtain a certificate or to supply the certificate during runtime execution of your server application.

## Using Certificates in Server Applications: Background Information

---

This section describes the requirements for the certificates exchanged with QBOE. Although these requirements may seem somewhat strict, especially for an application in development, please keep in mind that through this mechanism you have access to the confidential financial data of one or more small businesses. Therefore, Intuit must be absolutely sure of the identity of the application that QBOE is communicating with.

### Certificate Requirements for a Hosted Application

---

QBOE supports two types of applications:

- Desktop applications, which run on the user's desktop as a normal application.
- Hosted (also called server) applications, which run on a server as a web service.

Hosted applications are inherently more secure than desktop applications because both the application and QBOE can be sure of the identity of the other end of the network conversation through the exchange of server and client certificates over the secure HTTP (HTTPS) protocol.

TWO certificates are required for a hosted application to communicate successfully with QBOE:

- A client certificate
- A server certificate

## About the Client Certificate

When a 3rd party application sends a qbXML request to QBOE, that application is in the “client” role. It performs a POST of data to QBOE which is acting in the server role. In this scenario, the https protocol first checks the credentials of the server certificate presented by the QBOE application server and QBOE then requests a client certificate.

The client certificate presented should have the common name (CN) equal to the DNS name of the machine POSTing the request followed by the application login ID of the 3rd party application, the two items separated by a colon. For example, if an application POST's a request from “appserver.someone.com” and the application login for the application is “myapp.someone.com” then the CN of the client certificate should be “appserver.someone.com:myapp.someone.com”.

The client certificate must be signed by Intuit. The remainder of this chapter describes the process of generating a keypair, producing a certificate request for that keypair and submitting it to Intuit for signing using the Java “keytool”. (If you are not using Java for your development, please refer to the instructions for your development tools and server environment to determine how to generate an SSL keypair with the required common name and submit a certificate request.)

Before you can import the signed certificate, you must first install the Intuit CA certificate as a ROOT CA in your key storage. Again, the remainder of this chapter documents the process for the Java “keytool,” so if you use a different language and server environment please see the documentation for your environment to determine how to import the Intuit certificate followed by the certificate signed by Intuit.

## About the Server Certificate

When QBOE sends data (for example, a connection ticket or session ticket) to your application as a result of user action in QBOE (for example, completing a connection interview, canceling a connection, and so on) your application is acting as the server, while QBOE is the client.

In this case, just as when a user connects to your server from a browser via HTTPS, your server must present a server certificate to the client. The Common Name (CN) of the certificate must match the name of the server. For example, a connection to https://www.someone.com must present a server certificate to the client with a CN of “www.someone.com”. This certificate must be a completely valid certificate signed by a well-known ROOT CA such as Verisign or Thawte. If the certificate is not completely valid, QBOE will refuse to POST any data to the server and subscription/connection callbacks will fail.

You can easily determine if the certificate presented by your server is valid by connecting to it via https with an UNALTERED version of Internet Explorer. If Internet Explorer presents ANY warning messages (i.e. unsigned certificate, certificate signed by an unknown authority, expired certificate, etc.) then QBOE will refuse to send data to your server via the callback URLs you specified when you registered your application with Intuit.

### **IMPORTANT**

“Test” certificates from Verisign and Thawte require that a “Test CA” be installed as a trusted certification authority in your browser. Although installing this certificate in your browser will remove the warning, the Test CA certificate is not installed in QBOE. Accordingly, callbacks to your server will continue to fail until you obtain a non-test certificate from a valid certification authority.

## About the Java keytool Utility and Keystore Files

---

### **IMPORTANT**

Even if you are very familiar with keytool, please read the instructions in this chapter carefully because the IDN certificate granting process has special requirements.

The instructions in this chapter refer to a program called *keytool*, which is a standard Java utility located in the directory `<JavaInstallDirectory>\bin`. This utility is standard beginning with the Java JSE SDK 1.4; for earlier versions you may need to additionally download and use the Java JSSE package.

The keytool program is used to create a keystore and keys in the keystore, among other things. The keystore is a file containing one or more application certificates. Typically, a web server will have one keystore file for simplicity of management, although you could have as many keystore files as you want.

## Process Overview: Certificate Provisioning

---

To get a signed certificate from IDN you must

1. Use keytool to create public/private key pair for the application.
2. Generate a certificate request that contains the application key.
3. Register the application, during the course of which you supply the certificate request.
4. Import the signed certificate returned by IDN.
5. Import the Intuit certification authoring certificate.

This chapter describes all of these steps except for application registration, which is described in Chapter 3 of this guide.

# Creating Your Application Key

---

## **IMPORTANT**

In this procedure, when keytool prompts you for first name and last name, you **MUST** supply your DNS host name and AppLogin separated by colon. Otherwise, Intuit will not be able to build a certificate for you.

To create a key for your application,

- Invoke keytool as follows:

```
keytool -keystore <keystoreName> -alias <AppLogin>
-genkey
```

*keystoreName*

Supply the name of your existing keystore or the name of the keystore you want to create.

*AppLogin*

Specify the value `AppName.MyHost.net`, where `AppName` is the name of your application (no spaces) and `MyHost` is the name of your server host, and `.net` can be replaced by any valid Internet suffix (`.com`, `.net`, `.gov`, and so on.). This naming convention will ensure uniqueness of the AppLogins.

This value is used by QuickBooks to identify your application. It is used only internally to identify the application—your end user will never see this name. Write down the name you use for AppLogin because you must supply that same name for the AppLogin field later when you register your application with IDN.

- This next point is very important and easy to miss, especially if you have used keytool before. When the keytool utility prompts you for first and last name, you must instead supply your DNS host name and AppLogin, separated by a colon, like this:  
`www.MyHost.net:MyAppLogin`. where `.net` can be replaced by any valid Internet suffix (`.com`, `.net`, `.gov`, and so on.).

## **IMPORTANT**

During the creation of your application key as described below, you will specify your server host name and your AppLogin. At Intuit certificate creation time, IDN will perform a reverse DNS lookup on the IP address to make sure that the IP address matches the host name. Therefore, the hostname you use must support this reverse lookup.

- Supply the other parameters as normal when prompted by keytool, for example, your organization unit, your organization, etc.
- At the end of the keytool “interview” it displays the list of information you just supplied. Enter “yes” if it is correct or simply press Return if you need to edit one or more values. After you enter “yes”, it will take a few moments to generate the key.

- When prompted for the application key password, you can supply an individual password on the application key if you want, or you can simply accept the keystore password by pressing Return.

This completes the application key generation process. You are now able to generate a certificate request for your application.

## Generating a Certificate Request

---

After you create your application key you can create the certificate request, as follows:

- Invoke keytool as follows:

```
keytool -certreq -alias <AppLogin> -keystore  
<keystoreFileName> -file <AppLogin>.req
```

*AppLogin*

Supply the name you specified earlier for AppLogin when you created your application key.

*AppLogin.req*

Supply the name of the file that is to contain the certificate request data. Use your application's AppLogin with the .req suffix.

- When prompted, supply the keystore password. If you applied a password to your application key, you will be prompted for that as well.

The output of this process is a request file called <AppLogin>.req. When you register your application, you will copy the contents of this request file (it is a text file) into a text box in the registration form.

## Registering Your Application

---

After you successfully create your certificate request, you can register your server application. Please refer to Chapter 3 in this guide.

If you successfully register your application and submit your certificate request, IDN returns two certificates to you:

- A CA certificate that is emailed to you.
- An Intuit signed certificate that is returned to you from the registration process.

You need to import the CA certificate first, as this contains the public key needed to successfully import the signed certificate.

## Importing the Intuit CA Certificate

---

You must import the Intuit CA certificate before importing the Intuit signed certificate.

To import the Intuit CA certificate emailed to you by IDN into your keystore,

- Copy the contents of the Intuit CA certificate into a file that has the suffix `.cer` and save it in a directory accessible by `keytool`. Copy the entire data block, including the delimiters `-----BEGIN NEW CERTIFICATE REQUEST-----` and `-----END NEW CERTIFICATE REQUEST-----`.

- Invoke `keytool` as follows:

```
keytool -import -file <certFileName.cer>
        -alias intuit -keystore <keystoreName>
```

*certFileName*

Supply the name of the file in which you copied the Intuit CA certificate.

*keyStoreName*

Supply the name of your keystore.

- Supply the keystore password when prompted. If you applied a password to your application key, you will be prompted for that as well.
- When prompted whether you want to trust the certificate, enter “yes”.

At this point, you are ready to import the signed certificate supplied by IDN.

## Importing the Intuit Signed Certificate

---

The Intuit signed certificate is the certificate you need when communicating with QBOE.

To import the Intuit signed certificate into your keystore,

- Copy the contents of the Intuit signed certificate into a file that has the suffix `.cer` and save it in a directory accessible by `keytool`. Copy the entire data block, including the delimiters `-----BEGIN NEW CERTIFICATE REQUEST-----` and `-----END NEW CERTIFICATE REQUEST-----`.

- Invoke `keytool` as follows:

```
keytool -import -file <certFileName> -alias
        <AppLogin> -keystore <keyStoreName>
```

*certFileName*

Supply the name of the file in which you copied the Intuit signed certificate.

*AppLogin*

Supply the name you specified earlier for `AppLogin` when you created your application key.

*keyStoreName*

Supply the name of your keystore.

- Supply the keystore password when prompted. If you applied a password to your application key, you will be prompted for that as well.
- When prompted whether you want to trust the certificate, respond by entering `yes`.

At this point, your application can use the signed certificate to access QBOE.

## CHAPTER 3

# REGISTERING YOUR APPLICATION

This chapter provides instructions on registering your application with IDN. Every application integrated with QBOE must be registered via the Intuit Developer Network registration site. During the registration process, an application ID is assigned to your application and the application name is taken for subsequent display to the customer during certain activities, such as authorizations.

### **IMPORTANT**

Even if your desktop application uses the QBOE connector to handle communication with QBOE, you must still register your desktop application with IDN. Otherwise, the application will not be recognized by the QBOE server.

For server applications, in addition to the application ID, a successful registration also results in the generation of a certificate that the server application needs to communicate with QBOE.

Because the registration process is different for integrated server applications and integrated desktop applications, these two types of registration are described separately.

## Registering a Desktop Application

---

### **NOTE**

In order to use the self-registration web page, you must be logged into IDN. If you are not logged in

To register a desktop application,

- Go to the web page <http://appreg.quickbooks.com>. If you are not logged into IDN you will be prompted to supply your IDN login and password. After logging in, click on the production application or beta application tab as desired, and follow the instructions provided to register your application. Supply values as follows:
  - a. In the field Application Login, supply the AppLogin value you want to use. It should be short and must contain no spaces or special characters. Customers will not see this name: it is used only internally by QBOE.
  - b. In the Company Name field, supply the name of your company.
  - c. In the Application Description field, supply the name of your application. This is the name that is displayed to customers when they are prompted to authorize your application. It should match the name you display within your application. It should be one line and fairly short.

- d. In the Contact Email field, supply the email address where you can be contacted in the event of problems with your application, program updates, patch releases, and so on.
  - e. In the Application Type field, select “desktop”.
  - f. Do not specify any of the URLs listed in the various URL fields. These are for use by server applications.
  - g. Do not supply values for the Certificate fields. These are for use by server applications
- IDN will process the registration and return the Application ID.

## Registering a Server Application

---

### **IMPORTANT**

These instructions assume that you have already generated a certificate request file as described in the previous chapter on getting certificates for server applications.

To register a server application,

- 1. Go to the web page <http://appreg.quickbooks.com>. If you are not logged into IDN you will be prompted to supply your IDN login and password. After logging in, click on the production application or beta application tab as desired, and follow the instructions provided to register your application. Supply values as follows:
  - a. In the field Application Login, supply the AppLogin value that you used when you created your application key as described in the chapter on getting certificates for server applications.
  - b. In the Company Name field, supply the name of your company.
  - c. In the Application Description field, supply the name of your application. This is the name that is displayed to customers when they are prompted to authorize your application. It should match the name you display within your application. It should be one line and fairly short.
  - d. In the Contact Email field, supply the email address where you can be contacted in the event of problems with your application, program updates, patch releases, and so on.
  - e. In the Application Type field, select “hosted”.
  - f. In the various URL fields specify the URL you want to use for each purpose. You can use the same URL for all of these or different URLs, depending on your needs.

#### *Application Subscription*

The URL at which you will receive and handle connection tickets from the QBOE server as a result of a user authorizing your application to connect to the user’s QuickBooks company.

#### *Application Cancel*

The URL at which you will receive and handle notifications regarding user cancellation of authorization for your application.

*Application Change*

The URL at which you will receive and handle session tickets returned from QBOE server as a result of a session authentication.

*Application Marketing*

The URL at which the user is given a link to your presentation of the capabilities of your integrated application.

*Application Login*

The URL at which the user is given a link to the QBOE interface when reviewing connection information between their company file and your application.

- 2. After you submit the registration, IDN processes the registration and sends a confirmation to the email address that you registered. You need to:
  - a. Click in this email as directed in the email to display the application ID and other information about your application.
  - b. In this displayed form, click on the prompt “Sign Certificate for Production“ (or Beta, if for a beta application). This displays the Sign Certificate window.
  - c. In the Sign Certificate window, copy the certificate request from the certificate request file you created earlier (see the chapter on getting certificates for server applications). Copy the entire data block, including the delimiters -----BEGIN NEW CERTIFICATE REQUEST----- and -----END NEW CERTIFICATE REQUEST-----.
  - d. IDN will then process this and return the signed certificate to you.



## CHAPTER 4

# INTEGRATING A DESKTOP APPLICATION

### **IMPORTANT**

This chapter assumes you have read Chapter 1, "Overview." In particular, it assumes you are familiar with the access control topic and the interaction between the end user and the QBOE site and the QBOE connector, as described in that chapter.

This chapter shows how to integrate a desktop application with QBOE. It describes the entries you need to make in the Windows registry in order for your application to use the QBOE connector, new in SDK 4.0, which handles all of the special tasks of communicating with the QBOE company.

Prior to SDK 4.0, QBOE-integrated desktop applications required special connection code, ticket-management code, and request posting code that was not required by other desktop applications that communicated with QuickBooks using qbXML and QBFC. This increased the difficulty of porting applications or creating new applications. Beginning with SDK 4.0, a QBOE connector is available that handles all of the special work required to communicate with QBOE.

### **NOTE**

You may still write a desktop application that does not use the QBOE connector. For information on writing such an application, see Appendix E, "Integrating Desktop Applications without Using the QBOE Connector."

Notice that this chapter does not discuss the actual content of the qbXML messages that query and manipulate QuickBooks, nor does it discuss error recovery in detail. For detailed syntax and description, see the *Onscreen Reference*. For background information and conceptual understanding, see the *QuickBooks SDK Concepts Manual*.

## What You Need to Do

---

In order to enable your desktop application to send requests to a QBOE company, you must

- Make a few entries in the Windows registry so the QBOE connector can issue the requests on behalf of your application
- Connect to the QBOE company using the new (new in SDK 4.0) standard `OpenConnection2` method call, using the connection preference `remoteQBOE`.

That is it. The rest of your application code is the same as that described in the *Developer's Guide for QuickBooks* (for qbXML applications) or the *Developer's Guide for the QBFC Library* (for QBFC applications).

## Making Required Registry Entries

---

In order to communicate with the QBOE connector, a desktop application must register itself in the Windows registry under the following Windows Registry location:

```
HKEY_LOCAL_MACHINE\Software\Intuit\QBOEXMLRP
```

The main subentry under this location that you must supply is the application name (AppName). The AppName you specify here must match the value that you use in the AppName parameter in the OpenConnection2 call.

Then, under the application name, you need to supply the following key names with corresponding RG\_SZ values:

- AppID
- AppLogin

The values you supply for the AppID and AppLoginkeys are the AppID value and the AppLogin value that you obtained when you registered your application with IDN as described in Chapter 3, “Registering Your Application.”. This is a one-time activity, normally performed during application installation. Notice that the AppID and AppLogin keys (and their values!) must be stored as subkeys to the application name:



Figure 4-1 Registry Entry Hierarchy

### **NOTE**

In the registry under AppName, you may notice that the QBOE connector automatically stores certain items for your application as well, such as the encrypted connection ticket and also the unique InstallationID that you can use if you want to implement standard SDK error recovery. (See the *QuickBooks SDK: Concepts Manual* for information on error recovery.)

## Connecting to a QBOE Company

---

After your application registers itself in the Windows registry, it can connect to a QBOE company using the `OpenConnection2` method call. The following sample function shows the typical way to make a connection, request, and end the connection:

```
Public Function post(xmlStream As String) As String
    On Error GoTo errorHandler
    Dim qbXMLRP As QBXMLRP2Lib.RequestProcessor2
    Dim ticket As String

    qbXMLRP.OpenConnection2 AppID, AppName, remoteQBOE
    ticket = qbXMLRP.BeginSession("", QBXMLRP2Lib.qbFileOpenDoNotCare)
    post = qbXMLRP.ProcessRequest(ticket, xmlStream)
    qbXMLRP.EndSession ticket
    qbXMLRP.CloseConnection
    Exit Function

errorHandler:
    post = ""
    Log.out (Err.Description)
End Function
```

If you are familiar with qbXML programming, you'll notice that this sample is strikingly similar to, in fact, identical to, what you do in a non-QBOE qbXML application, with the exception of the new `OpenConnection2` call and the use of the `remoteQBOE` as the connection type.

However, there are a couple of other less obvious items that you may want to be aware of as well, with regard to what is actually happening in the two calls `OpenConnection2` and `BeginSession`.

## In `OpenConnection2`

---

The call to `OpenConnection2` causes the instantiation of the QBOE connector, which will persist until terminated by a call to `CloseConnection`. Notice that the `AppName` supplied in this call *must match* the value stored in the Windows registry, as described under “Making Required Registry Entries.” Notice that the `AppID` parameter in the `OpenConnection2` call is ignored, however, since the `AppID` value is read from the registry by the QBOE connector.

If your application did not register itself properly in the Windows registry, for example, there is no `AppID`, the call fails with the `HRESULT` error `SDKERROR_INCOMPLETE`.

If no valid connection ticket exists for the application, for example, the end user is using it for the first time, the call to `OpenConnection2` causes the QBOE connector to send the end user to the QBOE site to authorize your application and obtain a connection ticket. The user pastes this ticket into the QBOE connector dialog as prompted and then control returns to your program.

## In BeginSession

---

In the call to `BeginSession`, notice that no company file name is specified, rather this is left empty. In QBOE, there is no notion of a company file, so this must be left empty. Moreover, if you do make a call to determine the current company file, (`GetCurrentCompanyFileName`), the response will always be “QBOE company.”

Notice the ticket returned in the call to `BeginSession`. This is the session ticket. If the user has specified the connection ticket to use session authentication (the recommended method), the call to `BeginSession` causes the QBOE connector to send the end user to the QBOE site to login to the QBOE company and obtain a session ticket. The user pastes this ticket into the QBOE connector dialog as prompted and then control returns to your program. This session ticket stays in effect until you end the session by calling `EndSession`, or until the QBOE server closes the session after a certain period of time with no user activity in the application, approximately one hour.

If the call to `BeginSession` fails, the `HRESULT` error returned is `SDKERROR_COULD_NOT_OPEN`.

## In CloseConnection

---

The call to `CloseConnection` removes (destroys) the QBOE connector instance.

## Desktop Application Security Requirements

---

This section describes security provisions that must be observed in your application code. Failure to follow these requirements may result in your application losing access to QBOE.

The following security rules must be observed by your application:

- Your application may not automate any part of the QBOE user interface, including the application authorization process.
- Your application may not request and/or store the user’s QuickBooks logon and password.
- You cannot share connection tickets or session tickets between different applications.
- If your application is a browser application, you should not allow your pages to be cached.

## Handling Errors

---

For a list of errors that can be returned during runtime access or manipulation of QuickBooks company data, please see Appendix A of this guide.

To see errors or logged items generated during runtime you can look at the file `C:\Program Files\Common Files\Intuit\QuickBooks\qbsdklog.txt`. In addition to this file, in the same location, you can also see QBOE-specific errors in the file `QBOE.txt`.

## Using Installation IDs

---

The QBOE connector automatically creates an InstallationID for your application and stores it in the registry. You can use this installation ID if you want to use the standard SDK error recovery mechanism.

See the *QuickBooks SDK: Concepts Manual* for information on error recovery.



## CHAPTER 5

# INTEGRATING A SERVER APPLICATION

This chapter shows how to integrate a server application with QuickBooks Online Edition. It describes required behavior, provides Java code samples showing how to implement the integration, and also describes the sample integration (`SDKTest.java`) provided with the SDK.

All of the sample code in this chapter is taken from that sample, which is located at:

```
QBSDK4.0\samples\qboe\server\java\qbxml\SDKTest
```

This chapter does not discuss the content of the qbXML messages that query and manipulate QuickBooks, nor does it discuss error recovery. These are described in the *QuickBooks SDK Concepts Manual*.

## Server Application Security Requirements

---

This section describes security provisions that must be observed in your application code. Failure to follow these requirements may result in your application losing access to QuickBooks Online Edition.

The following security rules must be observed by your application:

- Your application may not automate any part of the QuickBooks Online Edition user interface, including the application attachment (authorization) process.
- Your application may not request and/or store the user's QuickBooks logon and password.
- You cannot share connection tickets or session tickets between different applications. Each of your applications needs to get its own tickets.
- The connection ticket must be stored securely
- The session ticket must be kept in memory only.
- Prevent your pages from being cached, for example by using meta tags.

These requirements are described in the following subsections.

### Storing the Connection Ticket Securely

---

One way to store connection tickets is to use the Windows `wincrypt` encryption functionality (see `wincrypt.h`). A C++ sample showing how to use this to store connection tickets is provided in Appendix D.

## Using Installation IDs

---

If you expect to deploy multiple instances of the same server application, for example for load balancing, you must assign each instance a unique installation ID. (This ID value needs to be unique only for your server instances.) The QuickBooks Online error recovery mechanism requires each instance of your server application to have a different installation ID. The installation ID is included in the signon message element that is in each qbXML document posted to the QuickBooks Online Edition server.

### **NOTE**

In a load balanced implementation, it is important that all requests appear to originate from a single IP address that corresponds to the registered server name in the certificate CN field.

## Integration Task Overview

---

To integrate a server application with QuickBooks Online Edition, you must

- Obtain a certificate from IDN and import it into your keystore as described in a previous chapter in this guide.
- Register your application with IDN, as described in a previous chapter in this guide.
- Set up your application so that it uses the IDN-supplied certificate to encrypt communications sent to QuickBooks Online Edition and to decrypt the responses.
- Support user authorization (subscribe) and cancellation (unsubscribe). This involves
  - > UI components in the form displayed to the user
  - > Server code that handles the user-issued command and redirects to the QuickBooks Online Edition server
  - > Server code that specifies a callback URL to handle the resulting connection tickets and cancellation notices, respectively.
- Get a session ticket at the start of every application session with a particular company. This involves the constructing and posting of a signon message.
- Support session authentication, in which the user must login to the desired company (via the QuickBooks Online Edition session authentication site) each time the user begins a session. This involves:
  - > Server code that checks for the “Session Authentication required” failure during an attempt to obtain a session ticket.
  - > Server code that handles the above failure by performing a Get at the QuickBooks Online Edition authentication URL so the user can log into QuickBooks there.
  - > Server code that specifies a callback URL to handle the resulting session ticket sent there by the QuickBooks session authentication server.
- Provide server code that assembles the various user-supplied information into qbXML requests and places these in a qbXML document to be posted to QuickBooks to carry

out the desired actions in QuickBooks. (Each document must have the signon message containing the session ticket.)

- Provide server code that posts the qbXML documents to the QuickBooks Online Edition URL and handles responses, including the performing of error recovery.
- Implement the links desired at the Application Marketing and Application Login callback URLs you specified when you registered your application.

Some of these tasks are described in more detail later in this chapter.

## Setting Up Encryption/Decryption Using the IDN Certificate

---

Using SSL at your web server in conjunction with the IDN certificate is a transparent way to implement the encryption/decryption required for communication with QuickBooks Online Edition.

For example, suppose your server application is implemented via a Java servlet. You just add a few lines of code in your servlet's init method to specify the use of SSL, your keystore, and your keystore password, and the IDN certificate will be used automatically for all communication with QuickBooks Online Edition.

### **NOTE**

To use the SSL functionality provided by Java, you need to import the `javax.net` and `javax.net.ssl` packages supplied in the Java JSSE API. You also need to import `com.sun.net.ssl.*`.

### Example: Implementing Encryption/Decryption in a Servlet

---

The following code shows the required lines in the servlet init method:

```
public void init(ServletConfig config) throws ServletException {
    super.init(config);

    // Set up the SSL properties to find the keystore correctly
    System.setProperty("javax.net.debug", "all");
    System.setProperty("javax.net.ssl.keystore",
        "/var/tomcat4/keys/keystore");
    System.setProperty("javax.net.ssl.keystorePassword", "SDKTest");
    System.setProperty("java.protocol.handler.pkgs",
        "com.sun.net.ssl.internal.www.protocol");

    / Make sure java is configured to handle HTTPS by adding the
    // appropriate SSL provider to the security manager.
    java.security.Security.addProvider(new
        com.sun.net.ssl.internal.ssl.Provider());
}
```

In the sample code above, `/var/tomcat4/keys/keystore` is the keystore used in the sample. In your own servlet, you would replace this with the path to your own keystore. Similarly, `SDKTest` is the sample's password: you would specify your own keystore password.

## Posting to the Data Exchange URL

---

The following is the URL string you must use when posting XML to a QuickBooks company:

```
https://webapps.quickbooks.com/j/AppGateway
```

## Supporting User Authorization (Getting Connection Tickets)

---

Supporting user authorization consists of three parts:

- Enabling the user to start the authorization process from your main form in the web browser.
- Responding to the user action in your server code by fully constructing a valid URL and sending the user there.
- Handling the callback from the authorization site in your server code at the Application Subscription URL you specified when you registered your application.

These three required pieces are described in the following sections.

### Enabling User Authorization in Your Main Form

---

In your main form displayed in the user's web browser, you must provide a means for the user to start the authorization process for your application. For example, you could supply a Submit button that causes a POST to be handled by your application.

### Constructing the URL Required for Authorization

---

In your server code, you handle the user's authorization action by fully constructing the authorization URL to the QuickBooks Online Edition authorization site and then sending the user to that site, where the user will complete the authorization process. The fully constructed URL contains the location of the authorization page along with required information about your application and an identifier (`appid`) that uniquely identifies the user within the company.

The following line shows a sample construction:

```
String subscribeURL = https://login.quickbooks.com/j/qbn/sdkapp/  
sessionauth2 + "?appid=" + myAppID + "&serviceid=" + myServiceID +  
"&appid=" + myAppData + "&url=" + callbackURL;
```

where

*-https://login.quickbooks.com/j/qbn/sdkapp/sessionauth2*  
is the location of the authorization page.

*-myAppID*  
is the appID assigned to your application when you registered the application.

*-myServiceID*  
is the value 2004

*myAppData*  
is the unique (unique within the application) name of the user for whom the connection ticket is intended. This is usually some sort of ID internal to your application.

*callbackURL*  
is your callback URL with the following appended:

```
"?appdata="+myAppData
```

After you send the user to the authorization site, the user is prompted by the authorization server and is led through the authorization process. After the process is complete, the user is notified of the success, and a connection ticket is returned to your Application Subscription callback URL. If the user authorized only a single session, a session ticket is returned.

The session ticket returned to the authorization callback from this activity is not the valid session ticket. For security reasons, the callback must use this first session ticket to perform a Get obtain the valid (second) session ticket. How this is done is shown in the following section.

## Handling the Authorization Callback

---

Upon user authorization of your application, the QuickBooks authorization server posts the connection ticket and/or session ticket to the Application Subscription URL you specified when you registered with IDN.

### Handling Session Tickets in the Callback

If a session ticket is returned to the callback, the callback must use that session ticket in a GET to obtain the second, valid, session ticket. This is done for security reasons. The URL used in this GET is constructed as follows:

```
https://login.quickbooks.com/j/qbn/sdkapp/connauth" + "?appid=" + appid +  
"&serviceid=2004" + "&conntkt=" + conntkt + "&sessiontkt=" + myTkt;
```

where

*-https://login.quickbooks.com/j/qbn/sdkapp/connauth*  
is the location of the confirmation page.

*-appid*

is the appID assigned to your application when you registered the application.

*-conntkt*

is connection ticket from the authorization.

*myTkt*

is the (first) session ticket obtained from the authorization.

### **Example: Performing a GET to Obtain the Second Session Ticket**

In the following snippet, the callback checks for the presence of the session ticket, and if there is one, constructs the confirmation URL, and invokes GET using that URL. The first four characters in this response are the session ticket, so this is parsed and stored.

```
if (tkst != null) {
log("In processQBTicket, need to process new sess ticket");
String mySubSvr = "https://login.quickbooks.com";
String loginPath = "/j/qbn/sdkapp/connauth";
String myTkt = tkst;
String authURL = mySubSvr + loginPath + "?appid=" + appid
    + "&serviceid=2004" + "&conntkt=" + conntkt
    + "&sessiontkt=" + myTkt;
log("\tauthURL = " + authURL);
try {
String resp = doRequest(authURL, "", "GET");
log("Result from authURL is: " + resp);
String status = resp.substring(0, 3);
if (status.compareTo("000") != 0) {
String err = "Problem updating session ticket: "
    + status;
log(err);
} else {
resp = resp.substring(4);
storeTicket(appdata, "sessiontkt", resp);
}
} catch (Exception e) {
log("process Session Ticket caught exception: " + e.toString());
}
}
```

For connection tickets, your server code handling this post needs to extract the connection ticket and the appdata and store them for subsequent use. (The appdata is the appdata value that was supplied in the constructed URL used to send the user to the authorization site.)

### **Example: Extracting and Saving the Connection Ticket**

The following sample method is invoked from the servlet doPost, which handles the authorization site posts.

In this sample, the HTTP request (the authorization post) is queried for the appdata and the connection ticket parameters and the connection ticket is stored in a file. The appdata value is used in the file name, and the connection ticket is written to the file. However, you aren't required to store the connection ticket this way.

```

protected void processQBTicket(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, java.io.IOException {
    PrintWriter outmain = null;
    log("in processQBTicket");
    try {
        PrintWriter out = response.getWriter();
        outmain = out;
        String conntkt = null;
        String appdata = null;
        String tkt = null;
        String authid = null;
        String appid = null;

        /*
        * here we work around a known bug in the current release of
        * QuickBooks Online edition, the callback does not set the
        * content-type header, so HttpServletRequest does not parse
        * the parameters correctly. Once we get a good content-type
        * we can use the simple code, otherwise we have to parse the
        * request body ourselves
        */
        if (request.getContentType() != null) {
            /*
            * Bug is gone, we can use HttpServletRequest to our
            * advantage
            */
            conntkt = request.getParameter("conntkt");
            appdata = request.getParameter("appdata");
            appid = request.getParameter("appid");
            // tkt should be there only if we have single session
            // authorization
            tkt = request.getParameter("sessiontkt");
            log("got parameters");
        }
        } else {
            /*
            * Bug is there, parse the request body ourselves
            */
            BufferedReader httpbody = request.getReader();
            String line;
            StringBuffer body = new StringBuffer();
            while ((line = httpbody.readLine()) != null) {
                body.append(line);
                log(line);
            }

            // pull the user, ticket, etc. from the body of the post
            StringTokenizer outerTokenizer = new StringTokenizer
                (body.toString(), "&");
            while (outerTokenizer.hasMoreTokens()) {
                String temp = outerTokenizer.nextToken();
                StringTokenizer innerTokenizer = new StringTokenizer
                    (temp, "=");
                String variable = innerTokenizer.nextToken();
                String value = innerTokenizer.nextToken();
            }
        }
    }
}

```

```

        if (variable.equals("appdata")) {
            appdata = value;
        } else if (variable.equals("conntkt")) {
            conntkt = value;
        } else if (variable.equals("tkr")) {
            tkt = value;
        } else if (variable.equals("authid")) {
            authid = value;
        } else if (variable.equals("appid")) {
            appid = value;
        }
    }
}
/*
 * log the data we got for debugging purposes, this is NOT
 * something you would want to do in production since it
 * would put cleartext tickets in the system logs.
 */
log("\tgot conntkt = " + conntkt);
log("\tgot appdata = " + appdata);
log("\tgot session tkt = " + tkt);
log("\tgot authid = " + authid);
log("\tgot appid = " + appid);

/*
 * Check for unexpected conditions
 */
if (appdata == null || (appdata.compareTo("")==0)) {
    throw new Exception("No appdata for ticket received");
}
if (conntkt != null) {
    storeTicket(appdata, "conntkt", conntkt);
} else {
    conntkt = retrieveTicket(appdata, "conntkt");
}
if (appid == null) {
    appid = retrieveTicket(appdata, "appid");
}
if (tkt != null) {
log("In processQBTicket, need to process new sess ticket");
String mySubSvr = "https://login.quickbooks.com";
String loginPath = "/j/qbn/sdkapp/connauth";
String myTkt = tkt;
String authURL = mySubSvr + loginPath + "?appid=" + appid
    + "&serviceid=2004" + "&conntkt=" + conntkt
    + "&sessiontkt=" + myTkt;
log("\tauthURL = " + authURL);
try {
    String resp = doRequest(authURL, "", "GET");
    log("Result from authURL is: " + resp);
    String status = resp.substring(0, 3);
    if (status.compareTo("000") != 0) {
        String err = "Problem updating session ticket: "
            + status;
        log(err);
    } else {
        resp = resp.substring(4);
    }
}

```

```

storeTicket(appdata,"sessiontkt",resp);
}
} catch (Exception e) {
    log("process Session Ticket caught exception: " + e.toString());
}
}
}
/*
 * QuickBooks online will basically ignore our response, but it
 * is worth providing one in case we want to debug this code
 * by faking a callback and sending our own data...
 */
out.println("<html>");
out.println("<head>");
out.println("<title>SDKTest Ticket Recieved</title>");
out.println("</head>");
out.println("<body>");
out.println("Received tickets from QBOE");
out.println("</body>");
out.println("</html>");
response.setStatus(response.SC_OK);
} catch (Exception e) {
    /*
     * handle something unexpected happening by logging it and
     * sending an appropriate response value to QuickBooks online
     */
    log("Got error receiving ticket: " + e.toString());
    response.setStatus(response.SC_ACCEPTED);
    if (outmain != null) {
        printError(outmain, "receiving ticket: " + e.toString());
    }
}
}
/*
 * close our response stream
 */
if (outmain != null) {
    outmain.close();
}
}
}

```

## Getting the Session Ticket

---

### **IMPORTANT**

If the connection ticket doesn't require session authentication, you could simply just use the connection ticket instead of the session ticket. Because of the performance advantages, however, this chapter always shows the use of session tickets. However, in some cases you may need to use only the connection ticket.

The connection ticket is obtained once and remains valid until the user cancels it. However, in order to perform data exchange with QuickBooks Online Edition, if use session tickets (or if these are mandatory because the company owner has specified session authorization), you must obtain a new session ticket for a user each time the user starts a new session in a company.

There are three considerations regarding getting a session ticket:

- Where to put the code?
- What code is required to get the session ticket?
- How do you handle the case where session authentication is required?

Each of these is described below.

## Where in the Server Application Should You Get Session Tickets?

You must add code that checks for an existing session ticket and gets a ticket if there isn't one. Where should you place this code? You might want to add this session ticket check at every point where a qbXML request-bearing document is being posted to the QuickBooks Online Edition. For example, you could construct a general purpose and central qbXML request post method, and place the session ticket check in this method.

## How Do You Get Session Tickets?

To get a session ticket, you need to build a qbXML document that contains the SignonMsgsRq element and the nested SignonAppCertRq element inside it. You post this to QuickBooks Online Edition and parse the response for the session ticket returned in the response.

However, if session authentication is required, the response will not include a session ticket, but an error code 2020 "Session Authentication required" will be returned instead. You need to check for this error in the response and send the user to the session authentication site to login to QuickBooks Online Edition, including a callback URL when you send the user to the site. The session ticket in this case is posted to the callback URL that you sent.

### **IMPORTANT**

Session tickets sent to the callback URL must be handled in a special way, that is, they must be used only to obtain a second session ticket, as described previously in this chapter under "Handling Session Tickets in the Callback" (page 31).

Table 5-1 on page 37 and Table 5-2 on page 37 describe the contents of the SignonAppCertRq and SignonAppCertRs (response) element, respectively.

Table 5-1 SignonAppCertRq Values

<b>Element Name</b>	<b>Description</b>
ClientDateTime	The ClientDateTime is the current system time, which provides a timestamp for the signon request.
ApplicationLogin	This is the name of the application that you supplied when you registered to obtain an application ID.
ConnectionTicket	The ticket posted by QuickBooks to your application as a result of a user authorization.
InstallationID	You must supply a value for this. If multiple instances of the server are running, you need to create a unique installation ID for each instance. QuickBooks Online Edition uses this for error recovery.
Language	Currently, supply the value "English".
AppID	The application ID assigned when you registered your application with Intuit Developer Network.
AppVer	Application version string

Table 5-2 SignonAppCertRs Values

<b>Element Name</b>	<b>Description</b>
ServerDateTime	The current system time at the QuickBooks Online Edition Site server taken at the time the request was processed.
StatusCode	The value indicating success or failure: if failure, the code indicates the nature of the failure, with various values possibly returned.  See Appendix A for a list of possible values.
SessionTicket	If session authentication is not in effect, the session ticket is returned in the response.
StatusSeverity	Indicates level of failure.
StatusMessage	Provides a user-readable indication of the failure.

## Java Samples: Getting the Session Ticket

The following two sample methods show a good way to get the session ticket. The first method is `getSessionTicket`, which is invoked prior to posting a qbXML document to QuickBooks. (See the sample server application `SDKTest` for the complete Java program.) The second method is `updateSignonAppCertBlock`, which constructs the actual message.

### Sample 1: Getting Tickets, Issuing Ticket Request, Handling Request Response

In `getSessionTicket`, form parameters from a multipart form are extracted into strings. The session ticket string is evaluated: if a session ticket already exists, it is returned to the caller to be used in the qbXML post. If no ticket exists, `updateSignonAppCertBlock` is invoked to build the `SignonAppCertRq` message.

The method `getSessionTicket` then posts this message to QuickBooks and parses the response for the session ticket. Notice the check for the failure due to session authentication requirements. If this occurs, `getManualSessionTicket` is invoked to send the user to the session authentication site to login to QuickBooks, and the session ticket will be posted by QuickBooks to callback URL that is specified when the user is sent.

```
private String getSessionTicket(HttpServletRequest request,
                               MultipartRequest mpRequest,
                               HttpServletResponse response)
    throws ServletException, java.io.IOException {
    log("in getSessionTicket");

    /*
     * Get our formData so we can build a correct SignonAppCertRq
     */
    FormData formData = new FormData(mpRequest);
    if (formData.invalid) {
        // Something bad in the form, gripe and return to the form.
        // Error message should be in FormError...
        String redirectURL = "https://" + request.getServerName() + "/"
            + "SDKTest";
        log(" redirectURL = " + redirectURL);
        log(" formData is invalid, sending to redirectURL");
        log("set FormError attribute to: \" + formData.FormError + "\"");
        HttpSession session = request.getSession();
        session.setAttribute("SDKTest.FormError", formData.FormError);
        response.sendRedirect(redirectURL);
        return null;
    }

    // Form data good...
    String myRqSvr = formData.rqSvr;
    String myAppLogin = formData.appLogin;
    String myAppID = formData.appID;
    String myServiceID = formData.serviceID;
    String myAppData = formData.appData;
    String conntkt = formData.conntkt;
    String sesstkt = formData.sesstkt;
    String requestPath = "j/AppGateway";
    String rqAddr = myRqSvr + "/" + requestPath;
    if ((sesstkt != null) && (sesstkt.compareTo("") != 0)) {
```

```

    return sesstkt;
}

// Assemble the correct header for the QuickBooks Online Edition SDK
String preamble = "<?xml version=\"1.0\"?>\n";
preamble += "<!DOCTYPE QBXML PUBLIC '-//INTUIT//DTD QBXML QBO 2.0//EN'
'http://apps.quickbooks.com/dtds/qbxmlops20.dtd'>\n";

// Build an empty request
String sessRq = preamble + "<QBXML>\n</QBXML>\n";

// Add the SignonAppCertRq aggregate to the empty request and put
// on the header...
sessRq = preamble + updateSignonAppCertBlock (sessRq,myAppLogin,
    myAppID, conntkt);

// Now we're ready to post our request.
try {
    log("About to post request to get a session ticket");
    String sessResp = doRequest(rqAddr, sessRq, "POST");
    log("Back from doRequest");
    // parse the response from QuickBooks
    Document doc = parseXML(sessResp);
    Element top = doc.getDocumentElement();
    // Look for the Signon response
    NodeList responses = top.getElementsByTagName("SignonMsgsRs");
    if (responses.getLength() > 0) {
        // got a response, look for the AppCert response
        Element resp = (Element)responses.item(0);
        NodeList dtResponses = resp.getElementsByTagName
            ("SignonAppCertRs");
        if (dtResponses.getLength() >0) {
            // got an AppCert response, check the status code
            // for an error...
            Element dtResponse = (Element)dtResponses.item(0);
            String dtStatusCode =dtResponse.getAttribute("statusCode");
            if (dtStatusCode.compareTo("0") != 0) {
                // error!
                if (dtStatusCode.compareTo(sessionAuthReqNew) == 0) {
                    // Session authentication error, user must login
                    getManualSessionTicket(request, mpRequest, response);
                } else {
                    // Should display the message, will fix after B1
                }
            } else {
                // Successful AppCert signon, look for the session
                // ticket in <SessionTicket> element. -- Note: there
                // might be a bug here, I probably need to look for
                // the authID as well and join them with a ":"
                NodeList sessTkts = dtResponse.getElementsByTagName
                    ("SessionTicket");
                if (sessTkts.getLength() >0) {
                    Element sessTkt = (Element)sessTkts.item(0);
                    Node child;
                    org.w3c.dom.Text tktData = (org.w3c.dom.Text)
                        sessTkt.getFirstChild();
                    if (tktData != null) {

```

```

        sesstkt = tktData.getData();
    }
}
}
}
} catch (Exception e) {
    System.out.println("Caught exception: " + e.toString());
    log("getSessionTicket Caught exception: " + e.toString());
}
return sesstkt;
}

```

## Sample 2: Building the SignonAppCertRq Message

In this method, a SignonMsgsRq element is constructed that contains a fully constructed SignonAppCertRq element. This will be posted to obtain a session ticket. Notice that this is an update method so it does check the supplied request string for an existing SignonMsgsRq element and discards unneeded elements if one exists.

The xmlwalk object and its methods are defined in the SDKTest sample.

```

private String updateSignonAppCertBlock(String request, String appLogin,
String appID, String conntkt)
{
    log("in updateSignonAppCertBlock");
    try {
        // First, parse what we have...
        Document doc = parseXML(request);
        Element top = doc.getDocumentElement();

        // Now check for the required SignonMsgsRq element
        NodeList signonRqs = top.getElementsByTagName("SignonMsgsRq");
        if (signonRqs.getLength() > 0) {
            //we have a SignonMsgsRq element, update the element...
            Element signonRq = (Element)signonRqs.item(0);

            //Get the list of all the Signon*Rq elements embedded in the
            //request XML
            NodeList signonDTRqs = signonRq.getElementsByTagName
                ("SignonDesktopRq");
            NodeList signonAppRqs = signonRq.getElementsByTagName
                ("SignonAppCertRq");
            NodeList signonTktRqs = signonRq.getElementsByTagName
                ("SignonTicketRq");
            int i;

            // Remove any SignonDesktopRq elements, since that is just
            // for desktop applications
            for(i=0; i<signonDTRqs.getLength(); i++) {
                Node node = signonDTRqs.item(i);
                signonRq.removeChild(node);
            }

            // Remove any SignonTicketRq elements, we want an AppCertRq
            for(i=0; i<signonTktRqs.getLength(); i++) {

```

```

        Node node = signonTktRqs.item(i);
        signonRq.removeChild(node);
    }

    // now update the SignonAppCertRq elements with our data
    for(i=0; i<signonAppRqs.getLength(); i++) {
        Node node = signonAppRqs.item(i);
        Node child;
        for (child = node.getFirstChild(); child != null; child =
            child.getNextSibling()) {
            String elemName = child.getNodeName();
            if (elemName.compareTo("ApplicationLogin") == 0) {
                replaceNodeText(child,appLogin);
            } else if (elemName.compareTo("AppID") == 0) {
                replaceNodeText(child,appID);
            } else if (elemName.compareTo("ConnectionTicket") == 0) {
                replaceNodeText(child,conntkt);
            } else if (elemName.compareTo("ClientDateTime") == 0) {
                replaceNodeText(child,getClientDate());
            }
        }
    }
}

// if there isn't a SignonAppCertRq, build one..
if (signonAppRqs.getLength() < 1) {
    Node firstChild = signonRq.getFirstChild();
    signonRq.insertBefore(buildSignonAppCertRq(doc,
        appLogin,appID,conntkt), firstChild);
}

// and return the whole modified XML request from top down
StringBuffer xmlBuf = new StringBuffer();
xmlwalk(top, xmlBuf);
return xmlBuf.toString();
} else {
    // we don't have a SignonMsgsRq, insert one.
    Element signonMsgsRq = doc.createElement("SignonMsgsRq");
    signonMsgsRq.appendChild(buildSignonAppCertRq(doc,
        appLogin, appID, conntkt));
    Node firstChild = top.getFirstChild();
    top.insertBefore(signonMsgsRq, firstChild);
    StringBuffer xmlBuf = new StringBuffer();
    xmlwalk(top, xmlBuf);
    return xmlBuf.toString();
}
} catch (Exception e) {
    log("Caught error in updateSignonAppCertBlock: " + e.toString());
}
}

// if we got here, something went wrong, so clear the request
return null;
}
}

```

## Supplying Client Date/Time in the Required Format

---

When you construct the various signon message elements required during the attempt to obtain a session ticket and during normal data exchange posts, you must supply a client time value. (That is, your server's time, since your server is a client of QuickBooks Online Edition.) You must use a specific time format.

The following Java method transforms the current time into the required format, using the Java Date and SimpleDateFormat classes:

```
private String getClientDate() {
    SimpleDateFormat f = new SimpleDateFormat();
    StringBuffer out = new StringBuffer();
    f.applyPattern("yyyyMMdd");
    f.format(new Date(), out, new FieldPosition(1));
    out.append("T");
    f.applyPattern("hhmmss");
    f.format(new Date(), out, new FieldPosition(1));
    return out.toString();
}
```

## Supporting User Cancellation of Authorization

---

Supporting user cancellation consists of three parts:

- Enabling the user to start the cancellation process from your main form in the web browser.
- Responding to the user action in your server code by fully constructing a valid URL and performing a Get on it.
- Handling the callback from the cancellation site in your server code at the Application Cancel URL you specified when you registered your application.

These three required pieces are described in the following sections.

### Enabling User Cancellation in Your Main Form

---

In your main form displayed in the user's web browser, you must provide a means for the user to start the cancellation process for your application. For example, you could supply a Submit button that causes a POST to be handled by your application.

### Constructing the URL Required for Cancellation

---

In your server code, you handle the user's cancel action by fully constructing the cancellation URL to the QuickBooks Online Edition cancellation site and then sending the user to that site, where the user will complete the process.

The fully constructed URL contains the location of the cancellation page along with required information about your application and an identifier (appdata) that uniquely identifies the user within the company.

The following line shows a sample construction:

```
String subscribeURL = https://login.quickbooks.com/j/qbn/sdkapp/cancel +
"?appid=" + myAppID + "&serviceid=" + myServiceID + "&appdata=" +
myAppData;
```

where

*-The first part*  
is the location of the authorization page.

*-myAppID*  
is the appID assigned to your application when you registered the application.

*-myServiceID*  
is the value 2004.

*myAppData*  
is the name that was specified when the connection ticket was created.

After you get the cancellation page, the user is prompted by the server and is led through the cancellation process. After the process is complete, the user is notified of the success.

## Handling the Cancellation Callback

---

Upon user cancellation of authorization, the QuickBooks authorization server posts a request to the Application Cancel URL you specified when you registered with IDN.

Your server code handling this post needs to extract the connection ticket and the appdata and use them for a lookup to delete the existing connection ticket from your persistent store.

## About the SDKTest.java Sample

---

Figure 5-1 on page 44 provides a quick overview of the SDKTest servlet functionality.

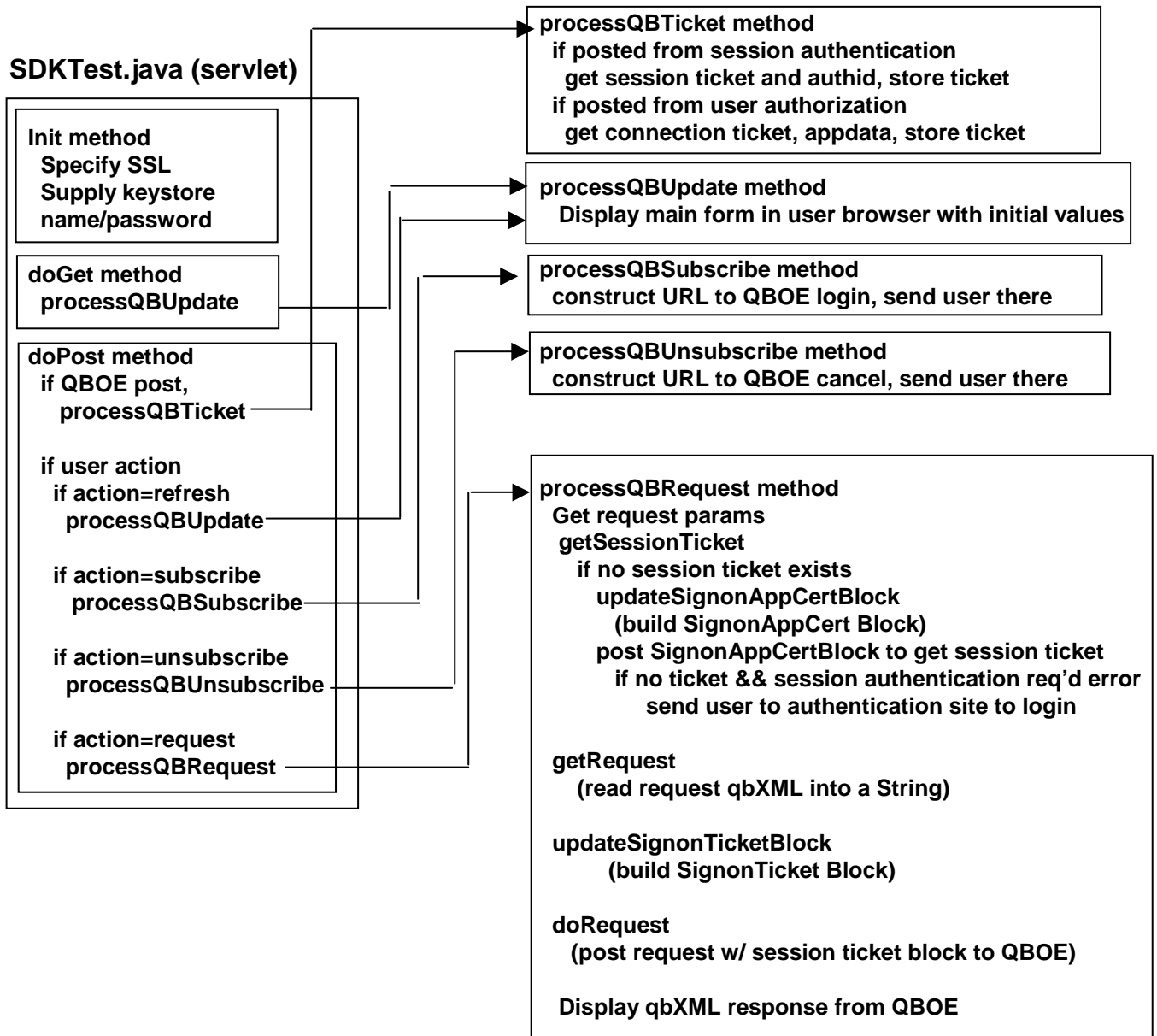


Figure 5-1 SDKTest Servlet Functional Overview

# Handling Errors

---

Your application must handle three types of errors:

- Standard HTTPS errors resulting from your application's attempt to use network resources.
- QuickBooks Online Edition errors resulting from your application's attempt to connect to it
- QuickBooks Online Edition errors resulting from your application's attempt to manipulate data in the user's company.

For lists and descriptions of the standard HTTP/S errors, please consult any of the many references on the subject. If you are new to this area, you may want to visit the web site [www.w3.org](http://www.w3.org), which is the web site for the World Wide Web consortium.

## **NOTE**

Certain standard errors will be returned by QuickBooks Online Edition under some circumstances. For example, malformed XML data will be rejected with a standard error 400, Bad Request.

For a list of the errors that can be returned during runtime access or manipulation of QuickBooks company data, please see Appendix A of this guide. Please see the *QuickBooks SDK: Concepts Manual* for details on how to handle these types of errors.

For a list of QuickBooks Online Edition connection-related errors, please see the following list:

*2000*

Authentication failed -- Invalid login name or password / certificate / ticket

*2010*

Unauthorized

*2020*

Session Authentication required

*2030*

Unsupported signon version

*2040*

Internal error

The sample code provided with the QuickBooks SDK shows how to handle many of these connection-related errors.



# APPENDIX A

## STATUS CODES RETURNED IN RESPONSES

Each qbXML response message returned from QuickBooks Online Edition includes a statusCode, a statusSeverity, and statusMessage element.

The table below lists the status codes that can be returned, along with descriptions.

Code	Meaning	Explanation
0	The QuickBooks server processed the request successfully.	Status OK
1	No match.	A query request did not find a matching object in QuickBooks.
500	One or more objects cannot be found	The query request has not been fully completed. There was a required element ("fieldValue") that could not be found in QuickBooks.
501	Object not in this qbXML specification	Unable to represent objectName "fieldValue" in this version of the qbXML spec.
510	Object cannot be returned	Unable to return object.
530	Unsupported field	The field "fieldName" is not supported by this implementation.
531	Unsupported enum value	The enum value "fieldValue" in the field "fieldName" is not supported by this implementation.
600	No cleared state to return	(For error recovery; no message is returned.)
1000	Internal error	There has been an internal error when processing the request.
1010	System not available	System not available
1030	Unsupported message	This request is not supported by this implementation.
1060	Invalid request ID	The request ID "fieldValue" is invalid, possibly too long, max 50 chars.
2000	Authentication failed	Signon failed. QuickBooks error message: fieldValue
2010	Access not authorized	Not authorized to access the server.
3000	Invalid object ID	The given object ID "fieldValue" in the field "fieldName" is invalid.
3010	Invalid Boolean	There was an error when converting the boolean value "fieldValue" in the field "fieldName"
3020	Invalid date	There was an error when converting the date value "fieldValue" in the field "fieldName"
3030	Invalid date range	Invalid date range: From date is greater than To date.

<b>Code</b>	<b>Meaning</b>	<b>Explanation</b>
3031	Invalid string range	The "From" or "To" values in the provided fieldName are invalid.
3035	Invalid time interval	There was an error when converting the time interval "fieldValue" in the field "fieldName"
3040	Invalid amount	There was an error when converting the amount "fieldValue" in the field "fieldName"
3045	Invalid price	There was an error when converting the price "fieldValue" in the field "fieldName"
3050	Invalid percentage	There was an error when converting the percent "fieldValue" in the field "fieldName"
3060	Invalid quantity	There was an error when converting the quantity "fieldValue" in the field "fieldName"
3070	String too long	The string "fieldValue" in the field "fieldName" is too long.
3080	Invalid string	The string "fieldValue" is invalid.
3085	Invalid number	There was an error when converting the number "fieldValue" in the field "fieldName"
3090	Invalid object name	There was an error when storing "fieldValue" in the "fieldName" field.
3100	Name is not unique	The name "fieldValue" of the list element is already in use.
3101	Resulting amount too large	Multiplying the rate and the quantity results in an amount that exceeds the maximum allowable amount.
3110	Invalid enum value	The enumerated value "fieldValue" in the field "fieldName" is unknown.
3120	Object not found	Object "fieldValue" specified in the request cannot be found.
3121	OwnerID not found	Data Extension Definitions specified by OwnerID fieldValue not found for this object type.
3130	Parent reference not found	There is an invalid reference to a parent "fieldValue" in the objectName list.
3140	Reference not found	There is an invalid reference to QuickBooks fieldName "fieldValue" in the objectName.
3150	Missing required element	There is a missing element "fieldName."
3151	Invalid element for request	Cannot use the element "fieldName" in this request.
3152	Invalid enum value for this request	The enumerated value "fieldValue" may not be used in the element "fieldName" in this request.
3153	Element conflict in request	This error is returned whenever there is a conflict in the elements in the request. Each element has valid value, but their combination becomes invalid.
3160	Object cannot be deleted	Cannot delete the object specified by the id = "fieldValue."
3161	Cannot delete before closing date	An attempt was made to delete a fieldValue with a date that is on or before the closing date of the company. If you are sure you really want to do this, please ask a user with Admin privileges to remove the password for editing transactions on or before to closing date (this setting is in the Accounting Company Preferences), then try again.

<b>Code</b>	<b>Meaning</b>	<b>Explanation</b>
3162	Not allowed in multi-user mode	This operation is not allowed in multi-user mode.
3170	Object cannot be modified	There was an error when modifying a fieldValue.
3171	Cannot modify before closing date	An attempt was made to modify a fieldValue with a date that is on or before the closing date of the company. If you are sure you really want to do this, please ask a user with Admin privileges to remove the password for editing transactions on or before to closing date (this setting is in the Accounting Company Preferences), then try again.
3172	Cannot modify prior to last condense	An attempt was made to modify a fieldValue with a date that is on or before the last inventory condensed date.
3173	Related object deleted or modified	The related fieldName transaction object fieldValue was deleted or modified.
3175	Object is in use	There was an error adding, modifying or deleting fieldValue because it is already in use.
3176	Related object is in use	The related fieldName transaction object fieldValue is already in use." // "specified by the ID" is appended to fieldValue if necessary.
3177	Duplicate AppliedToTxn IDs	The transaction object "fieldValue" may only be provided once in this request.
3180	Object cannot be added	There was an error when saving a fieldValue.
3185	Object cannot be voided	Cannot void the object specified by the id = "fieldValue"
3190	Cannot clear required element	Cannot clear the element in the fieldName field.
3200	Outdated edit sequence	The provided edit sequence "fieldValue" is out-of-date.
3205	Invalid address	There was an error when composing an address in "fieldValue"
3210	Other validation error	The "fieldName" field has an invalid value "fieldValue"
3230	Status rollback	The request has been rolled-back.
3231	Status unprocessed	The request has not been processed.
3240	Time creation mismatch	Object "fieldValue" specified in the request cannot be found.
3250	Feature not enabled	This feature is not enabled or not available in this version of QuickBooks.
3260	Insufficient permissions	Insufficient permission level to perform this action.
3261	Application has no sensitive data permission	The integrated application has no permission to access sensitive data.
3270	Missing posting account	Missing posting account.

<b>Code</b>	<b>Meaning</b>	<b>Explanation</b>
3280	Item type mismatch	The item "fieldValue" cannot be used in this line item. It does not have a correct type.
9000	Host processing delay	(For error recovery; no message is returned.)
9001	Invalid checksum	(For error recovery; no message is returned.)
9002	No stored response found	(For error recovery; no message is returned.)
9003	Reinitialization problem	(For error recovery; no message is returned.)
9004	Invalid message ID	(For error recovery; no message is returned.)

# APPENDIX B

## SUPPORTED QBXML OPERATIONS

This appendix lists the operations and queries that are supported by QuickBooks Online Edition. It also lists limitations that are in effect for some of these operations, particularly the Add operations.

### List Object Support

---

The Add, Query, and Modify operations are supported for the following list objects:

- Account
- Class
- Customer
- Employee
- ItemService
- PaymentMethod
- Term
- Vendor

### Transaction Object Support

---

The Add and Query operations are supported for the following transaction objects:

- Bill
- Check
- CreditCardCharge
- CreditCardCredit
- CreditMemo
- Invoice
- JournalEntry
- ReceivePaymentAdd
- SalesReceipt
- TimeTracking
- VendorCredit

TxnDel is also supported for the transactions that were created by the same application that is attempting to delete them.

In SDK 4.0, the following new aggregates are supported for CreditMemo, Invoice, and SalesReceipt:

- Added the following support for SalesReceiptAdd, InvoiceAdd, and CreditMemoAdd
  - > Added the DiscountLineAdd/Ret aggregates
  - > Added the SalesTaxLineAdd/Ret aggregates
  - > Added the ShippingLineAdd/Ret aggregates
- Added the IsTaxable Boolean type to the following:
  - > InvoiceLineAdd/Ret
  - > SalesReceiptLineAdd/Ret
  - > CreditMemoLineAdd/Ret
- Added support for APAccountRef in the following:
  - > BillAdd/Ret
  - > VendorCreditAdd/Ret

## Data Extensions Support for Company Object

---

The following data extension operations are supported for the company object:

- DataExtDefAdd
- DataExtDefMod
- DataExtDefDel
- DataExtAdd (You can add up to 20k of data, different than the Desktop limit of 4k aggregate.)
- DataExtDel
- DataExtMod

## List Query Support

---

The following filters are available to List queries:

- ListID
- FullName
- DateCreated
- DateModified
- MaxReturned
- Name

## Transaction Query Support

---

### **IMPORTANT**

Beginning with SDK 4.0, QBOE supports an enhanced list of transaction query filters. Please refer to the Onscreen Reference under each transaction query for the filters that are supported.

In the various transaction queries supported by QBOE, the default MaxReturned value is 1000. This may not always be optimal for performance reasons. Furthermore, prior to SDK 4.0, it was not possible to set up queries for those instances where you wanted to be able to get more than 1000 transactions because QBOE does not return more than 1000 transactions for a query.

Beginning with SDK 4.0, QBOE supports transaction filters that will enable you to set up your queries the way that you need to in order to break up large queries into more manageable chunks. These filters are listed and documented under each type of transaction query in the *Onscreen Reference*.

## Other Query Support

---

The following queries are also supported:

- EntityQuery
- HostQuery
- CompanyQuery
- ReceivePaymentQuery

## Unsupported Objects Related to Add Operations

---

The following table lists objects that are not supported or that have limited support in the various Add operations.

Type of Add Operation	Not Supported	Limited Support	Not Implemented
Account	IsActive, LastCheckNumber, BankNumber		
Customer	IsActive, Contact, AltContact, CustomerTypeRef, SalesRepRef, SalesTaxCodeRef, ItemSalesTaxRef, ResaleNumber, AccountNumber, CreditLimit, JobStatus, JobStartDate, JobProjectedEndDate, JobEndDate, JobDesc, JobTypeRef, SSN		Notes
Employee	IsActive, EmployeeType, Gender, BirthDate, AccountNumber		

Type of Add Operation	Not Supported	Limited Support	Not Implemented
Vendor	IsActive, Contact, AltContact, VendorTypeRef, CreditLimit		Notes
ItemService	IsActive, SalesTaxCodeRef, SalesAndPurchase		
Class	IsActive		
Bill	ItemLine, ItemGroupLine Note: In SDK 4.0, new aggregates were created for QBOE that may provide alternative functionality to the above listed unsupported items.		
VendorCredit	ItemLine, ItemGroupLine Note: In SDK 4.0, new aggregates were created for QBOE that may provide alternative functionality to the above listed unsupported items.		
Check	Address in Add, ItemLine, ItemGroupLine		
CreditCardCharge	ItemLine, ItemGroupLine		
CreditCardCredit	ItemLine, ItemGroupLine		
SalesReceipt	header ClassRef(?), IsPending, DueDate, SalesRepRef, FOB, ItemSalesTaxRef, CustomerSalesTaxCodeRef, SalesReceiptLineGroup, detail SalesTaxCodeRef  Note: In SDK 4.0, new aggregates were created for QBOE that may provide alternative functionality to the above listed unsupported items. These new aggregates are: DiscountLineAdd/Ret SalesTaxLineAdd/Ret ShippingLineAdd/Ret and the IsTaxable boolean	ShipMethodRef, CustomerMsgRef	

Type of Add Operation	Not Supported	Limited Support	Not Implemented
CreditMemo	<p>header ClassRef(?), IsPending, PONumber, DueDate, SalesRepRef, FOB, ItemSalesTaxRef, CustomerSalesTaxCodeRef, CreditMemoLineGroup, detail SalesTaxCodeRef</p> <p>Note: In SDK 4.0, new aggregates were created for QBOE that may provide alternative functionality to the above listed unsupported items. These new aggregates are:</p> <p>DiscountLineAdd SalesTaxLineAdd ShippingLineAdd and the IsTaxable boolean</p>	ShipMethodRef, CustomerMsgRef	
Invoice	<p>header ClassRef(?), IsPending, IsFinanceCharge, DueDate, SalesRepRef, FOB, ItemSalesTaxRef, CustomerSalesTaxCodeRef, SuggestedDiscountAmount, SuggestedDiscountDate, InvoiceLineGroup, detail SalesTaxCodeRef</p> <p>Note: In SDK 4.0, new aggregates were created for QBOE that may provide alternative functionality to the above listed unsupported items. These new aggregates are:</p> <p>DiscountLineAdd SalesTaxLineAdd ShippingLineAdd and the IsTaxable boolean</p>	ShipMethodRef, CustomerMsgRef	
TimeTracking	PayrollItemWageRef		



# APPENDIX C

## SIGNON MESSAGES AND RESPONSES XML

### **IMPORTANT**

This information is included only for server applications. Desktop application no longer need to supply signon messages if they use the OpenConnection2 method call on the RequestProcessor, with the connection preference parameter value of remoteQBOE.

Each qbXML document that is posted to the QuickBooks Online Edition server must include a fully constructed signon message. The responses returned from the server therefore also include a signon response.

Instructions for constructing these signon messages and handling their responses are provided in the chapters in this guide about integrating a desktop application and integrating a server application.

The following sample XML shows how the signon messages and the corresponding responses are structured.

```
<?xml version="1.0" ?>
<!--
-->
<!--
===== -->
<!-- INTUIT CONFIDENTIAL.
-->
<!-- Copyright 2001-2002 Intuit Inc. All rights reserved.
-->
<!-- Use is subject to the terms specified at:
-->
<!--          http://developer.intuit.com/legal/devsite_tos.html
-->
<!--
-->
<!--
===== -->
<!--
-->
<!-- Sample data for dtd: qbxmlso20.dtd
-->
<!--
-->
<!-- This dtd contains requests/responses for the Signon message set.
-->
<!--
-->
<!-- Comments use the following abbreviations:
-->
```

```

<!-- QBD stands for the QuickBooks Desktop SDK
-->
<!-- QBOE stands for the QuickBooks Online Edition SDK
-->
<!--
-->
<!-- Message set Signon contains the following requests and responses:
-->
<!--
-->
<!-- Signon (AppCert, Desktop and Ticket)
-->
<!--
-->
<!-- This means that Signon has, for example, 3 separate requests.
-->
<!-- They are: SignonAppCert, SignonDesktop and SignonTicket
-->
<!--
-->
<QBXML>
  <SignonMsgsRq>
    <!-- SignonAppCertRq contains 1 optional attribute: 'requestID' -->
    <SignonAppCertRq requestID = "UIDTYPE">
      <ClientDateTime>DATETIME</ClientDateTime>
      <ApplicationLogin>STRTYPE</ApplicationLogin>
      <ConnectionTicket>STRTYPE</ConnectionTicket>
      <InstallationID>IDTYPE</InstallationID> <!-- opt -->
      <Language>STRTYPE</Language>
      <AppID>STRTYPE</AppID>
      <AppVer>STRTYPE</AppVer>
    </SignonAppCertRq>
    <!-- SignonDesktopRq contains 1 optional attribute: 'requestID' -->
    <SignonDesktopRq requestID = "UIDTYPE">
      <ClientDateTime>DATETIME</ClientDateTime>
      <ApplicationLogin>STRTYPE</ApplicationLogin>
      <ConnectionTicket>STRTYPE</ConnectionTicket>
      <InstallationID>IDTYPE</InstallationID> <!-- opt -->
      <Language>STRTYPE</Language>
      <AppID>STRTYPE</AppID>
      <AppVer>STRTYPE</AppVer>
    </SignonDesktopRq>
    <!-- SignonTicketRq contains 1 optional attribute: 'requestID' -->
    <SignonTicketRq requestID = "UIDTYPE">
      <ClientDateTime>DATETIME</ClientDateTime>
      <SessionTicket>STRTYPE</SessionTicket>
      <AuthID>IDTYPE</AuthID> <!-- opt -->
      <InstallationID>IDTYPE</InstallationID> <!-- opt -->
      <Language>STRTYPE</Language>
      <AppID>STRTYPE</AppID>
      <AppVer>STRTYPE</AppVer>
    </SignonTicketRq>
  </SignonMsgsRq>
  <SignonMsgsRs>
    <!-- SignonAppCertRs contains 4 attributes -->
    <!-- 'requestID' is optional -->
    <!-- 'statusCode' is required -->

```

```

<!-- 'statusSeverity' is required -->
<!-- 'statusMessage' is optional -->
<SignonAppCertRs requestID = "UUIDTYPE"
    statusCode = "INTTYPE"
    statusSeverity = "STRTYPE" statusMessage = "STRTYPE">
    <ServerDateTime>DATETIME</ServerDateTime>
    <SessionTicket>STRTYPE</SessionTicket> <!-- opt -->
</SignonAppCertRs>
<!-- SignonDesktopRs contains 4 attributes -->
<!-- 'requestID' is optional -->
<!-- 'statusCode' is required -->
<!-- 'statusSeverity' is required -->
<!-- 'statusMessage' is optional -->
<SignonDesktopRs requestID = "UUIDTYPE"
    statusCode = "INTTYPE" statusSeverity = "STRTYPE"
    statusMessage = "STRTYPE">
    <ServerDateTime>DATETIME</ServerDateTime>
    <SessionTicket>STRTYPE</SessionTicket> <!-- opt -->
</SignonDesktopRs>
<!-- SignonTicketRs contains 4 attributes -->
<!-- 'requestID' is optional -->
<!-- 'statusCode' is required -->
<!-- 'statusSeverity' is required -->
<!-- 'statusMessage' is optional -->
<SignonTicketRs requestID = "UUIDTYPE" statusCode = "INTTYPE"
    statusSeverity = "STRTYPE" statusMessage = "STRTYPE">
    <ServerDateTime>DATETIME</ServerDateTime>
    <SessionTicket>STRTYPE</SessionTicket> <!-- opt -->
</SignonTicketRs>
</SignonMsgsRs>
</QBXML>

```



# APPENDIX D

## SUPPORTED QBXML OPERATIONS FOR QUICKBOOKS SIMPLE START EDITION

This appendix lists the subset of qbXML objects and operations that are supported in the SDK for use on the QuickBooks Simple Start edition.

- All QBOE-supported items listed in Appendix B “Supported qbXML Operations” EXCEPT for the following UNSUPPORTED items:
  - > Vendors with an opening balance (Vendor must be a new vendor with NO opening balance: if it has opening balance, it won’t work)
  - > Bill
  - > VendorCredit
  - > TimeTracking



# APPENDIX E

## INTEGRATING DESKTOP APPLICATIONS WITHOUT USING THE QBOE CONNECTOR

This appendix shows how to integrate a desktop application with QuickBooks Online Edition without using the QBOE connector. Writing an application for use with the QBOE connector is far simpler than following the technique described in this appendix, but the method described in this chapter does provide more control over the dialogs that are posted during the process of obtaining connection and session tickets.

This appendix describes required setup procedures, required behavior, and suggests ways to build and parse qbXML documents. It does not discuss the actual content of the qbXML messages that query and manipulate QuickBooks, nor does it discuss error recovery in detail, which are described in the *QuickBooks SDK Concepts Manual*.

A desktop application is any executable residing on the client system, such as a standard executable, or a browser, or an ActiveX control.

### Application Interaction with QuickBooks Online Edition Sites

The following is a list of the different types of interactions that your desktop application should support:

- Interactions with the authorization site
- Interactions with the cancellation site
- Interactions with the session authentication (login security) site
- Interactions with the QuickBooks Online Edition application site, where the user's company is accessed by your application

Each of these is described in more detail in the following sections.

#### Interactions with the QuickBooks Online Edition Authorization Site

As shown in Figure 1-1, your application must provide some way for the user to authorize your application to connect to the user's QuickBooks company. A typical way to do this is via a UI component such as a button. The figure shows how to handle the button click, namely by launching a web browser and getting the authorization page for the user. Upon completing the authorization, the user pastes the resulting connection ticket into your application.

## Desktop Application

## QuickBooks Online Edition

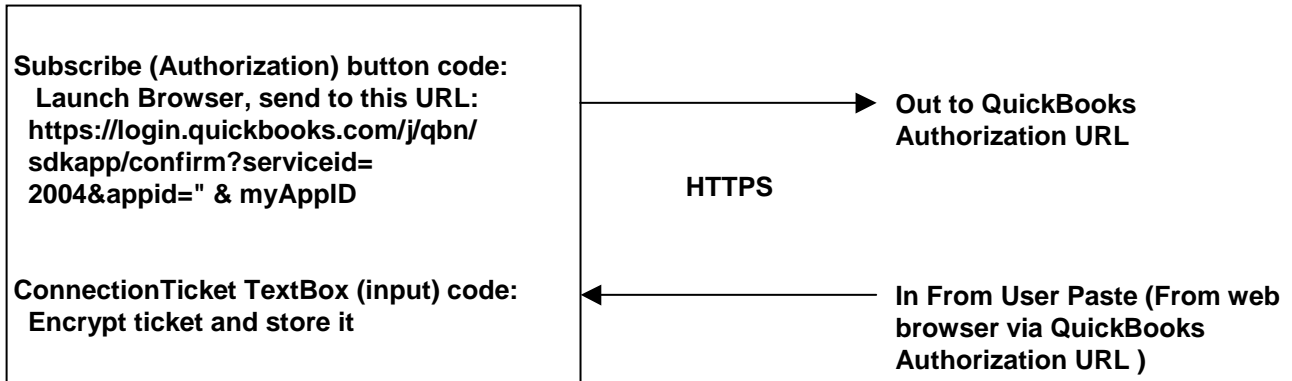


Figure 1-1 Authorization Site Interaction

### The Authorization URL

The following is the URL string you must use:

```
https://login.quickbooks.com/j/qbn/sdkapp/confirm?serviceid= 2004&appid=" & myAppID"
```

where myAppID is your application ID. See “Supporting User Authorization (Getting Connection Tickets)” (page 69) for more information on appID.

### Getting the Connection Ticket from the User

After the user completes the connection interview on the authorization site, authorizing your application, a connection ticket is returned from QuickBooks to the browser. As shown in the figure, you provide a text box where the user can copy this ticket, then you must store it securely.

## Interactions with the QuickBooks Online Edition Cancellation Site

As shown in Figure 1-2, your application must provide some way for the user to cancel the authorization, via a command button. The figure shows how to handle the button click, namely by launching a web browser and getting the cancellation page for the user.

## Desktop Application

## QuickBooks Online Edition

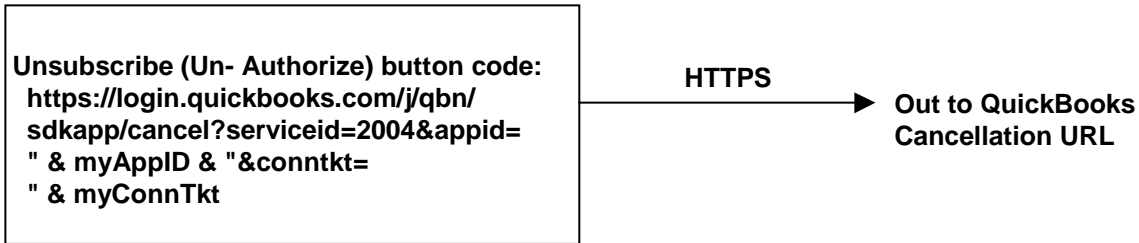


Figure 1-2 Cancellation Site Interaction

### The Cancellation URL

The following is the URL string you must use:

```
https://login.quickbooks.com/j/qbn/sdkapp/cancel?serviceid=2004&appid= " & myAppID & "&conntkt= " & myConnTkt"
```

where myAppID is your application ID, and myConnTkt is the connection ticket being cancelled.

## Interactions with the Session Authentication Site

---

As illustrated in Figure 1-3, at the beginning of a session under a connection ticket that requires session authentication, any attempt to post to the QuickBooks data exchange URL will return the error code 2020 in the response message. Your application must check for this and send the user to the authentication site using the first authentication URL to log into QuickBooks when this error occurs.

After the user logs on at this URL, the user receives a session ticket which the user must paste into a UI input component you provide, for example a text box. You should store this in memory only.

Notice that for security reasons, this session ticket must be replaced “behind the scenes” with a hidden Get at the authentication site using the second authentication URL. The response to this Get contains the real session ticket to be used in the current session.

## Desktop Application

## QuickBooks Online Edition

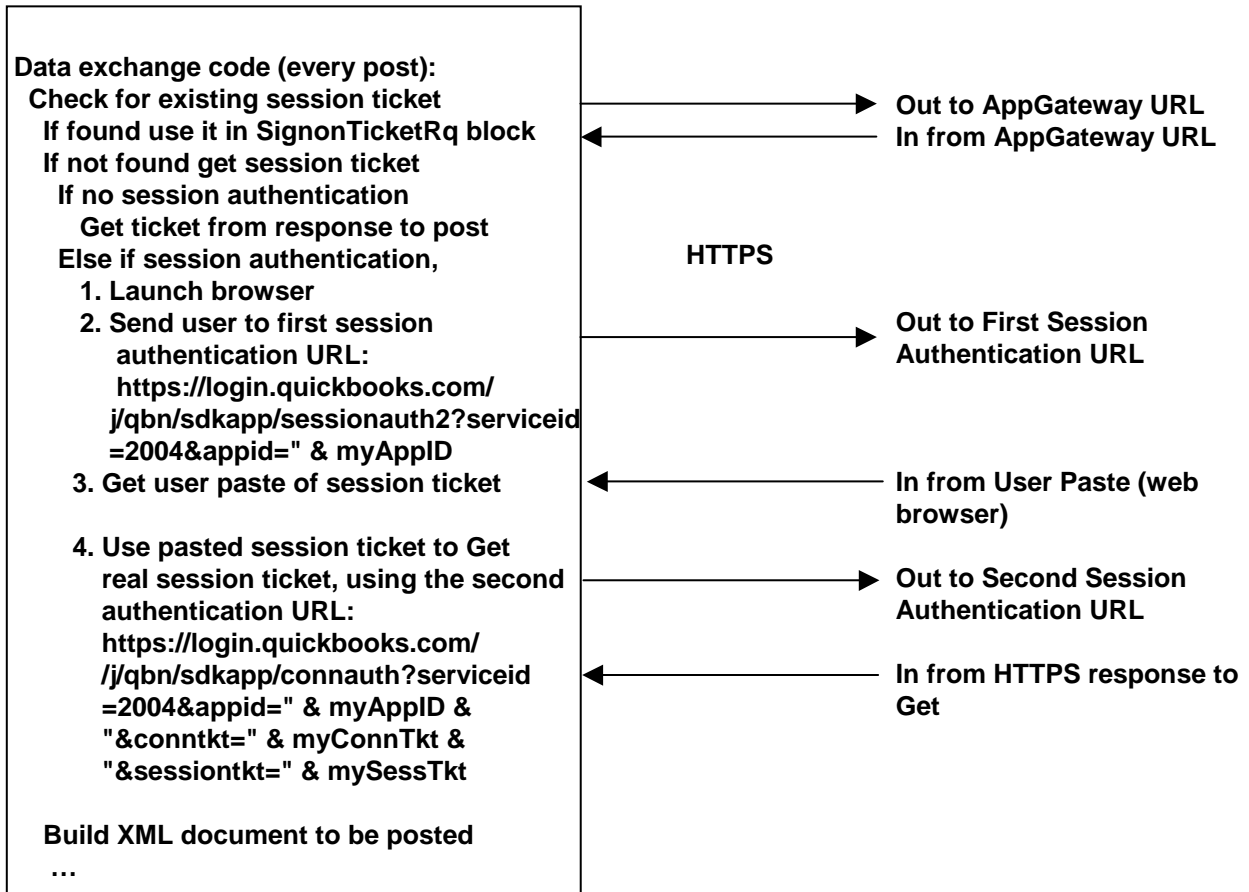


Figure 1-3 Session Authentication Site Interaction

### The First Session Authentication URL

The following is the URL string you must use when you send the user to the authentication site:

```
https://login.quickbooks.com/j/qbn/sdkapp/sessionauth2?serviceid=2004&appid=" & myAppID"
```

where myAppID is your application ID. Notice that this URL has changed from the one previously used, with the connection ticket information being removed for enhanced security. (The old URL string will still work, but it is deprecated.)

### The Second Session Authentication URL

The following is the URL string you must use after the user pastes in the session ticket obtained from the first session authentication URL:

`https://login.quickbooks.com/j/qbn/sdkapp/connauth?serviceid=2004&appid=" & myAppID & "&conntkt=" & myConnTkt & "&sessiontkt=" & mySessTkt`

where myAppID is your application ID, myConnTkt is the connection ticket and mySessTicket is the ticket that the user pasted in from the first session authentication URL.

## Normal Data Exchange with the QuickBooks Company

“Data exchange” occurs whenever your application sends qbXML requests to the QuickBooks company to get data or to send new data, including the initial post to get the session ticket. As shown in Figure 1-4, this exchange must be performed in an HTTPS post of an XML document containing a fully constructed SignonTicketRq element and validly formed qbXML requests.

If required data is missing from the ticket request element or if elements in the QBXML element are invalid, QuickBooks will return an error in the response.

### Desktop Application

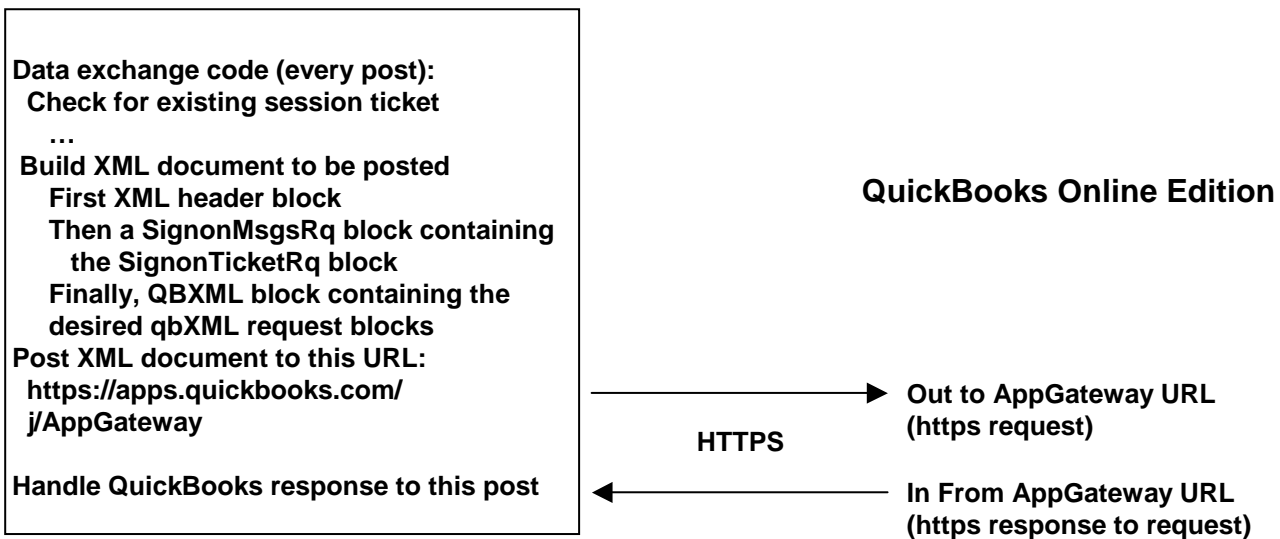


Figure 1-4 Data Exchange Site Interaction

## Posting to the Data Exchange URL

The following is the URL string you must use when posting XML to a QuickBooks company:

`https://apps.quickbooks.com/j/AppGateway`

## Desktop Application Security Requirements

---

This section describes security provisions that must be observed in your application code. Failure to follow these requirements may result in your application losing access to QuickBooks Online Edition.

The following security rules must be observed by your application:

- Your application may not automate any part of the QuickBooks Online Edition user interface, including the application attachment (authorization) process.
- Your application may not request and/or store the user's QuickBooks logon and password.
- You cannot share connection tickets or session tickets between different applications. Each of your applications must get its own tickets.
- The connection ticket must be stored securely
- The session ticket must be kept in memory only.
- If your application is a browser application, you should not allow your pages to be cached.

The requirement to store connection tickets securely is described in more detail below.

### Storing the Connection Ticket Securely

---

A recommended way to store connection tickets is to use the Windows wincrypt encryption functionality (see wincrypt.h). A C++ sample showing how to use this to store connection tickets is provided in this appendix under the heading "Using wincrypt to Store Connection Tickets."

To use Windows encryption in Visual Basic, please refer to the web site <http://www.cryptovb.com> for more information about applying the Microsoft CryptoAPI to protect the connection ticket.

### Using Installation IDs

---

If you want to use the QuickBooks Online Edition error recovery mechanism, then you must implement a way to create globally unique IDs for each application installation. Then, during application runtime, whenever you post a qbXML document, you must include that installation ID inside the SignonTicketRq element that accompanies all qbXML posts.

If you don't use unique installation IDs, your application will still work, with the exception of any error recovery that depends on unique IDs. However, you must supply some value for the installation ID.

See the *QuickBooks SDK: Concepts Manual* for information on error recovery.

## Integration Task Overview

---

To integrate a desktop application with QuickBooks Online Edition, you must

1. Register your application with IDN, as described in Chapter 3 of this guide.
2. Implement a way to create a globally unique ID for each application installation.
3. Provide UI components and code to support user authorization (subscribe) of your application so it can communicate with the user's company.
4. Provide UI components and code to support user cancellation (unsubscribe) of that authorization.
5. Get a session ticket at the start of every application session with a particular company. This involves the posting of a fully constructed SignonDesktopRq element inside a SignonMsgsRq signon message.
6. Support session authentication, in which the user must login to the desired company (via the QuickBooks Online Edition session authentication site) each time the user begins a session.
7. Whenever you post a qbXML document to QuickBooks Online Edition, add a fully constructed SignonTicketRq element inside a SignonMsgsRq element and make sure these precede the QBXMLMsgsRq element containing the qbXML requests.
8. Parse the response from QuickBooks to handle the returned data and perform any required error recovery.

The remainder of this chapter provides details for most of the above listed tasks.

## Supporting User Authorization (Getting Connection Tickets)

---

On your main form or in a menu pull-down in your main application form, you need to provide a way to let the user initiate the authorization (subscription) process for your application. For example, in response to a command button click, you could launch a web browser and send the user to the authorization URL.

Be aware that at the successful conclusion of a user authorization, the QuickBooks Online Edition authorization page displays the connection ticket which the user must copy into your application. So, you also need to supply a UI component, such as a text box to accept that ticket.

### VB Sample: Subscribe Button-Click Handler

---

The following code responds to the user's push of an authorization command button.

```
Private Sub SubscribeBtn_Click()  
  
    Dim loginSvr As String  
  
    Dim myAppID As String
```

```

loginSvr = SubServerText.Text

myAppID = AppID.Text

Dim loginURL As String

Rem Build URL, open browser for the user

Rem to complete the interview in QuickBooks Online Edition.

loginURL = loginSvr & "/j/qbn/sdkapp/
             confirm?serviceid=2004&appid=" & myAppID"

SubscriptionURLText.Caption = loginURL

showURL (loginURL)

End Sub

```

In the sample button click handler, the URL is assembled from the official QuickBooks login URL that already exists in the sample application as a text box default (SubServerText), the value of which is:  
<https://login.quickbooks.com>.

To this value is appended the relative path at the login server that takes you to the actual authorization page and provides certain required parameters:  
 /j/qbn/sdkapp/confirm?serviceid=2004&appid=" & myAppID".

In this appended text string, the value /j/qbn/sdkapp/confirm? will be part of every authorization URL because it is the relative path to the authorization page at the QuickBooks Online Edition login server.

*serviceid*

The service ID value is always 2004.

*appid*

The application ID assigned to your application when you registered it with IDN.

Once the authorization URL is fully constructed, showURL is invoked to launch the web browser and send the user to the QuickBooks authorization page. See “Sending Users to QuickBooks Online Edition Sites” (page 86) for information on showURL.

What is not shown in the sample above is the text box that accepts the connection ticket from the user’s pasting of it there. You need to supply this text box and store that value persistently in a secure manner, for example encrypting it.

## Getting a Session Ticket

---

To post qbXML requests to QuickBooks Online Edition, you should use a session ticket, although this is only absolutely required when the connection ticket requires session authentication. It is easier to code for that, due to the nature of the error checking and also

due to the way the XML documents are constructed. So every document containing qbXML requests should have the session ticket inside a fully formed SignonTicketRq element that itself is inside a SignonMsgsRq element.

A new session ticket must be obtained in every new application session with QuickBooks Online Edition. So, you can't just store and use that ticket as you can with the connection ticket.

The connection ticket is used to obtain the session ticket. That is, you post a request (SignonDesktopRq) that contains the connection ticket to QuickBooks. The response (SignonDesktopRs) from QuickBooks contains either a session ticket if session authentication is not required, or an error message indicating that session authentication is required.

## VB Sample: Getting a Session Ticket

---

The following sample function shows a good way to get the session ticket:

```
Private Function getSessionTicket() As String
Dim reqSvr As String
    Dim myAppID As String
    Dim myConnTkt As String
    Dim mySessTkt As String
    reqSvr = ReqServerText.Text
    myAppID = AppID.Text
    myConnTkt = ConnTktText.Text
    mySessTkt = SessTktText.Text
    getSessionTicket = ""

    If Len(mySessTkt) > 0 Then
        If (gotManualTicket Or Not updatedManualTicket) Then
            Dim loginSvr As String
            loginSvr = SubServerText.Text
            Dim authURL As String
            authURL = loginSvr & "/j/qbn/sdkapp/connauth?serviceid=
                2004&appid=" & myAppID
            authURL = authURL & "&conntkt=" & myConnTkt & "&sessiontkt=" &
                mySessTkt
            Dim http As New XMLHTTP40
            http.open "GET", authURL, False
            http.send
            Dim resp As String
            Dim status As String
            resp = http.responseText
            status = Mid(resp, 1, 3)
            If (Not status = "000") Then
                MsgBox "Problem updating session ticket: " & status
                getSessionTicket = ""
                Exit Function
            End If
            resp = Mid(resp, 4)
            getSessionTicket = resp
            SessTktText.Text = resp
        End If
    End If
End Function
```

```

    gotManualTicket = False
    updatedManualTicket = True
Else
    getSessionTicket = mySessTkt
End If
Else
Else
    Rem No session ticket from user, get one automatically
    Dim signon As String
    Rem Build an empty request
    signon = updateSignonDesktopBlock
        ("<QBXML>" & vbCrLf & "</QBXML>")
    signon = "<?xml version=" & Chr(34) & "1.0" & Chr(34) & "?>" &
        vbCrLf & "<!DOCTYPE QBXML PUBLIC '-//INTUIT//DTD QBXML
        QBO 2.0//EN' 'http://apps.quickbooks.com/dtlds/
        qbxmlops20.dtd'>" & vbCrLf & signon
    Rem show the request just for grins
    MessageText.Text = signon
    Dim signonURL As String
    signonURL = reqSvr & "/j/AppGateway"
    Dim response As String
    Rem POST the request
    response = postReq(signonURL, signon)
    Rem parse response and see if we have an error or a ticket
    Dim doc As DOMDocument40
    Set doc = New DOMDocument40
    doc.async = False
    doc.validateOnParse = False
    If (doc.loadXML(response)) Then
        Dim top As IXMLDOMNode
        Set top = doc.documentElement
        Dim responses As IXMLDOMNodeList
        Rem All we care about is the SignonMsgsRs part
        Set responses = top.selectNodes("SignonMsgsRs")
        If responses.length > 0 Then
            Dim rs As IXMLDOMElement
            Set rs = responses.Item(0)
            Dim dtResponses As IXMLDOMNodeList
            Rem Then we need the response to the SignonDesktopRq
            Set dtResponses = rs.selectNodes("SignonDesktopRs")
            If dtResponses.length > 0 Then
                Dim dtRs As IXMLDOMElement
                Set dtRs = dtResponses.Item(0)
                Dim dtStatusCode As String
                Rem Look at the status code
                dtStatusCode = dtRs.getAttribute("statusCode")
                If "0" <> dtStatusCode Then
                    Rem We got an error, is it the one we expect?
                    Dim dtStatusMsg As String
                    dtStatusMsg = dtRs.getAttribute
                        ("statusMessage")
                    Rem Always check statusCode, statusMessage
                    Rem message may change but the code won't
                    If "2020" = dtStatusCode Then
                        MsgBox ("Session ticket required, please
                        click OK and login to QBOE in the internet
                        explorer window which will appear")
                    End If
                End If
            End If
        End If
    End If
End If
End If

```

```

        getSessionTicket = getManualSessionTicket
    Else
        Rem Not the error we expected, bail.
        MsgBox ("Got error return from QuickBooks
        Online Edition: " & dtStatusMsg)
        MessageText.Text = response
    End If
Else
    Rem No error, must have a good
    Rem Session ticket.
    Dim dtSessionTkt As IXMLDOMNode
    Rem Get the <SessionTicket> aggregate node
    Set dtSessionTkt = dtRs.selectSingleNode
    ("SessionTicket")
    If Not dtSessionTkt Is Nothing Then
        SessTktText.Text = dtSessionTkt.Text
        getSessionTicket = dtSessionTkt.Text
    End If
End If
End If
End If
Else
    Rem Something VERY unexpected happened -- we could not
    Rem parse response from QuickBooks, show the problem.
    With doc.parseError
        MsgBox ("Error parsing response from ticket request "
        & .errorCode & ": " & .filepos & .Line &
        .linepos & .reason & .srcText & .url)
    End With
    Rem need to go the manual route.
    getSessionTicket = getManualSessionTicket()
End If
End If
End Function

```

This sample function checks a text box property value for an existing session ticket, for purposes of convenience. (This ticket should be kept in memory only.)

### **Possibility 1: No Session Ticket Exists Yet**

In the preceding sample function, if there is no session ticket (that is, `Len(mySessTkt) > 0` is false), in the outermost Else clause the sample function `updateSignonDesktopBlock` is invoked to build an XML document containing the connection ticket in a `SignonDesktopRq` request. See “Building the `SignonDesktopRq` Element to Get a Session Ticket” (page 75) for information on building the `SignonDesktopRq` element.

After the request is posted to QuickBooks (in `postReq`), the response is parsed using Microsoft XML DOM features. If there is no error the function returns the session ticket. Notice that the element `SignonDesktopRs` in the response is parsed for the Session Authentication Required error code (2020). This error means that the user must login to the QuickBooks company to get the session ticket. Accordingly, `getManualSessionTicket` is invoked to send the user to the session authentication site to log on.

## Possibility 2: Session Ticket Exists, No Session Authentication

In the preceding sample function, if there is a session ticket (that is, `Len(mySessTkt) > 0` is true), in the outermost If clause, there is a check to determine whether the session ticket is a preliminary ticket, that is, it was obtained from the first session authentication URL, and is the ticket pasted into the application by the user. (The sample global variable `gotManualTicket` will be true if the session ticket is the preliminary ticket and the sample global variable `updatedManualTicket` will be false.)

If the session ticket is not the result of a session authentication, the session ticket is simply returned by the function to the caller.

## Possibility 3: Preliminary Session Ticket Exists via Session Authentication

In the preceding sample function, if the sample global variable `gotManualTicket` is true or the sample global variable `updatedManualTicket` is false, then the session ticket was obtained from the first session authentication URL, and therefore a preliminary ticket that must be used to obtain the final ticket.

As shown in the sample function, the second authentication URL is constructed:

```
Dim authURL As String

authURL = https://login.quickbooks.com/j/qbn/sdkapp/
connauth?serviceid=2004&appid=" & myAppID & "&conntkt=" & myConnTkt &
"&sessiontkt=" & mySessTkt
```

and an HTTP Get is performed using the data supplied in the URL string.

```
Dim http As New XMLHTTP40

http.open "GET", authURL, False

http.send
```

Finally, the HTTP response is parsed for the "real", or final session ticket. Notice that the first three characters in the response are the status code, with the code "000" indicating success. The ticket starts as the fourth character in the response. The sample function extracts this ticket and returns it to the caller:

```
Dim resp As String
Dim status As String
resp = http.responseText
status = Mid(resp, 1, 3)
If (Not status = "000") Then
    MsgBox "Problem updating session ticket: " & status
    getSessionTicket = ""
    Exit Function
End If
resp = Mid(resp, 4)
getSessionTicket = resp
SessTktText.Text = resp
gotManualTicket = False
updatedManualTicket = True
```

## Building the SignonDesktopRq Element to Get a Session Ticket

---

In order to get a session ticket, you post an XML document to the QuickBooks data exchange URL that contains a SignonMsgsRq element with a fully formed nested SignonDesktopRq element. This initial post normally contains no qbXML requests. If successful, the response to this post contains the session ticket. See "Parsing the SignonDesktopRs Response for the Session Ticket" (page 79) for information on parsing this response.

### Sample XML for the SignonDesktopRq Element

The following XML sample shows how the SignonDesktopRq element must be constructed inside the parent SignonMsgsRq element.

```
<SignonMsgsRq>
  <SignonDesktopRq requestID = "UUIDTYPE">
    <ClientDateTime>DATETIME</ClientDateTime>
    <ApplicationLogin>STRTYPE</ApplicationLogin>
    <ConnectionTicket>STRTYPE</ConnectionTicket>
    <InstallationID>IDTYPE</InstallationID> <!-- opt -->
    <Language>STRTYPE</Language>
    <AppID>STRTYPE</AppID>
    <AppVer>STRTYPE</AppVer>
  </SignonDesktopRq>
</SignonMsgsRq>
```

The elements, attributes, and their values within the SignonDesktopRq element are described in Table 1-1.

Table 1-1 SignonDesktopRq Values

Element Name	Description
ClientDateTime	The ClientDateTime is the current system time, which provides a timestamp for the signon request. See "Supplying Client Date/Time in the Required Format" (page 86).
ApplicationLogin	This is the name of the application that you supplied when you registered to obtain an application ID.
ConnectionTicket	The ticket copied by the user into your application as a result of the application authorization process.
InstallationID	It is recommended that you create a unique installation ID for each instance of your desktop application. That value must be retrieved and inserted in this element. QuickBooks Online Edition requires this for error recovery, which is a feature recommended for applications.
Language	Currently, supply the value "English".
AppID	The application ID assigned when you registered your application with Intuit Developer Network.
AppVer	Application version string

## VB Sample: Building a SignonDesktopRq Message

The following two sample functions show how to build a signon request to obtain a session ticket. The first function (updateSignonDesktopBlock) does various types of checking and operations based on the supplied request string, then if necessary invokes the second function (buildSignonDesktopRq) to actually build the element:

```
Private Function updateSignonDesktopBlock(request As String) As String
    Dim doc As DOMDocument40
    Set doc = New DOMDocument40
    doc.async = False
    doc.validateOnParse = False
    Rem Parse the request
    If (doc.loadXML(request)) Then
        Rem Parse was successful, check if we have what we expect
        Dim top As IXMLDOMNode
        Set top = doc.documentElement
        If ("QBXML" <> top.baseName) Then
            MsgBox ("Missing top-level <QBXML> in request")
            pdateSignonDesktopBlock = Null
            Exit Function
        End If
    End If
```

```

Rem Now find the SignonMsgsRq aggregate node
Dim signonMsgsRq As IXMLDOMNode
Set signonMsgsRq = top.selectSingleNode("SignonMsgsRq")
If Not signonMsgsRq Is Nothing Then
    Rem Got SignonMsgsRq, make sure it has a SignonDesktopRq
    Dim signonDTRqs As IXMLDOMNodeList
    Dim signonAppRqs As IXMLDOMNodeList
    Dim signonTktRqs As IXMLDOMNodeList
    Set signonDTRqs = signonMsgsRq.selectNodes
("SignonDesktopRq")
    Set signonAppRqs = signonMsgsRq.selectNodes
("SignonAppCertRq")
    Set signonTktRqs = signonMsgsRq.selectNodes
("SignonTicketRq")
    Dim curNode As IXMLDOMNode
    Rem no <SignonAppCertRq> aggregates, for hosted apps only
    Set curNode = signonAppRqs.nextNode
    Do While (Null <> curNode)
        signonMsgsRq.removeChild (curNode)
        Set curNode = signonAppRqs.nextNode
    Loop
    Rem no <SignonTicketRq> aggregates
    Set curNode = signonTktRqs.nextNode
    Do While (Null <> curNode)
        signonMsgsRq.removeChild (curNode)
        curNode = signonTktRqs.nextNode
    Loop
    Rem Update or build the <SignonDesktopRq> aggregate
    Set curNode = signonDTRqs.nextNode
    If (Null <> curNode) Then
        Rem We have a SignonDesktopRq, update the info in it
        Do While (Null <> curNode)
            Dim child As IXMLDOMNode
            Set child = curNode.firstChild
            Do While (Null <> child)
                Dim name As String
                Select Case child.nodeName
                    Case Is = "ClientDateTime"
                        child.Text = getClientDate
                    Case Is = "ApplicationLogin"
                        child.Text = AppLoginText.Text
                    Case Is = "AppID"
                        child.Text = AppID.Text
                    Case Is = "ConnectionTicket"
                        vchild.Text = ConnTktText.Text
                End Select
            Loop
        Loop
    Else
        Rem No SignonDesktopRq, build one and insert it.
        Set child = signonMsgsRq.firstChild
        Set child = signonMsgsRq.insertBefore
(buildSignonDesktopRq(doc), child)
    End If
Else
    Rem No SignonMsgsRq, build one, put on top of <QBXML>
    Set signonMsgsRq = doc.createElement("SignonMsgsRq")

```

```

        Dim newElement As IXMLDOMNode
        Set newElement = signonMsgsRq.insertBefore
        (buildSignonDesktopRq(doc), Null)
        Set child = top.insertBefore(signonMsgsRq, top.firstChild)
    End If
    Rem Return the text of the DOM we just built/edited
    updateSignonDesktopBlock = top.xml
Else
    Rem something went wrong, show the error.
    With doc.parseError
        MsgBox ("Error parsing request " & .errorCode & ": " &
        .filepos & .Line & .linepos & .reason & .srcText & .url)
    End With
    updateSignonDesktopBlock = ""
End If
End Function

```

Invoked by updateSignonDesktopBlock

\*\*\*\*\*

```

Private Function buildSignonDesktopRq(doc As DOMDocument40) As
    IXMLDOMElement
    Dim signonDTRq As IXMLDOMElement
    Set signonDTRq = doc.createElement("SignonDesktopRq")
    Dim newElement As IXMLDOMNode
    Dim curElem As IXMLDOMNode

    Set newElement = makeDOMELEMENT(doc, "ClientDateTime",
    getClientDate)
    Set curElem = signonDTRq.insertBefore(newElement, Null)
    Set newElement = makeDOMELEMENT(doc, "ApplicationLogin",
    AppLoginText.Text)
    Set curElem = signonDTRq.insertBefore(newElement, Null)
    Set newElement = makeDOMELEMENT(doc, "ConnectionTicket",
    ConnTktText.Text)
    Set curElem = signonDTRq.insertBefore(newElement, Null)
    Set newElement = makeDOMELEMENT(doc, "Language", "English")
    Set curElem = signonDTRq.insertBefore(newElement, Null)
    Set newElement = makeDOMELEMENT(doc, "AppID", AppID.Text)
    Set curElem = signonDTRq.insertBefore(newElement, Null)
    Set newElement = makeDOMELEMENT(doc, "AppVer", "1.0")
    Set curElem = signonDTRq.insertBefore(newElement, Null)
    Set buildSignonDesktopRq = signonDTRq
End Function

```

In the first sample function, a request string is supplied to the sample `updateSignonDesktopBlock` function, which loads the string into a DOM document and checks it for a properly constructed `SignonMsgsRq` element and `SignonDesktopRq` element. If these are not found, they are constructed.

The reason this check is performed is that in this particular sample the request string happens to be from a user supplied qbXML file where we have no knowledge of how it was constructed. Thus, if you were to supply a pure qbXML document containing only the QBXML element, the QBXMLMsgsRq element and nested qbXML requests, this function would still work. This could be useful if you are porting an application designed for use with other QuickBooks products.

If the `SignonMsgsRq` and `SignonDesktopRq` elements are not present in the supplied string, they are then constructed in the DOM document, via the second sample function listed above, `buildSignonDesktopRq`.

Notice that *any other signon request elements* in the supplied string are deleted. The reason is that this sample function is invoked simply to build a `SignonDesktopRq` to get a session ticket, so the other elements and/or attributes are undesirable.

The function returns a string containing the fully constructed signon element in XML format that can then be posted to the QuickBooks data exchange URL to get a session ticket.

## Parsing the SignonDesktopRs Response for the Session Ticket

After you post an XML document containing the `SignonDesktopRq` element to the QuickBooks data exchange URL, you receive a response XML document containing the element `SignonMsgsRs` and the nested element `SignonDesktopRs`. You need to parse these to obtain the session ticket or determine what went wrong if no such ticket is present.

### **Sample XML for the SignonDesktopRs Element**

The following XML sample shows how the returned `SignonDesktopRs` element is constructed inside the returned parent `SignonMsgsRs` element. You need to know this in order to parse the response.

```
<SignonMsgsRs>
  <SignonDesktopRs requestID = "UIDTYPE" statusCode = "INTTYPE"
    statusSeverity = "STRTYPE" statusMessage = "STRTYPE">
    <ServerDateTime>DATETIME</ServerDateTime>
    <SessionTicket>STRTYPE</SessionTicket> <!-- opt -->
  </SignonDesktopRs>
</SignonMsgsRs>
```

The elements, attributes, and their values within the SignonDesktopRs element are described in Table 1-2.

Table 1-2 SignonDesktopRs Values

Element Name	Description
ServerDateTime	The current system time at the QuickBooks Online Edition Site server taken at the time the request was processed.
StatusCode	The value indicating success or failure: if failure, the code indicates the nature of the failure, with various values possibly returned.  See Appendix A for a list of possible values.
SessionTicket	If the initial post connection ticket represents authorization without session authentication, the session ticket is returned in the response.  If the connection ticket was created with session authentication, however, no session ticket is returned and the error "Session Authentication Required" is returned.
StatusSeverity	Indicates level of failure.
StatusMessage	Provides a user-readable indication of the failure.

### VB Sample: Parsing and Handling the SignonDesktopRs Response

A good example of parsing this particular response is provided in the sample function getSessionTicket, which is listed under "VB Sample: Getting a Session Ticket" (page 71). The main item to observe in this listing is the check for the error code 2020, "Session Authentication Required" which will be encountered any time session authentication is used. If that error is detected, you must send the user to the session authentication site

## Constructing the SignonTicketRq Element for Data Exchange

---

Once a session ticket has been obtained either from the initial post (no session authentication) or from a user copy/paste (session authentication), your application can perform the main part of the qbXML work, namely, data exchange.

You need to construct the desired individual qbXML request messages within the main QBXMLMsgsRq element as described in the *QuickBooks SDK Concepts Manual*.

Then, in the same QBXML document, in front of the opening QBXMLMsgsRq tag, and just under the document's QBXML tag, you must insert the SignonMsgsRq element that contains a SignonTicketRq element.

## Sample XML for the SignonTicketRq Element

---

The following sample shows how the SignonTicketRq element is constructed inside the parent SignonMsgsRq element.

```
<SignonMsgsRq>

  <SignonTicketRq requestID = "UUIDTYPE">

    <ClientDateTime>DATETIME</ClientDateTime>

    <SessionTicket>STRTYPE</SessionTicket>

    <AuthID>IDTYPE</AuthID> <!-- opt -->

    <InstallationID>IDTYPE</InstallationID> <!-- opt -->

    <Language>STRTYPE</Language>

    <AppID>STRTYPE</AppID>

    <AppVer>STRTYPE</AppVer>

  </SignonTicketRq>

</SignonMsgsRq>
```

The elements, attributes, and their values within the SignonTicketRq element are described in Table 1-3.

Table 1-3 SignonTicketRq Values

Element Name	Description
ClientDateTime	The ClientDateTime is the current system time, which provides a timestamp for the signon request.
ApplicationLogin	This is the name of the application that you supplied when you registered to obtain an application ID.
SessionTicket	The session ticket, either returned automatically from the initial post or copied by the user during session authentication.

Table 1-3 SignonTicketRq Values

Element Name	Description
InstallationID	You need to create a unique installation ID for each instance of your desktop application. QuickBooks Online Edition uses this for error recovery. That value must be retrieved and inserted in this element.
AuthID	NA
Language	Currently, supply the value "English".
AppID	The application ID assigned when you registered your application with Intuit Developer Network.
AppVer	Application version string

## VB Sample: Constructing the SignonTicketRq Element

The following two sample functions show how to construct the SignonTicketRq element inside the parent SignonMsgsRq element. The following two sample functions show how to build a signon request to obtain a session ticket.

The first function (updateSignonTicketBlock) does various types of checking and operations based on the supplied request string, then if necessary invokes the second function (buildSignonTicketRq) to actually build the element:

```
Private Function updateSignonTicketBlock(request As String) As String
    Dim doc As DOMDocument40
    Set doc = New DOMDocument40
    doc.async = False
    doc.validateOnParse = False
    Rem Parse the input request...
    If (doc.loadXML(request)) Then
        Rem successful parse, figure out what we have
        Dim top As IXMLDOMNode
        Set top = doc.selectSingleNode("QBXML")
        If (top Is Nothing) Then
            Rem we expect at least the minimal qbXML: <QBXML/>
            MsgBox ("Missing top-level <QBXML> in request")
            updateSignonTicketBlock = Null
            Exit Function
        End If
        Rem Now look for the SignonMsgsRq aggregate...
        Dim signonMsgsRq As IXMLDOMNode
        Set signonMsgsRq = top.selectSingleNode("SignonMsgsRq")
        If Not signonMsgsRq Is Nothing Then
            Rem <SignonMsgsRq> present, should have SignonTicketRq
            Dim signonDTRqs As IXMLDOMNodeList
            Dim signonAppRqs As IXMLDOMNodeList
            Dim signonTktRqs As IXMLDOMNodeList
            Set signonDTRqs = signonMsgsRq.selectNodes
                ("SignonDesktopRq")
            Set signonAppRqs = signonMsgsRq.selectNodes
```

```

        ("SignonAppCertRq")
Set signonTktRqs = signonMsgsRq.selectNodes
        ("SignonTicketRq")
Dim curNode As IXMLDOMNode
Rem No SignonAppCertRq aggregates, they are only for
Rem hosted apps
Set curNode = signonAppRqs.nextNode
Do While Not curNode Is Nothing
    Dim removed As IXMLDOMNode
    Set removed = signonMsgsRq.removeChild(curNode)
    Set curNode = signonAppRqs.nextNode()
Loop
Rem No SignonDesktopRq, we want a SignonTicketRq
Set curNode = signonDTRqs.nextNode
Do While Not curNode Is Nothing
    Dim foo As IXMLDOMNode
    Set foo = signonMsgsRq.removeChild(curNode)
    Set curNode = signonDTRqs.nextNode()
Loop
Rem Now make sure we have a SignonTicketRq aggregate...
Set curNode = signonTktRqs.nextNode
If Not curNode Is Nothing Then
    Rem Have a SignonTicketRq, update data from UI
    Do While Not curNode Is Nothing
        Dim child As IXMLDOMNode
        Set child = curNode.firstChild
        Do While Not child Is Nothing
            Dim name As String
            Select Case child.nodeName
                Case Is = "ClientDateTime"
                    child.Text = getClientDate
                Case Is = "AppID"
                    child.Text = AppID.Text
                Case Is = "SessionTicket"
                    child.Text = SessTktText.Text
            End Select
        Loop
    Loop
Else
    Rem No SignonTicketRq, build and put in SignonMsgsRq
    Set child = signonMsgsRq.firstChild
    Set child = signonMsgsRq.insertBefore
        buildSignonTicketRq(doc), child)
End If
Else
    Rem No SignonMsgsRq, build it and SignonTicketRq
    Set signonMsgsRq = doc.createElement("SignonMsgsRq")
    Dim newNode As IXMLDOMELEMENT
    Set newNode = signonMsgsRq.insertBefore
        (buildSignonTicketRq(doc), Null)
    Set child = top.insertBefore(signonMsgsRq, top.firstChild)
End If
Rem we now have an updated request with SignonMsgsRq aggregate
Rem in "top" element, return the text of the updated request
updateSignonTicketBlock = top.xml
Else
    Rem Unable to parse the request XML; show the user the problem

```

```

        With doc.parseError
            MsgBox ("Error parsing request " & .errorCode & ": " &
                .filepos & .Line & .linepos & .reason & .srcText & .url)
        End With
        updateSignonTicketBlock = Null
    End If
End Function

*****
Rem Invoked from updateSignonTicketBlock
*****

Private Function buildSignonTicketRq(doc As DOMDocument40) As
IXMLDOMElement
Rem build (using DOM) the canonical <SignonTicketRq> aggregate
Rem using the information from the UI form.
    Dim signonTktRq As IXMLDOMElement
    Set signonTktRq = doc.createElement("SignonTicketRq")
    Dim newNode As IXMLDOMElement
    Set newNode = signonTktRq.insertBefore(makeDOMElement(doc,
        ClientDateTime", getClientDate), Null)
    Set newNode = signonTktRq.insertBefore(makeDOMElement(doc,
        "SessionTicket", SessTktText.Text), Null)
    Set newNode = signonTktRq.insertBefore(makeDOMElement(doc,
        "Language", "English"), Null)
    Set newNode = signonTktRq.insertBefore(makeDOMElement(doc,
        "AppID", AppID.Text), Null)
    Set newNode = signonTktRq.insertBefore(makeDOMElement(doc,
        "AppVer", "1.0"), Null)
    Set buildSignonTicketRq = signonTktRq
End Function

```

In the first sample function, a request string is supplied to the sample `updateSignonTicketBlock` function, which loads the string into a DOM document and checks it for a properly constructed `SignonMsgsRq` element and `SignonTicketRq` element. If these are not found, they are constructed.

The reason this check is performed is that in this particular sample the request string happens to be from a user supplied qbXML file where we have no knowledge of how it was constructed. Thus, if you were to supply a pure qbXML document containing only the QBXML element, the QBXMLMsgsRq element and nested qbXML requests, this function would still work. This could be useful if you are porting an application designed for use with other QuickBooks products.

If the `SignonMsgsRq` and `SignonTicketRq` elements are not present in the supplied string, they are then constructed in the DOM document, via the second sample function listed above, `buildSignonTicketRq`.

Notice that *any other signon request elements* in the supplied string are deleted. The reason is that this sample function is invoked simply to build a `SignonTicketRq`, so the other elements are undesirable.

The function returns a string containing the fully constructed signon elements in XML format that are then used in the qbXML posts to the QuickBooks data exchange URL to carry out the desired manipulations of the QuickBooks company data.

## Supporting User Cancellation of Authorization

---

On your main form or in a menu pull-down in your main application form, you need to provide a way to let the user cancel the authorization (unsubscribe) for your application. For example, in response to a command button click, you could launch a web browser and send the user to the authorization URL.

### VB Sample: Unsubscribe Button-Click Handler

---

The following code responds to the user's push of the cancellation button.

```
Private Sub UnsubscribeBtn_Click()  
    Dim loginSvr As String  
    Dim myAppID As String  
    Dim myConnTkt As String  
    Dim mySessTkt As String  
    loginSvr = SubServerText.Text  
    myAppID = AppID.Text  
    myConnTkt = ConnTktText.Text  
    mySessTkt = SessTktText.Text  
    Dim cancelURL As String  
    cancelURL = loginSvr & "/j/qbn/sdkapp/cancel?serviceid=2004&appid=  
        " & myAppID" & "&conntkt=" & myConnTkt"  
    If Len(mySessTkt) > 0 Then  
        cancelURL = cancelURL & "&tkt=" & mySessTkt  
    End If  
    SubscriptionURLText.Caption = cancelURL  
    showURL (cancelURL)  
End Sub
```

In the sample button click handler, the URL is assembled from the official QuickBooks login URL that already exists in the sample application as a text box default (SubServerText), the value of which is:  
<https://login.quickbooks.com>.

To this value is appended the relative path at the cancellation server that takes you to the actual cancel page and provides certain REQUIRED parameters: /j/qbn/sdkapp/cancel?serviceid= 2004&appid=" & myAppID" & "&conntkt=" & myConnTkt".

In this appended text string, the value /j/qbn/sdkapp/cancel? will be part of every cancel URL because it is the relative path to the cancellation page at the QuickBooks Online Edition login server. The other parameters are required, but will vary for every application.

*serviceid*

Always use the value 2004.

*appid*

The application ID assigned to your application when you registered it with IDN.

*conntkt*

The connection ticket being cancelled.

Once the cancellation URL is fully constructed, `showURL` is invoked to launch the web browser and send the user to the QuickBooks cancel page. See “Sending Users to QuickBooks Online Edition Sites” (page 86) for information on `showURL`.

## Supplying Client Date/Time in the Required Format

---

When you construct the various signon message elements required during the attempt to obtain a session ticket and during normal data exchange posts, you must supply a client time value in a specific time format.

The following VB function shows how to get the current system time and transforms it into the required format:

```
Private Function getClientDate() As String
Rem utility function to get date in "standard" qbXML format
Dim today
    today = Now
    getClientDate = Year(today) & leadingZero(Month(today)) &
        leadingZero(Day(today)) & "T"
    getClientDate = getClientDate & leadingZero(Hour(today)) &
        leadingZero(Minute(today)) & leadingZero(Second(today))
End Function
```

## Sending Users to QuickBooks Online Edition Sites

---

In your application, you will need to launch a web browser and send users to various QuickBooks URLs to have them authorize your application, cancel that authorization, and so on. Notice that this is not the posting of normal data exchange data to a QuickBooks company, but a simple redirect to another site. (See the following section for information on posting data to a QuickBooks company.)

A convenient way to send the user to the QuickBooks Online Edition authorization site, or the cancellation site, etc., is to assemble the URL and then supply it to the InternetExplorer object available in Visual Basic, as shown in the following sample `showURL` function:

```
Public Sub showURL(iFname As String)

Rem Just open an IE browser to the desired URL

    Dim IE1 As New InternetExplorer

    IE1.Visible = True
```

```
IE1.Navigate (iFname)
```

```
End Sub
```

## Posting Data to a QuickBooks Online Edition Company

---

In order to perform data exchange with a QuickBooks company, you need to post a validly formed qbXML document to QuickBooks Online Edition. The following Visual Basic sample function shows a convenient way to do this, with the validly formed XML document (containing the session ticket and qbXML requests) supplied as an input to the function.

```
Private Function postReq(url As String, xml As String) As String
Rem POST the given XML to the given URL
Dim objXMLHttp As XMLHTTP40
Set objXMLHttp = New XMLHTTP40
objXMLHttp.open "POST", url, False
objXMLHttp.setRequestHeader "content-type", "application/x-qbxml"
objXMLHttp.send xml

Rem we've posted, so now lets check the status (200 is HTTP_OK)
If ("200" <> objXMLHttp.Status) Then
Rem Something went wrong, show the error and send it to caller
postReq = objXMLHttp.Status & ": " & objXMLHttp.StatusText &
vbCrLf & objXMLHttp.responseBody
MsgBox (postReq)
Else
Rem all's well, send response text QuickBooks Online Edition)
postReq = objXMLHttp.responseText
End If
End Function
```

This function uses the XMLHTTP object available from Microsoft's XML Core Services 4.0 SDK. It posts the supplied string, which in your application will always be a well formed qbXML document with either qbXML requests or a SignonMsgsRq/SignonTicketRq used to return a session ticket.

The sample does some rudimentary error checking, which you probably would want to augment. Notice that the function returns the response messages from QuickBooks.

## Implementing "Logout" Functionality

---

In certain types of applications where multiple users have access to the same computer, it may be desirable to implement logout functionality where each user can logout of the QuickBooks company without shutting down the application.

Subsequent users would then be required to login to the QuickBooks company, assuming the company administrative user has selected Session Authentication to support this.

To implement this in your code, you could simply have a logout button that destroys the session ticket that is stored in memory. (Session tickets should only be stored in memory.) If there is no session ticket at the client, any subsequent attempt to access the QuickBooks company will result in the display of the QuickBooks login page, requiring the user to log on to the QuickBooks company.

## Handling Errors

---

Your application must handle three types of errors:

- Standard HTTPS errors resulting from your application's attempt to use network resources.
- QuickBooks Online Edition errors resulting from your application's attempt to connect to it
- QuickBooks Online Edition errors resulting from your application's attempt to manipulate data in the user's company.

For lists and descriptions of the standard HTTP/S errors, please consult any of the many references on the subject. If you are new to this area, you may want to visit the web site [www.w3.org](http://www.w3.org), which is the web site for the World Wide Web consortium.

### **NOTE**

Certain standard errors will be returned by QuickBooks Online Edition under some circumstances. For example, malformed XML data will be rejected with a standard error 400, Bad Request.

For a list of the errors that can be returned during runtime access or manipulation of QuickBooks company data, please see Appendix A of this guide. Please see the *QuickBooks SDK: Concepts Manual* for details on how to handle these types of errors.

For a list of QuickBooks Online Edition connection-related errors, please see the following list:

2000  
Authentication failed -- Invalid login name or password / certificate / ticket

2010  
Unauthorized

2020  
Session Authentication required

2030  
Unsupported signon version

2040  
Internal error

The sample code provided with the QuickBooks SDK shows how to handle many of these connection-related errors.

## Using wincrypt to Store Connection Tickets

---

The following C++ example shows how to encrypt and decrypt the connection ticket using the functionality provided in the Windows encryption library `wincrypt.h`.

```
#include <stdio.h>
#include <windows.h>
#include <wincrypt.h>
#define ENCODING_TYPE (PKCS_7_ASN_ENCODING | X509_ASN_ENCODING)
void HandleError(char *s);

unsigned char *GetRandomBytes( unsigned long *length )
{
    HCRYPTPROV    hCryptProv;
    BYTE        pbData[1024]; // Size chosen arbitrarily.
    BYTE        *output;

    //-----
    // Acquire a CSP context.

    if(CryptAcquireContext(
```

```

    &hCryptProv,
    NULL,
    NULL,
    PROV_RSA_FULL,
    0))
{
    printf("CryptAcquireContext succeeded. \n");
}
else
{
    HandleError("Error during CryptAcquireContext!\n");
}

if(CryptGenRandom(
    hCryptProv,
    *length,
    pbData))
{
    printf("Random sequence generated. \n");
    output = new unsigned char [*length];
    memcpy( output, pbData, *length );
}
else
{
    HandleError("Error during CryptGenRandom.");
    output = NULL;
}

return output;
}

void main()
{

    // Encrypt data from DATA_BLOB DataIn to DATA_BLOB DataOut.
    // Then decrypt to DATA_BLOB DataVerify.

    //-----
    // Declare and initialize variables.

    DATA_BLOB DataIn;
    DATA_BLOB DataOut;
    DATA_BLOB DataVerify;
    BYTE *pbDataInput =(BYTE *)"Hello world of data protection.";
    DWORD cbDataInput = strlen((char *)pbDataInput)+1;
    DataIn.pbData = pbDataInput;
    DataIn.cbData = cbDataInput;
    CRYPTPROTECT_PROMPTSTRUCT PromptStruct;

```

```

LPWSTR pDescrOut = (LPWSTR)0xbaadf00d ; // NULL;

//-----
// Begin processing.

printf("The data to be encrypted is: %s\n",pbDataInput);

//-----
// Initialize PromptStruct.

ZeroMemory(&PromptStruct, sizeof(PromptStruct));
PromptStruct.cbSize = sizeof(PromptStruct);
PromptStruct.dwPromptFlags = (CRYPTPROTECT_PROMPT_ON_PROTECT |
CRYPTPROTECT_PROMPT_ON_UNPROTECT );
PromptStruct.szPrompt = L"Hey- Look Here: This is a user prompt.";

//-----
// Generate some random bytes to ensure better security.

unsigned char *randomBytes = NULL;
unsigned long randomLength = 16; // use 16 bytes of randomness.

randomBytes = GetRandomBytes( &randomLength );
DATA_BLOB random;
random.cbData = randomLength;
random.pbData = randomBytes;

if ( randomBytes == NULL )
    HandleError("Error generating random bytes.");

//-----
// Begin protect phase.

if(CryptProtectData(
    &DataIn,
    L"We're protecting test data. ", // A description sting.
    &random, // Optional entropy.
    NULL, // Reserved.
    &PromptStruct, // Pass a PromptStruct.
    0,
    &DataOut))
{
    printf("The encryption phase worked. \n");
}
else
{

```

```

    HandleError("Encryption error!");
}
//-----
//   Begin unprotect phase.

if (CryptUnprotectData(
    &DataOut,
    &pDescrOut,
    &random,          // Optional entropy
    NULL,            // Reserved
    &PromptStruct,   // Optional PromptStruct
    0,
    &DataVerify))
{
    printf("The decrypted data is: %s\n", DataVerify.pbData);
    printf("The description of the data was: %S\n",pDescrOut);
}
else
{
    HandleError("Decryption error!");
}
//-----
// At this point, memcmp could be used to compare DataIn.pbData and
// DataVerify.pbData for equality. If the two functions worked
// correctly, the two byte strings are identical.

//-----
//   Clean up.

delete randomBytes;
LocalFree(pDescrOut);
LocalFree(DataOut.pbData);
LocalFree(DataVerify.pbData);
} // End of main

//-----
// This example uses the function HandleError, a simple error
// handling function, to print an error message to the standard error
// (stderr) file and exit the program.
// For most applications, replace this function with one
// that does more extensive error reporting.

void HandleError(char *s)
{
    fprintf(stderr, "An error occurred in running the program. \n");
    fprintf(stderr, "%s\n", s);
    fprintf(stderr, "Error number %x.\n", GetLastError());
}

```

```
    fprintf(stderr, "Program terminating. \n");
    exit(1);
} // End of HandleError
```



# INDEX

## A

- access rights, and connection tickets 7
- activity logs 8
- application registration 9
- architectures 3
- authorization, multiples 8

## C

- callback URLs 8
- connection key 1
- connection tickets, and access rights 7
- connection tickets, and session authentication 7

## D

- Desktop applications, architecture 3

## I

- IDN, registering with 9
- Intuit Developer Network, see IDN 9

## J

- Java, Keystore 13
- Java, keytool, using 13

## L

- login key 1
- login security 1

## M

- multiple authorization 8

## R

- registration, application 9
- registration, server vs desktop 9

## S

- server applications, getting certificates 13
- session authentication, defined 7
- session authentication, runtime checks for 8
- session tickets 7
- supported architectures 3

## U

- URLs 8

