Intuit QuickBooks® SDK

QuickBooks Web Connector Programmer's Guide

Version 2.0

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/.

Chapter 1: Introduction to QBWC Programming

Chapter 2: The QBWC Communication Model

Chapter 3: Implementing a Web Service for QBWC

Chapter 4: Building The QWC File for Your Users

Chapter 5: Exchanging Data with QuickBooks and QBPOS

Chapter 6: Interacting Directly with the Web Connector

Chapter 7: Understanding the End-User Experience and Setup

Chapter 8: Handling Errors

Chapter 9: How Do I TroubleShoot Problems?

Chapter 10: QBWC Callback Web Method Reference

Appendix A: Understanding and Responding to QBWC Error Codes

ABOUT THIS GUIDE

This *Programmer's Guide* describes the integration of QuickBooks and QuickBooks POS with web services via the QuickBooks Web Connector (QBWC) application. The purpose of this guide is to provide the details you need to know in order to successfully create a web service that talks to QuickBooks or QuickBooks POS.

In this guide, the examples are in C-sharp.

Who Should Read This Guide

This guide is for developers who are creating web service applications that integrate with QBWC.

In order to create the web service, we assume you are familiar with the platform you are developing for and the language you are using to program in. You should also know a little about SOAP, about XML and how to build an XML document.

Before You Begin

Be sure to familiarize yourself with the material contained in the *Onscreen Reference* for QuickBooks and for QuickBooks POS, which contains the qbXML and qbposXML syntax for each request and response message type.

What's New in This Release

The following improvements have been made to QBWC 2.0:

- Support for interactive mode:
  - > Protocol Handler for Internet Explorer that allows web-based applications to interact directly with the web connector.
  - > Three new optional web methods to facilitate Interactive mode:
    -InteractiveURL
    -InteractiveRejected
    -isInteractiveDone
- Masterkeys are now handled and maintained automatically by .NET managed password storage mechanism.
- Update locking mechanism to help manage company file during simultaneous updates from multiple web connector clients.
- Included VERBOSE mode for logging level. With this, we now have three log levels: NONE = No logging, DEBUG (default setting) = Logging + first 50 characters of request/response xml, VERBOSE = Logging + complete request/response xml

- Added a response value of an O: (stands for Okay) O:<QBWC_Version_Supported_By_Server> for clientVersion(). It provides an update path for user if server's QBWCVersion is greater than user's QBWCVersion
- NoOp for sendRequestXML(). When sendRequestXML() call receives an empty string, QBWC calls getLastError(). If a NOOP is sent back from web-service for the getLastError(), QBWC will pause update for 5 seconds. This allows a web-service to tell QBWC to wait five seconds before calling sendRequestXML() again.
- New webmethod getServerVersion() provides a way for web-service to notify QBWC of itís version. This version string shows up in the More Information pop-up dialog in QBWC.
- Notification (system tray pop up) is now turned off by default.
- A new optional QWC parameter <CertURL> to provide means to provide certificate server for ssl certificates other than web server.
- A new optional QWC parameter <Notify> introduced. Value of true will enable notification (pop up at systray) at app level. Anything else will disable notification.
- - A new optional QWC parameter <AppDisplayName> is introduced. If available, QBWC will use this to display name in the QBWC UI. Otherwise, use <AppName> as usual. This is just for UI purpose. Update process still uses the <AppName> (or, AppUniqueName if provided)
- A new optional parameter <AppUniqueName> is introduced. If this element is available in QWC file, QBWC will not go into itís typical clone/replace mode for AppName and directly use the replace routine.
- There are new optional QWC file parameters for three rp.AuthPreferences parameters IsReadOnly (true/false), UnattendedModePref (umpRequired/umpOptional), and PersonalDataPref(pdpNotNeeded/pdpOptional/pdpRequired).
- Improved performance due to code refactoring.
- Improved error messages with suggestions on actions where applicable.

CHAPTER 1
INTRODUCTION TO QBWC PROGRAMMING

If you are developing a web-based application that works with QuickBooks or QB POS, you'll want to consider implementing a solution designed to work with the QuickBooks Web Connector (QBWC). QBWC enables web-based applications to access Quickbooks and QuickBooks Point of Sale (QBPOS) over the internet.

QuickBooks Supported by QBWC

The following QuickBooks versions/editions are supported/not supported as indicated:

- Enterprise Edition: all editions
- Pro & Premier, QB 2002 and later
- Simple Start Edition: QuickBooks 2006 and later
- Online Edition: Not supported

Why Do I Need to Support QBWC in My Web Service?

There are a couple of reasons why you need to include QBWC support in your implementation of a web-based application that talks to QuickBooks or QBPOS. The first revolves around the basic COM issue, which applies more to QuickBooks integrations than to QBPOS. The second revolves around the firewall issue, which potentially applies to both.

What is the COM Issue and How Does QBWC Solve This?

In order for an application to access QuickBooks via the SDK, it must instantiate the QuickBooks SDK request processor via COM. COM requires the COM object server and its client (your application) to be resident on the same machine, or at least in the same LAN (if you use DCOM and configure things very carefully). Consequently, your web-based application, which is not in the same LAN or on the same machine, cannot access QuickBooks directly via the request processor.

To get around this limitation, in the past, developers have created a go-between application that resides on the same machine as QuickBooks and does the QuickBooks interaction, passing the results back to their web-based app. This approach does work, but adds a significant amount of learning and work to the implementation effort.

Which is why QBWC was developed. QBWC is a free and standard go-between application that can be used by any web-based application that needs to talk to QuickBooks or QBPOS. The core function of QBWC is to act as the conduit through which all qbXML/qbposXML requests and responses pass between web-based applications and QuickBooks or QBPOS.

What is the Firewall Issue and How Does QBWC Solve That?

For QBPOS, the request processor can be on a remote machine, such as the one hosting your web service. So a web service could conceivably talk to remote QBPOS installations. However, with this, there is a security issue as those remote QBPOS installations would have to open their firewall to each web service.

For QuickBooks, some developers have considered using the Remote Data Sharing (RDS) feature introduced with QB SDK 2.1 to allow their web-based application to talk to QuickBooks. This approach is not recommended for various security reasons, one of which being the requirement of opening a firewall port to the RDS server, which is not secure enough for this purpose when it comes to the internet, as RDS is a LAN solution, not an internet one.

QBWC eliminates the firewall issue by using an “upside-down” communication model where the QBWC initiates the session with the web service over HTTPS and asks the web service if it has work for QuickBooks or QBPOS (see Figure 1-1.) Consequently, there is no need to open any ports.

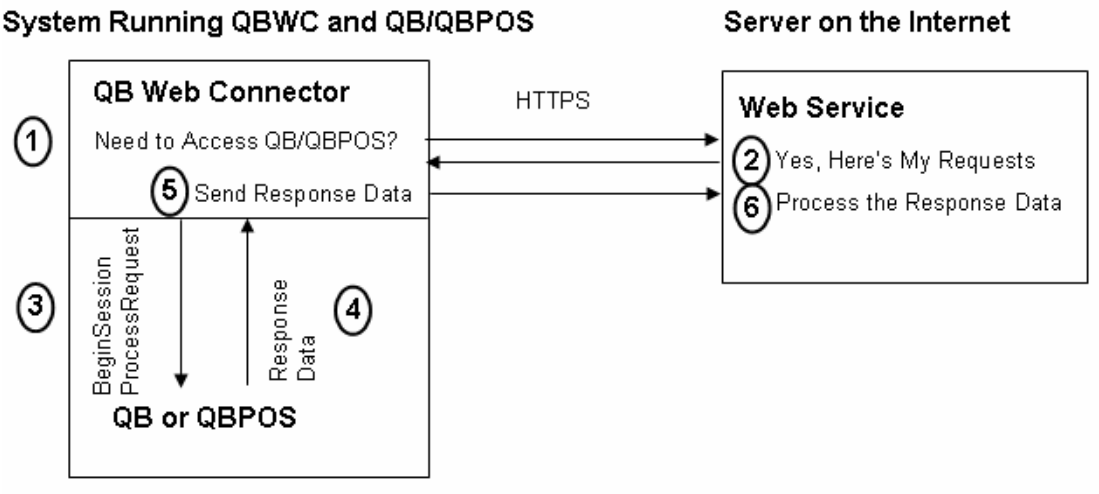


Figure 1-1 QBWC "upside-down" communication model

IMPORTANT

Some firewalls may initially interfere with QBWC in that they may prevent QBWC's initial outbound contacting of the web services. In these special cases, some additional tweaking of the firewall to allow QBWC to reach out to web services may be required.

Are There Any Alternatives to QBWC?

The only recommended alternative to QBWC to enable web-based application integration is for a developer to write their own go-between application, in effect, replacing QBWC with their own implementation.

There are other alternatives, but these are not recommended. Using RDS involves substantial security risk, as we've already mentioned. Using the unsupported Intuit Interchange Format (IIF), as some web developers have done, is not recommended because this bypasses the QuickBooks business logic, and so could result in data that is unsound from an accounting and a QuickBooks business logic perspective.

The QBWC-to-Web Service Communication Universe

There are two aspects of the overall QBWC-to-web service communication that you need to keep in mind:

- What does the initial customer interaction look like?
- What does ongoing data communication with the web service look like?

Initial Customer Interaction with Your Web Service

The customer's first "communication" with your web service is an out-of-band communication in which your customer does all the things that need to get done before any data communication can happen. A typical activity sequence for this initial stage is shown in Figure 1-2.

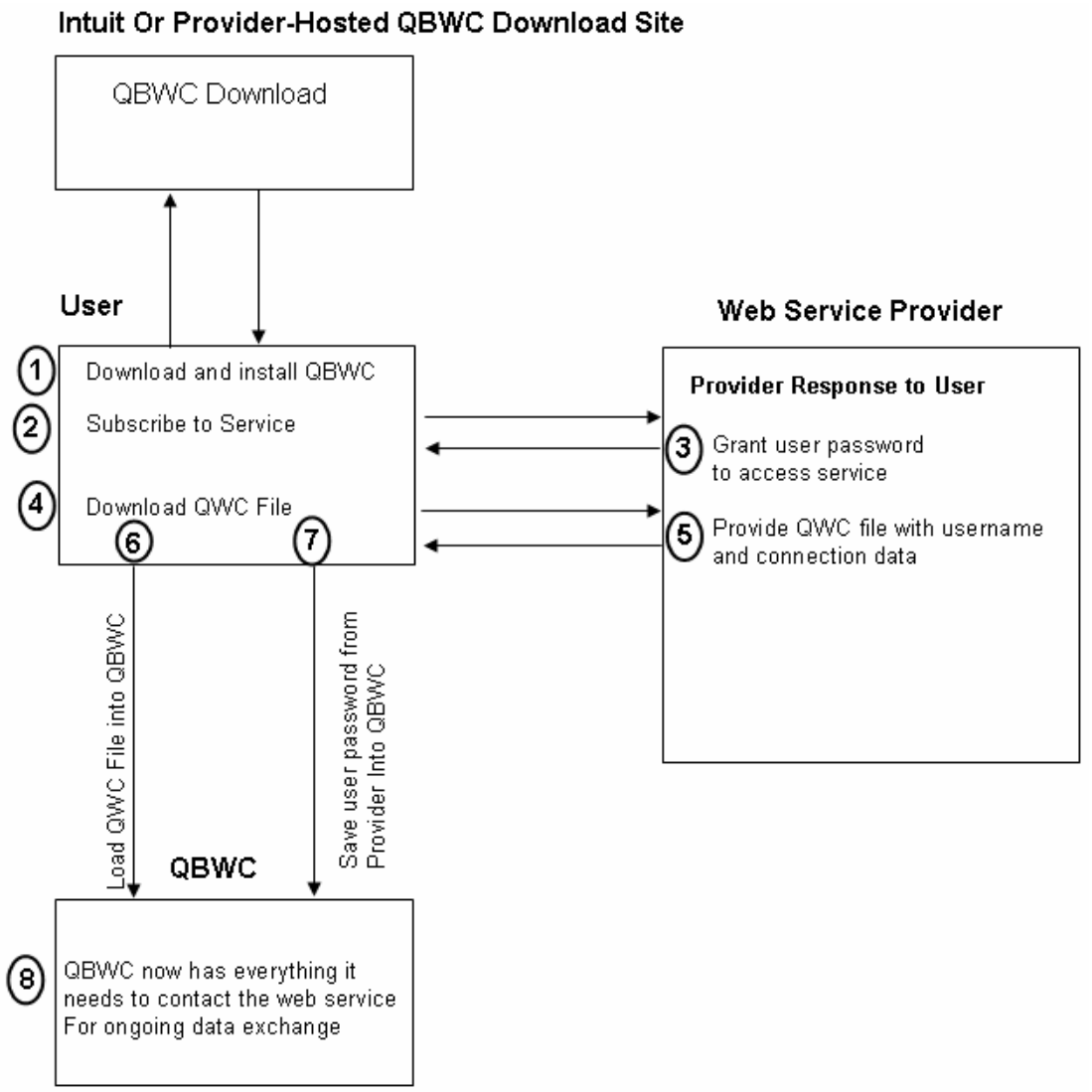


Figure 1-2 How a User Gets Ready to Access Your Web Service

As shown in the figure, the user must first learn about your service and what it requires, downloads and installs QBWC, subscribes to your service and obtains a password and a QWC configuration file. The QWC file, when loaded into QBWC, automatically transfers *almost* everything QBWC needs to contact your web service, such as user name, URLs, and so forth. (See Chapter 4, “Building The QWC File for Your Users,” for details on constructing the QWC file.)

The only thing NOT automatically loaded into QBWC is the user password from the web service provider. For security reasons, the user needs to save this manually into QBWC, where it is encrypted and stored.

Ongoing Communication Between QBWC and a Web Service

Figure 1-3 shows a high level view of the communication between a user's local system running QuickBooks/QuickBooks POS with QBWC talking to a web service over the internet.

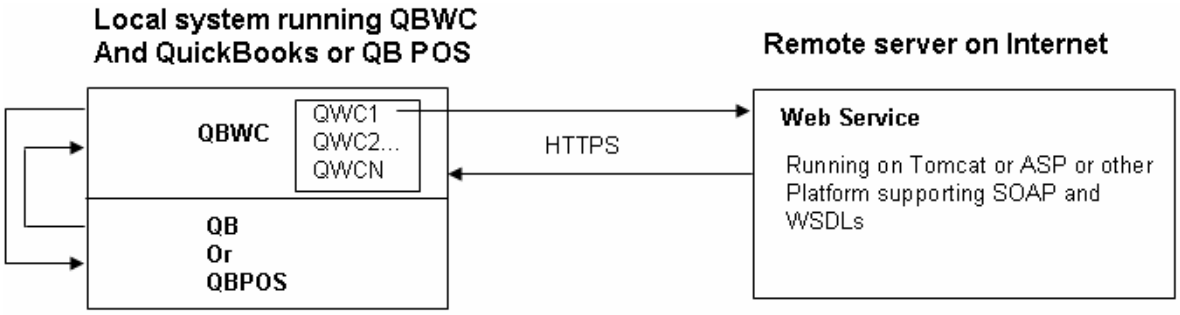


Figure 1-3 High-level communication diagram

QBWC uses the QWC file from each web service provider to locate that providers web service and begin the communication sequence. A detailed view of this communication sequence can be found in Chapter 2, "The QBWC Communication Model." Your web service must implement the SOAP-based interfaces listed in Chapter 2 and described in detail in Chapter 10.

QBWC contacts the web service when your customer asks it to or at the regular intervals scheduled by your customer. If your web service needs to do some work for the customer, it responds with requests for QuickBooks or QB POS, which QBWC forwards to QuickBooks or QB POS, then returns the responses to your web service. If the web service has no work to be done at the time QBWC makes contact, then the communication simply stops.

What Will My Web Service Solution Look Like?

This document does not cover certain aspects of your total solution, such as getting subscription requests from your customer and providing them with passwords, the mechanism used to supply QWC files and so forth. It describes primarily those core pieces of functionality you must provide:

- The QWC file you provide to the customer that contains all the connection data
- The web service QBWC callbacks you must implement in your web service

How to Build a QWC File

Each customer will have to have a separate QWC file with their unique username. The user downloads this and opens it to automatically load all its data into QBWC. Instructions on constructing this QWC file are provided in Chapter 4, "Building The QWC File for Your Users."

How to Build QBWC Support into Your Web Service

To enable QBWC to work with your web service, you need only implement the following SOAP interfaces

- authenticate
- clientVersion
- closeConnection
- connectionError
- getLastError
- receiveResponseXML
- sendRequestXML

These callbacks are described in detail in Chapter 10, “QBWC Callback Web Method Reference.” A detailed description of how they are used is provided in Chapter 2, “The QBWC Communication Model.”

Are There Samples to Jumpstart My Work?

The QB SDK package provides two sample web services, one for QuickBooks, one for QuickBooks POS, each with their own QWC file. These are located in the QB SDK samples subdirectory \samples\qbdt\c-sharp\qbXML\WCWebService and in the QBPOS SDK samples subdirectory \Samples\qbpos\c-sharp\qbposxml\QWCPOSWebService.

Each sample web service can be run locally on your system along with QBWC, that is, with no certificates, to keep things simple. Directions on building and running the sample are provided in the readme.html page for the samples.

Frequently Asked Questions

The rest of this chapter provides answers to several frequently asked questions.

What Platforms and Languages can I use in my Implementation?

The web service should be able to run on any platform that supports standard SOAP for communication. Platforms that are known to work include Apache Tomcat (Axis)and ASP (.Net).

Why Do I Need SOAP?

There are several technologies designed to allow dissimilar applications to talk to each other. That is, you can write one side in C# on Windows XP and the other could be Cobol on an IBM Mainframe (which is an extreme example, perhaps). A few years ago a technology called Common Object Request Broker Architecture (CORBA) was popular,

but SOAP (Simple Object Access Protocol) has emerged to grab greater mindshare. Although if you like cheap jokes, you could argue that the advantages offered by each were largely a wash.

The main point is that SOAP provides a way for data to flow between two disparate systems. From the perspective of the remote system, it doesn't matter what language or technology you use to implement your web service, so long as this system is capable of interpreting the object/message being passed via SOAP.

Which QuickBooks/QB POS Versions Support QBWC?

QBWC works with any QuickBooks or QB POS product that supports the QB SDK and QBPOS SDK, respectively, except QuickBooks Online edition. However, older versions may not support some of the newer SDK requests that newer QuickBooks or QB POS versions support. QBWC does return QuickBooks version data to help you determine whether your web service will work with the customer's QuickBooks or not.

Can I Specify Which QuickBooks Editions Access My Service?

There is an <AuthFlags> parameter in the QWC file that allows you to specify which QuickBooks editions are supported by your web service. By default, all editions are supported, even Simple Start editions. This default is different from the default behavior of AuthFlags for QB SDK applications. For QB SDK applications the default AuthFlags support is not for all QB editions, but only Enterprise, Premier, and Pro, in order to avoid breaking existing applications that didn't know about Simple Start edition.

Since QBWC is a new product, it makes sense to require application developers to think about Simple Start edition from the beginning. Accordingly, when you test your application, and you use the default AuthFlags, you must test your application against Simple Start edition, to make sure Simple Start provides the functionality your application requires.

Here are the values you can supply for <AuthFlags>:

Table 1-1 Supporting QuickBooks Editions

QB Supported	AuthFlags Value
Support All (default)	0x0
SupportQBSimpleStart	0x1
SupportQBPro	0x2
SupportQBPremier	0x4
SupportQBEnterprise	0x8

If you want to support several editions but not all, you can AND the values for the editions you want to support.

Does My Web Service Need a Certificate to Access QBWC?

All production communication between web services and QBWC uses HTTPS. This means SSL is required, which means you'll need a X.509 certificate. This is a standard certificate that you can obtain from companies (such as Verisign) that provide them.

Developing a Web Service Without Certificates

However, for development purposes only, you can bypass the certificate issue by using http and specifying "localhost" in your QWC AppURL settings. This will allow you to run your web service on your system and let it talk to the QBWC installation on your system. You could also optionally use a domain name within your LAN to test against a server somewhere in your LAN. For more details, see Chapter 4, "Building The QWC File for Your Users."

Alternatively, if you want to use a self-signed certificate *for testing purposes only,* you can use the tool Microsoft provides for this on the *Microsoft web site* . Follow the instructions at the Microsoft web site. Or, you could also use openssl to test against if you happen to have cygwin installed. (Though the openssl instructions say it is for IIS 6.0, it works fine with IIS 5.1.)

Where is the QBWC WSDL?

The QBWC WSDL is located *at the IDN web site* .

Is There a Limit to the Number of Messages I Send to QBWC?

There is no limit on the number of messages your web service sends to QuickBooks Web Connector (QBWC). It depends on your application -- when in response to receiveResponseXML() you send a return value of 100 (which means 100% completed) then QBWC will stop calling sendRequestXML().

Why QBWC and Not a Simple Web Interface?

Some developers may wonder why we supply QBWC rather than providing a web interface. This is a philosophical question that we'll have to address in a more longwinded way.

Once upon a time, Quickbooks started out with the premise of making an accounting system for people who weren't accountants. The assumption is that the end users are experts in their business, not ours. This remains our assumption. We can't assume that our customers are going to figure out all the technical and security issues that surround the seemingly simple issue of providing an application that opens internet access to our customer's financial data. That access must be kept as secure as possible and it must keep the business owner in charge of access.

In other words, it can be difficult for developers to handle all the server components and security components necessary to be successful at a QBWC-centric web service, but we feel that the difficulty is better on the developer side than pushing this difficulty off to the end users. Anyone developing an application that opens financial data to direct internet access should carefully consider the security implications.

If you (as the developer) understand your user community well enough to know that they can manage these decisions, then this functionality would make a good value-add you can develop. If you are considering this business opportunity, please make sure you cover the following areas:

1. Your clients will have to allow external traffic through their firewall into their QuickBooks/QuickBooks POS system.
2. The ISP hosting your client will have to allow this traffic through, not all ISP's allow this. Make sure you charge enough for your product to recoup the investment you'll have to make at installation time working with ISP technical support. In the end, some of your clients may be forced to use a different ISP.
3. Your client will have to configure SSL servers, if they don't have an IT department, they'll likely look to you for help. If they misconfigure, or forget to enable security, their data will be transferred in the clear. If they don't understand this they will likely be upset with you for not making this clear.
4. In order to contact their system, they will have to have updated DNS records. If they don't have a permanent IP address, they'll have to have DDNS somewhere. They will likely look to you for help on this.
5. Because this is financial data, the bad guys have an incentive to try to get in. Your clients will need to keep up with security patches in all the middle wear. And they need to have very good policies about not allowing Trojan software or viruses onto their systems. And they need to understand Phishing and prepare their staff to recognize and react appropriately when they face it.

CHAPTER 2
THE QBWC COMMUNICATION MODEL

In QBWC-to-web service communication, the typical QB/QBPOS SDK application pattern is turned a bit upside-down. In a typical QB/QBPOS SDK application, the application contacts QuickBooks or QuickBooks POS when it needs access.

In QBWC-to-web service communication, it is QBWC that contacts the application (the web service) to ask whether that application wants to access QuickBooks/QuickBooks POS. The QuickBooks/QuickBooks POS user decides how often QBWC "polls" your web service by using the scheduler feature inside QBWC. Figure 2-1 illustrates this communication flow.

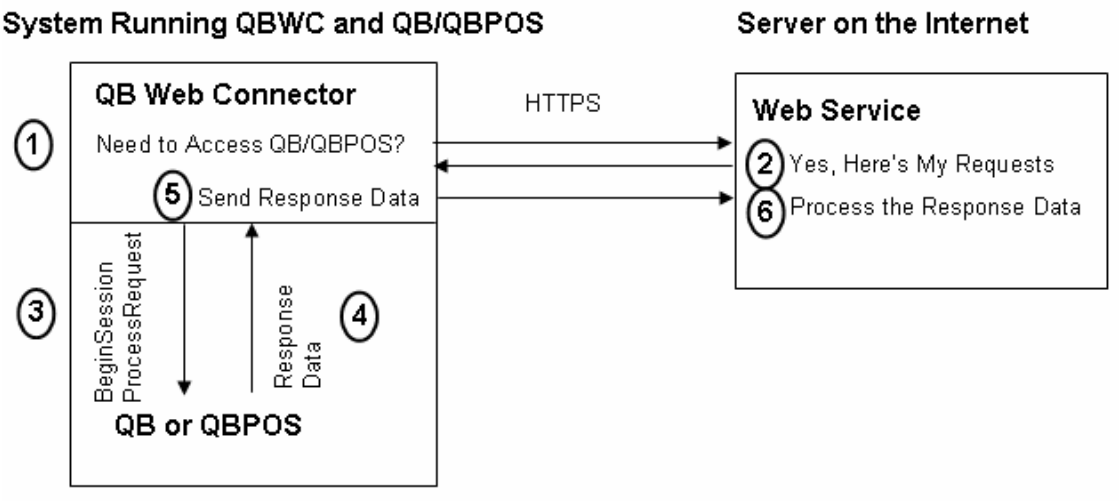


Figure 2-1 QBWC and Web Service Communication Flow

A Closer Look at the Communication Model

The communication flow overview shown in the following figures gives some pretty good hints as to how your web service needs to be structured to talk to the Web Connector. The Web Connector will be calling into your web service and that your web service will need to supply certain callback methods that will be expected by the Web Connector in order to support the communication flow. Your web service must implement all of the callbacks shown in the figures. (For detailed descriptions and sample code for each callback, see Chapter 10, "QBWC Callback Web Method Reference.")

To implement the callbacks in your web service, you'll need a more precise picture of what is transpiring within the QBWC-web service communication. So, take a look at the following illustrations as well as the running commentary provided with each drawing.

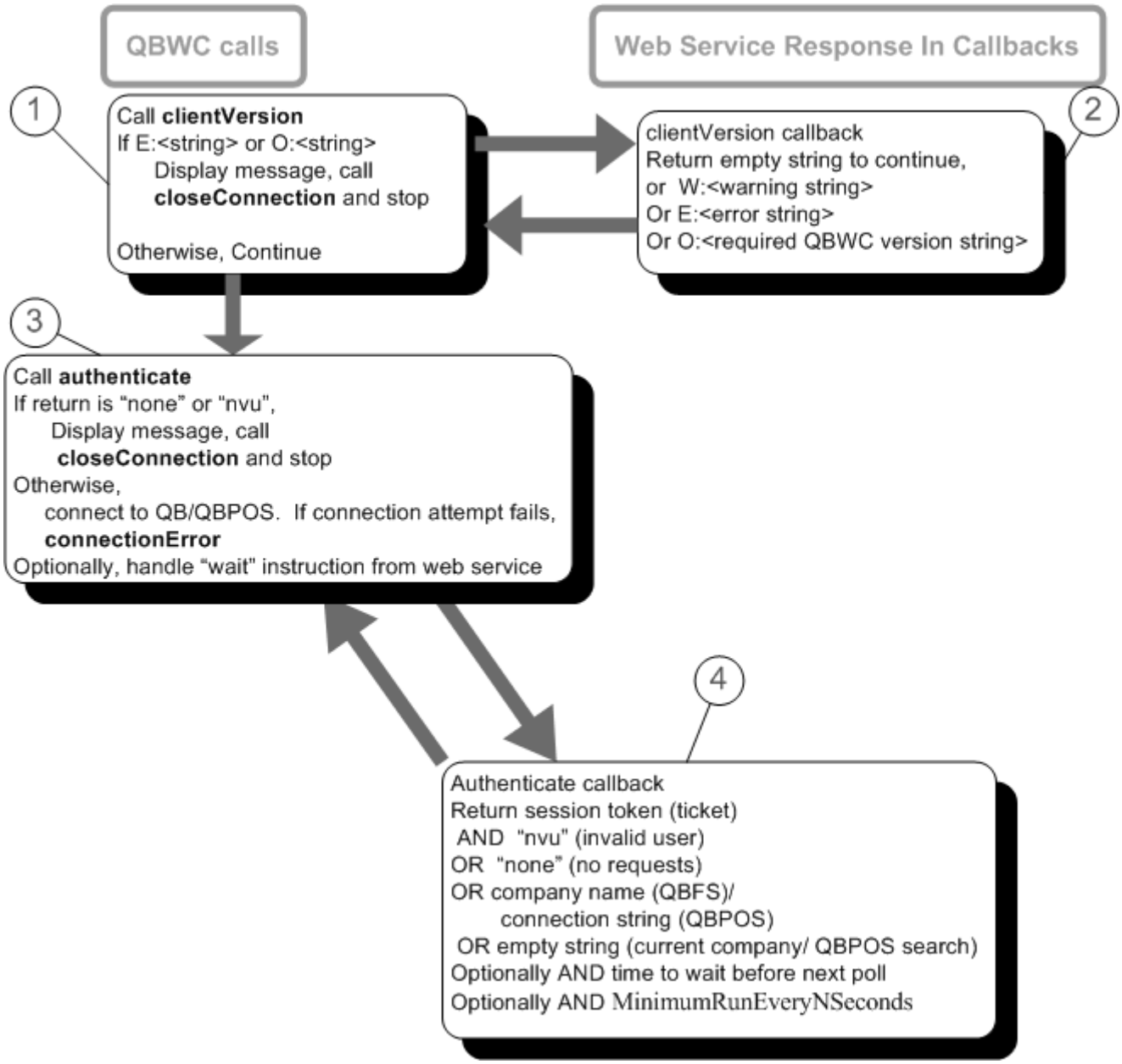


Figure 2-2 clientVersion and authenticate callbacks

What Figure 2-2 shows is what happens when a user at the QBWC clicks the Update button or when a scheduled Update for a web service occurs. Here is what happens:

1. First, QBWC calls your clientVersion callback to let your web service know which QBWC version is calling. The purpose of this is to let you warn your user or even stop the update if your web service doesn't support that QBWC version. Notice that if your web service doesn't have the optional clientVersion callback, QBWC proceeds to the authenticate call. In fact, QBWC always proceeds to call authenticate unless it receives a message string prefixed by the two characters "E:" or the two characters "O:" (O as in Oscar, not zero).

   You can append a string after the colon; for E: this would be an error string, for O: this would be a QBWC version string, which is used to tell the user at the client end what QBWC version your web service requires.

2. Next, QBWC calls your authenticate callback, supplying the username you provided your user via the QWC file, as described in Chapter 4, "Building The QWC File for Your Users." QBWC also sends the password that you supplied to your user and which the user has stored into QWBC.

   Your return to the *authenticate* call will be a string array with a maximum of four strings.

   The first member of the array is a session token, which could be a GUID or anything else that you want to use to identify the session. This token will be returned by QBWC in subsequent callbacks in the session.

   The second member of the string array can contain a variety of things.

   a. If the username and password in the authenticate call is invalid, you would supply the value "nvu".
   b. If on the other hand the user data is valid but you have no work to do for that user, you would supply the value "none".
   c. If you do have work to do for the that user, you can supply the full pathname of the company to be used in the current update.
   d. If you want to use whatever QuickBooks company is currently open at the client end, simply supply an empty string.

      The optional third member of the string array contains the number of seconds to wait before the next update. You would use this to in effect tell that QBWC client not to bother you for a specified time.

      The optional fourth member of the string array contains the number of seconds to be used as the MinimumRunEveryNSeconds time for your web service, which tells QBWC how frequently your web service needs to be contacted.

What happens when QBWC gets this string array? If the second member of the string array contains "none" or "nvu", QBWC will display a message, call closeConnection, and stop the session.

If the second member contains a string with any other value, including an empty string, QBWC will use that string as the path to the company file and will attempt to open it; an empty string means use the currently open company file.

If the connection attempt fails, QBWC calls connectionError, and uses the pathname string returned from connectionError to retry the connection until the connectionError callback tells it to stop.

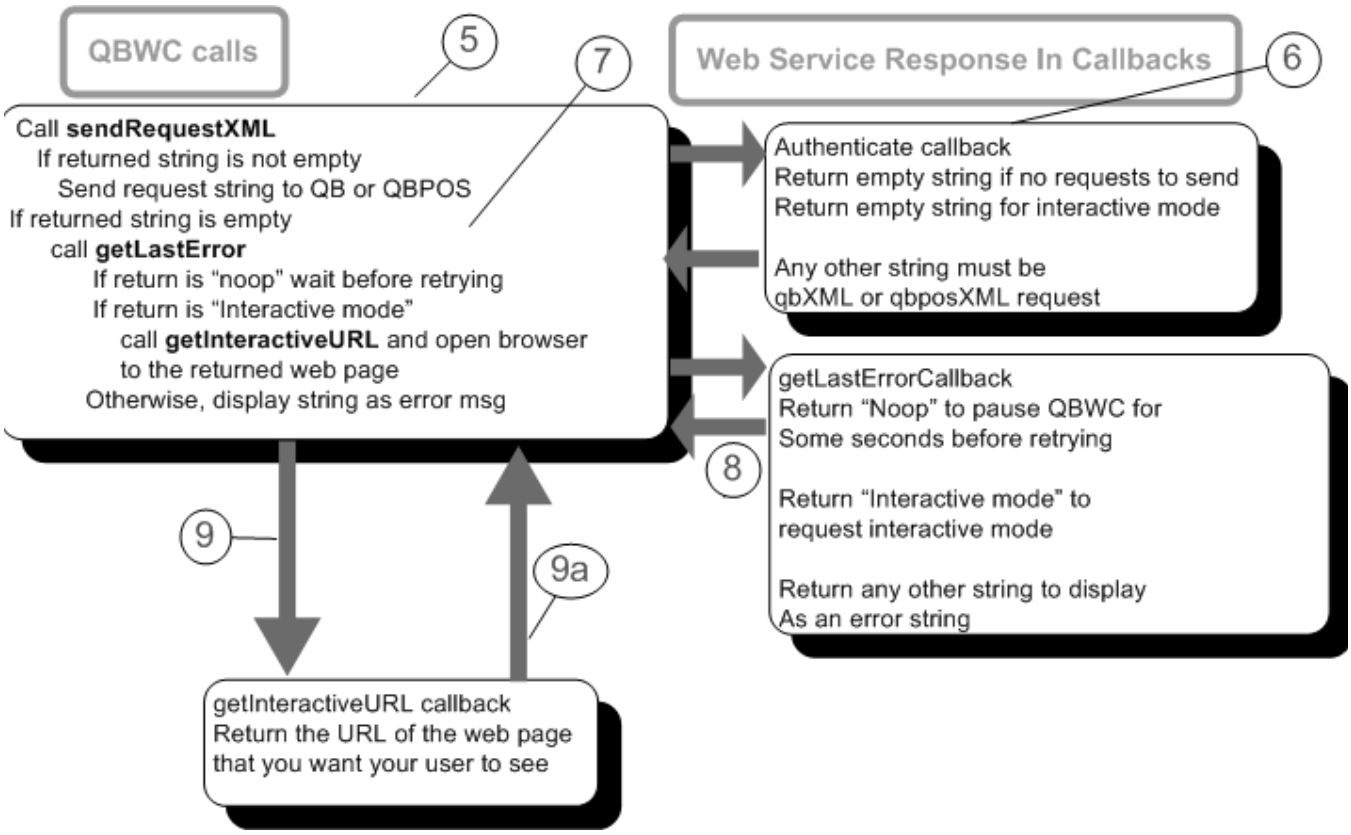


Figure 2-3 sendRequestXML, getLastError, and getInteractiveURL

What Figure 2-3 shows is what happens if the clientVersion and authenticate calls succeeded and resulted in further work. QBWC invokes OpenConnection and BeginSession on the indicated company and calls your sendRequestXML callback.

3. QBWC invokes sendRequestXML once the connection and session with QB or QBPOS is started. If your web service is a QB web service, the first time QBWC calls sendRequestXML in the session, it fills the *strHCPResponse* parameter with the results of a HostQuery, CompanyQuery, and PreferencesQuery, as this data can be useful for your web service when it constructs requests. If your web service is a QBPOS web service, the first time QBWC calls sendRequestXML it fills the *strHCPResponse* parameter with the string "HOSTQUERY/COMPANYQUERY/PREFQUERY is currently not supported in QBPOS." For all subsequent invocations of sendRequestXML during the session, strHCPResponse contains only an empty string, for both QBPOS and for QB.

4. If the return from sendRequestXML is not empty, QBWC passes the supplied string to the QB or QBPOS request processor to be handled. The string must be a validly constructed qbposXML or qbXML request message set

   If the return from your sendRequestXML callback is an empty string, QBWC calls getLastError to see whether your web service needs to:

   a. Report an error, in which case an error string would be returned from your getLastError callback
   b. Wait a few seconds before calling sendXMLRequest again, in which case the string "NoOp" would be returned.
   c. Start interactive mode, in which case the string "Interactive mode" would be returned.

5. As shown in Figure 2-3, if the string "Interactive mode" is returned at this point from getLastError, QBWC calls getInteractiveURL and starts a web browser open to the page specified in the return to this call.

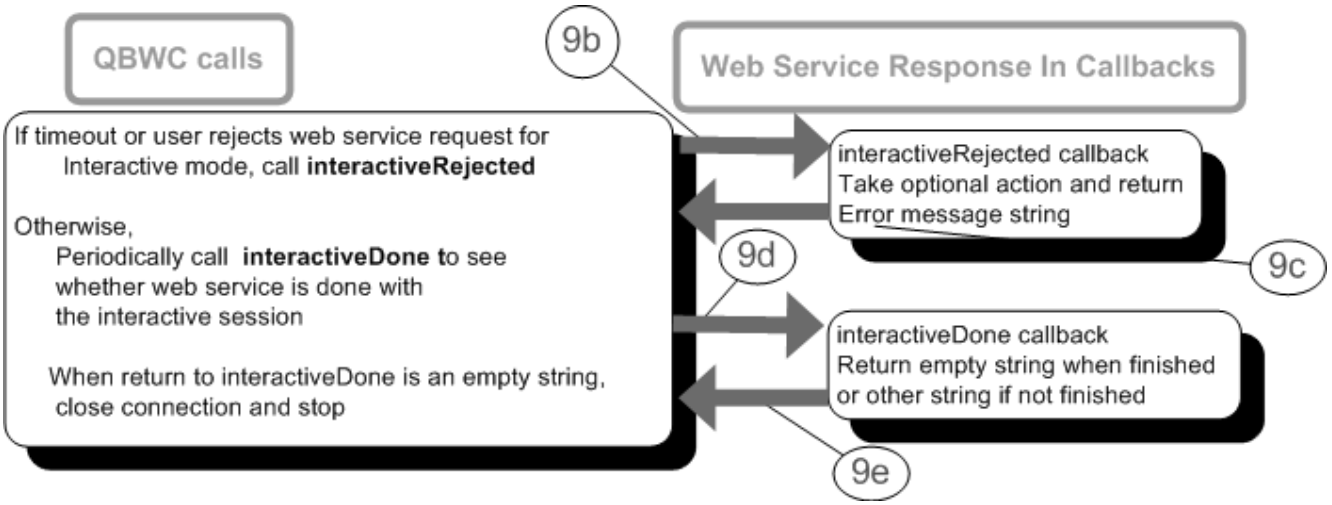


Figure 2-4 QBWC callbacks for interactive mode

6. As hinted at in Figure 2-4, just because the web service wants interactive mode doesn't mean it will get it. QBWC must ask the user if they authorize interactive mode by popping up a dialog. If the user responds with "no" or if the user is not there, then QBWC calls the interactiveRejected callback to let the web service know about this. Your callback does whatever you need it to do, but all you need to return is some sort of message you want displayed to your user.
7. If the user does authorize interactive mode, then the user will be taken to your web page and will be doing whatever supported activities are available at that page. You will handle user input and respond by invoking docontrol and/or doquery as described in the chapter "Interacting Directly with the Web Connector".
8. While the user is doing this, or until there is a timeout because the user got tired and went to lunch, QBWC will periodically call interactiveDone to see if you are finished.

If you are finished or if you know that there is a timeout, return an empty string in the callback to end the interactive mode session.

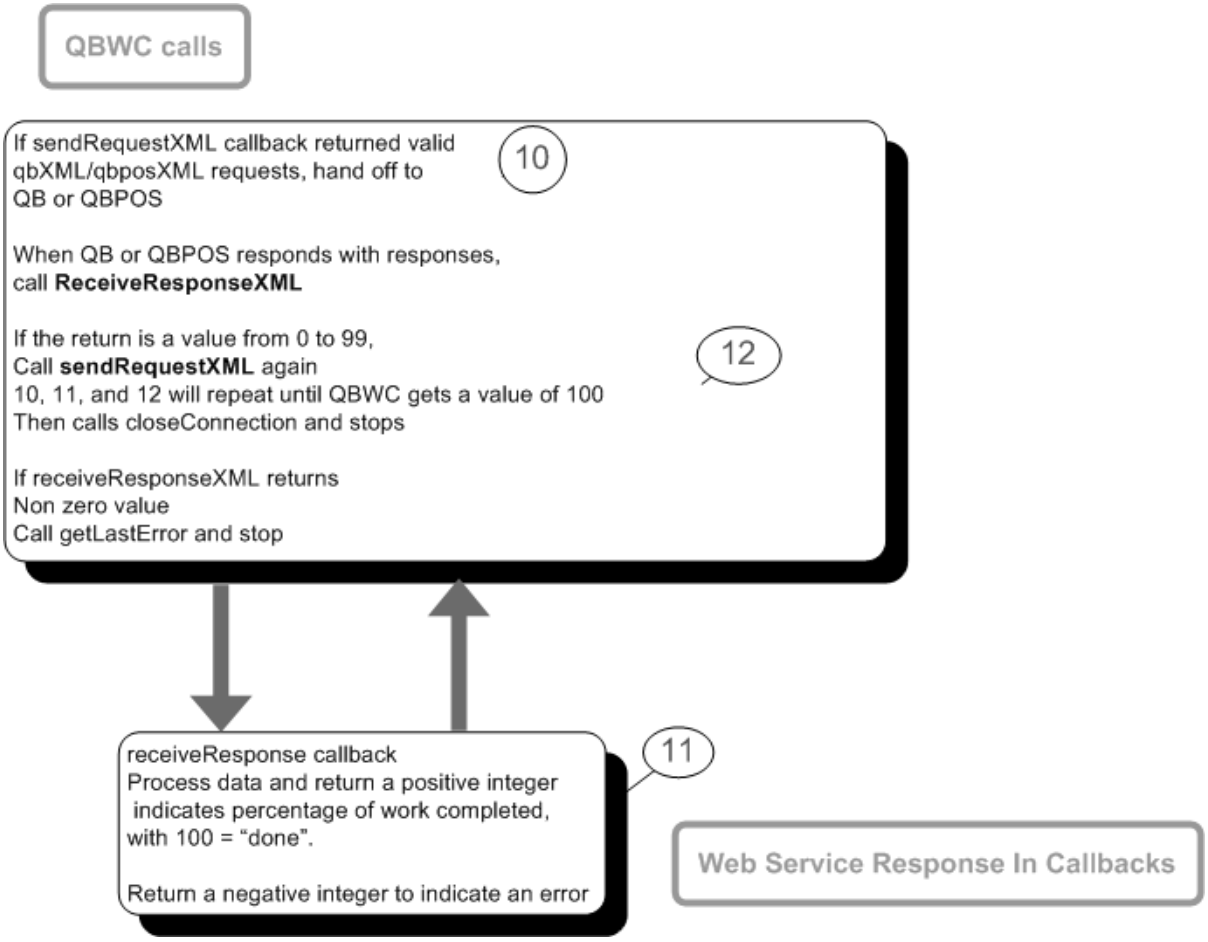


Figure 2-5 Finishing the communication

9. If there is no interactive mode, then the return from sendRequestXML contains qbXML or qbposXML requests. QBWC hands this off to QB or QBPOS and once it has responses, sends them to your web service by calling receiveResponseXML. The data returned by QB or QBPOS in response to the incoming requests is supplied in the QBWC receiveResponseXML, in the *response* parameter. Your callback returns a negative integer if there are any errors, such as out of sequence calls due to network problems. Otherwise, it returns a positive integer between 0 and 100, indicating the percentage of work done up to that point, with a value of 100 meaning that the update activity is finished. If there is work left, then QBWC calls sendRequestXML again to allow your web service to continue its work.
10. If the return from receiveResponseXML is a negative integer, QBWC calls getLastError to allow your web service to supply some message string to inform the user. This message is displayed by QBWC and then QBWC invokes closeConnection to end the session.

11. If not called prior to this point by some error condition, once all update activity is finished, as indicated by the web service's return to the receiveResponseXML call, then QBWC invokes closeConnection and ends the current update session.

CHAPTER 3
IMPLEMENTING A WEB SERVICE FOR QBWC

In general, there are two approaches to building SOAP web services and their clients (code-first and WSDL-first), with a fair degree of religious argument for each approach, however there is little dispute that when interoperability is important, a WSDL-first approach makes the most sense.

In the case of the Web Connector, the Web Connector is acting as a SOAP client which was written based on a canonical standard .NET web service's WSDL, so we followed the WSDL-first approach to building the client and as we went through the process of testing with .NET and Java web services we found the WSDL-first approach to be the most reliable method of ensuring that the web services we built to support the web connector remain compatible with the web connector.

In fact, we spent quite a bit of time tracking down what appeared to be bugs in the web connector (and were certainly less than graceful failures) that turned out to be caused by web services that did not conform to the WSDL we expected! We therefore strongly recommend a wsdl-first approach to developing a web service that works with the web connector!

Generating and Implementing the Service Skeleton with .NET

The .NET framework SDK includes the wsdl.exe tool for generating web service skeleton code or web service client proxy classes from a pre-existing WSDL, by default it will create a client proxy class, but with the /server switch will cause it to instead generate a skeleton class for use in a web service, we therefore start by generating the service skeleton, we'll use C# for our example, though the same principles apply for VB.NET, all that needs to change is the /language flag to wsdl.exe.

Wsdl.exe /server /language:cs http://developer.intuit.com/uploadedFiles/Support/QBWebConnectorSvc.wsdl

This will generate a file called QBWebConnectorSvc.cs containing an abstract class with methods for each of the SOAP WebMethods that the web connector expects:

```
// This source code was auto-generated by wsdl, Version=1.1.4322.2032.
using System.Diagnostics;
using System.Xml.Serialization;
using System;
using System.Web.Services.Protocols;
using System.ComponentModel;
using System.Web.Services;

/// <remarks/>
```

```
[WebServiceBindingAttribute(Name="QBWebConnectorSvcSoap",
                            Namespace="http://developer.intuit.com/")]
public abstract class QBWebConnectorSvc : WebService {

   /// <remarks/>
   [WebMethodAttribute()]
   [SoapDocumentMethodAttribute("http://developer.intuit.com/authenticate",
                            RequestNamespace="http://developer.intuit.com/",
                            ResponseNamespace="http://developer.intuit.com/",
                            Use=Description.SoapBindingUse.Literal,
                            ParameterStyle=SoapParameterStyle.Wrapped)]

   public abstract string[] authenticate(string strUserName,
                                         string strPassword);

   /// <remarks/>
   [WebMethodAttribute()]
   [SoapDocumentMethodAttribute("http://developer.intuit.com/sendRequestXML",
                            RequestNamespace="http://developer.intuit.com/",
                            ResponseNamespace="http://developer.intuit.com/",
                            Use=Description.SoapBindingUse.Literal,
                            ParameterStyle=SoapParameterStyle.Wrapped)]
   public abstract string sendRequestXML(string ticket,
                            string strHCPResponse,
                            string strCompanyFileName,
                            string qbXMLCountry,
                            int qbXMLMajorVers,
                            int qbXMLMinorVers);

   /// <remarks/>
   [WebMethodAttribute()]    [SoapDocumentMethodAttribute("http://developer.intuit.com/
                            receiveResponseXML",
                            RequestNamespace="http://developer.intuit.com/",
                            ResponseNamespace="http://developer.intuit.com/",
                            Use=Description.SoapBindingUse.Literal,
                            ParameterStyle=SoapParameterStyle.Wrapped)]

   public abstract int receiveResponseXML(string ticket, string response,
                            string hresult, string message);

   /// <remarks/>
   [System.Web.Services.WebMethodAttribute()]
   [SoapDocumentMethodAttribute("http://developer.intuit.com/connectionError",
                            RequestNamespace="http://developer.intuit.com/",
                            ResponseNamespace="http://developer.intuit.com/",
                            Use=Description.SoapBindingUse.Literal,
                            ParameterStyle=SoapParameterStyle.Wrapped)]
```

```
public abstract string connectionError(string ticket, string hresult,
                                       string message);

   /// <remarks/>
   [WebMethodAttribute()]
   [SoapDocumentMethodAttribute("http://developer.intuit.com/getLastError",
                            RequestNamespace="http://developer.intuit.com/",
                            ResponseNamespace="http://developer.intuit.com/",
                            Use=Description.SoapBindingUse.Literal,
                            ParameterStyle=SoapParameterStyle.Wrapped)]

   public abstract string getLastError(string ticket);

   /// <remarks/>
   [WebMethodAttribute()]
   [SoapDocumentMethodAttribute("http://developer.intuit.com/closeConnection",
                            RequestNamespace="http://developer.intuit.com/",
                            ResponseNamespace="http://developer.intuit.com/",
                            Use=Description.SoapBindingUse.Literal,
                            ParameterStyle=SoapParameterStyle.Wrapped)]

   public abstract string closeConnection(string ticket);
}
```

This file can then be added (Add>Add Existing Item) to a Web project in Visual Studio and then we can add a new web service (Add>Add Web Service) to inherit from this class and implement each of the interfaces defined by the parent class. Intellisense makes this quite easy, as shown in Figure 3-1.

Figure 3-1 Implementing the Web Service

The highlighted areas of Figure 3-1 represent changes made to the standard template created by Visual Studio, in particular note that we must add the WebService and WebServiceBinding attributes to the class (they are not inherited from the abstract base

class and the Namespace is particularly important) and, similarly, we must add [WebMethod] attributes to each of the web methods we define (again, they are unfortunately not inherited from the base class. For absolute completeness we could also copy over the SoapDocumentMethod attributes from the base class, but it seems as though those are either properly inherited or the defaults are correct in .NET.

Generating and Implementing the Service Skeleton with Java and Apache Axis

If you are not using .NET, the likely alternative is the Apache Axis web services framework. Similar to the .NET framework's wsdl.exe tool, Apache Axis includes the wsdl2java tool for creating a complete java source framework to implement a web service based on a WSDL. Again, by default WSDL2Java generates a Java Proxy class to communicate with a web service implementing the given WSDL, but the -server-side and -skeletonDeploy flags can be used to create the server-side framework, including an Axis Web Services Deployment Descriptor (wsdd) file:

java org.apache.axis.wsdl.WSDL2Java-server-side-skeletonDeploy http://developer.intuit.com/uploadedFiles/Support/QBWebConnectorSvc.wsdl

The result is a complete set of .java source files, of particular interest is the QBWebConnectorSvcSoapImpl.java file, shown below, where we will implement the logic for each of the six methods required by the web connector. The rest of the generated files require no modification, unless you wish to change the package name from the default com.intuit.developer they need only be compiled as part of the overall solution.

```
/**
* QBWebConnectorSvcSoapImpl.java
* This file was auto-generated from WSDL
 * by the Apache Axis 1.2.1 Jun 14, 2005 (09:15:57 EDT) WSDL2Java emitter.
 */

package com.intuit.developer;

   public class QBWebConnectorSvcSoapImpl implements
   com.intuit.developer.QBWebConnectorSvcSoap{

   public com.intuit.developer.ArrayOfString authenticate(
                        java.lang.String strUserName,
                        java.lang.String strPassword)
                        throws java.rmi.RemoteException {
      return null;
   }

   public java.lang.String sendRequestXML(java.lang.String ticket,
                        java.lang.String strHCPResponse,
                        java.lang.String strCompanyFileName,
                        java.lang.String qbXMLCountry,
```

```
                        int qbXMLMajorVers,
                        int qbXMLMinorVers)
                        throws java.rmi.RemoteException {
      return null;
   }

   public int receiveResponseXML(java.lang.String ticket,
                        java.lang.String response,
                        java.lang.String hresult,
                        java.lang.String message)
                        throws java.rmi.RemoteException {
      return -3;
    }

   public java.lang.String connectionError(java.lang.String ticket,
                        java.lang.String hresult,
                        java.lang.String message)
                        throws java.rmi.RemoteException {
    return null;
   }

   public java.lang.String getLastError(java.lang.String ticket)
                        throws java.rmi.RemoteException {
      return null;
   }

   public java.lang.String closeConnection(java.lang.String ticket)
                        throws java.rmi.RemoteException {
      return null;
   }
}
```

Unlike the .NET case, this is not an abstract class and we are free to simply write the appropriate logic to implement each of the six methods directly in the QBWebConnectorSvcSoapImpl.java file.

One interesting quirk here is that although both the .NET and Java services are generated from the same source WSDL, there can be a subtle difference in the way the return values are SOAP encoded - this is the reason for the <Style> tag in the Quickbooks Web Connector descriptor (QWC) file described earlier, it is our experience that a web service implemented as described here requires the DocWrapped style, while the .NET service works fine with the default Document style. Finally, if you choose to ignore the WSDL-first approach, Axis tends to default to a style of argument encoding which requires the RPC value for the <Style> field of the QWC file.

CHAPTER 4
BUILDING THE QWC FILE FOR YOUR USERS

Once your users find out about your web service and subscribe to it, they need to add it to their QB web connector so the web connector can access it on behalf of their QuickBooks or QuickBooks POS company. This assumes your user has already downloaded and installed the QB web connector! (See Chapter 7, “Understanding the End-User Experience and Setup.”)

How do your customers add your web service to their web connector? All they need to do is download your QWC file in any way you choose, including from your web application, and open it by double-clicking on it or by clicking Add an Application in the web connector UI (see Figure 4-1). The QWC file contains all of the connection information the web connector needs to connect to your web service, except the user password for your web service. Opening the QWC file automatically loads the QWC data into the web connector.

This chapter tells how to build the QWC file so it performs all this magic transparently for your user. It describes the required and optional content of the QWC file.

A Sample QWC File

Listing 4-1 shows a typical QWC file. This one happens to be used by the sample web service (WCWebService) provided with the QB SDK.

_____Listing 4-1 Sample QWC File

```
<?xml version="1.0"?>
<QBWCXML>
   <AppName>WCWebService1</AppName>
   <AppID></AppID>
   <AppURL>http://localhost/WCWebService/WCWebService.asmx</AppURL>
   <AppDescription>A short description for WCWebService1</AppDescription>
   <AppSupport>http://developer.intuit.com</AppSupport>
   <UserName>iqbal1</UserName>
   <OwnerID>{57F3B9B1-86F1-4fcc-B1EE-566DE1813D20}</OwnerID>
   <FileID>{90A44FB5-33D9-4815-AC85-BC87A7E7D1EB}</FileID>
   <QBType>QBFS</QBType>
   <Scheduler>
      <RunEveryNMinutes>2</RunEveryNMinutes>
   </Scheduler>
</QBWCXML>
```

In the QWC file above, notice that the root document element is <QBWCXML> (QuickBooks Web Connector XML) and within it are all of the required fields, along with one optional aggregate, <Scheduler>, which we provided just so you could see how to build that particular aggregate.All of these fields, and other optional fields are described in Table 4-1.

How Do I Set the QBWCXML Fields in the QWC File?

Table 4-1 shows the required and optional qbwcXML tags in the QWC and describes how to set the values for these.

Table 4-1 qbwcXML Tags

qbwcXML Element	Required?	Description
AppDescription	Y	This brief description of the application is displayed in the QB web connector (in the authorization dialog and in the main QBWC form under the application name. For best results we recommend a maximum description size of 80 characters.
AppDisplayName	N	QBWC will use this to display name in the QBWC UI. Otherwise, use <AppName> as usual. This is just for UI purpose. Update process still uses the <AppName> (or, AppUniqueName if provided).
AppID	Y, but can be empty	The AppID of the application, supplied in the call to OpenConnection. Currently QB and QB POS don't do much with the AppID, so you can supply the tag without supplying any value for it. However, you can supply an AppID value here if you want.
AppName	Y	The name of the application visible to the user. This name is displayed in the QB web connector. It is also the name supplied in the SDK OpenConnection call to QuickBooks or QuickBooks POS.
AppSupport	Y	The URL where your user can go to get support for your application. (Do not specify an Intuit site!) The domain name used in the AppSupport URL must match the domain name used in the AppURL. For internal development and testing only, you can specify localhost or a machine name in place of the domain name. If you specify a machine name, the machine name used for AppSupport must match the machine name used for AppURL.
AppUniqueName	N	If this element is available in QWC file, QBWC will not go into it's typical clone/replace mode for AppName and directly use the replace routine.
AppURL	Y	The URL of your web service. The domain name used in the AppSupport URL must match the domain name used in the AppURL. For internal development and testing only, you can specify localhost or a machine name in place of the domain name. If you specify a machine name, the machine name used for AppSupport must match the machine name used for AppURL. Unless you are using localhost (for development and testing only!) the domain name specified in AppSupport and in AppURL must match. To maintain a secure exchange of financial data, your AppURL must use the HTTP protocol over SSL (https://...). If you don't use HTTPS, the web connector won't connect with your web service.

qbwcXML Element	Required?	Description
AuthFlags	N	This element is used only for QuickBooks (QBType=QBFS). It specifies which QuickBooks editions are supported by your web service. By default, all editions are supported, including Simple Start edition. If you set this to exclude some QuickBooks edition, and QBWC attempts to connect to that excluded edition, there will be a connection error. Here are the values you can supply for <AuthFlags>: 0x0 (All, default) 0x1 (SupportQBSimpleStart) 0x2 (SupportQBPro) 0x4 (SupportQBPremier) 0x8 (SupportQBEnterprise) If you want to support several editions but not all, you can AND the values for the editions you want to support.
FileID	Y	FileID - the Web Connector stores this as an extension to the company record with a specific OwnerID known only to your web service (see below in the table for information on OwnerID) the first time the Web Connector connects to the company. The point is to ensure that your web service knows it is talking to the file it has always talked to. The OwnerID value is specific to only your web service. If you were to do a CompanyQuery, specifying your web service's OwnerID value in the query in the OwnerID query field, you would see something like this somewhere in the Ret: <DataExtRet> <OwnerID>{59028731-65dc-11db-bd13-0800200c9a66} </OwnerID> <DataExtName>FileID</DataExtName> <DataExtType>STR255TYPE</DataExtType> <DataExtValue>{72832751-65dc-11db-bd13-0800200c9a66} </DataExtValue> </DataExtRet> Notice that the name of this data ext is "FileID" and that the value of this data ext is the GUID that is your web service OwnerID. If a company file is accessed by 10 different web services, there would be 10 FileID data exts returned in the company query, each with a different DataExtValue.
IsReadOnly		Used to inform QBXMLRP2 (request processor) whether your service is reading data only, or is also writing data to the company. Specify true if write access is needed, or false if not.
Notify	N	Value of true will enable notification (pop up at systray) at app level. Anything else will disable notification.

qbwcXML Element	Required?	Description
OwnerID	Y	This is a GUID that represents your application or suite of applications, if your application needs to store private data in the company file for one reason or another. One of the most common uses is to check (via a QuickBooks SDK CompanyQuery request) whether you have communicated with this company file before, and possibly some data about that communication. You should generate one GUID per application only and not per application version or per QWC file! This private data will be visible to any application that knows the OwnerID. (See the QB SDK *Programmer's Guide* for more information on data extensions and OwnerID.)
PersonalDataPref	N	Used to inform QBXMLRP2 (request processor) whether your service requires access to personal/sensitive data. Specify pdpNotNeeded/ pdpOptional if it does not, and pdpRequired if it does.
QBType	Y	Specify the value QBFS if your web service is designed for use with QuickBooks Financial software. Specify the value QBPOS if your web service is designed for use with QuickBooks Point-of-Sale (QBPOS).
Scheduler	N	Your end user can specify the update interval in the QB web connector UI. You can optionally supply a default update interval by including the <Scheduler> aggregate, but be aware that the user can override your settings in the UI. You can specify one of two fields within this aggregate: <RunEveryNMinutes> specifies the number of minutes that should elapse between connections to your web service. <RunEveryNSeconds> specifies the number of seconds that should elapse between connections to your web service. Using this tag to set the update interval results the update interval to be displayed in the web connector UI as "Real Time".Notice that network/internet bandwith issues may not support the use of small values here, for example, an update interval of one second. If this is the case, the update will occur at the first available time after the specified elapsed time. The end user cannot specify an update interval of less than 1 minute, but they may discover they can change this in the registry. If you supply both <RunEveryNMinutes> and <RunEveryNSeconds> in your Scheduler aggregate, the web connector will simply pick the first one in the QWC and use that one. In practice, the Scheduler aggregate defines a minimum time between connections, and should not be taken to guarantee an update time. Also, remember that scheduled updates happen only when the web connector is running. If the user shuts down the web connector there will be no scheduled updates until the web connector is restarted by the user.

qbwcXML Element	Required?	Description
Style	N	The SOAP encoding style used by your web service. If not supplied, the default used is Document. Document is the standard encoding style used by .NET when the [WebMethod] attribute is applied to a function declaration. Optionally, you can specify the value DocWrapped. DocWrapped interoperates very well with Axis web services that are built as we recommend, using WSDL2Java to generate Java web classes from the standard WSDL used by the web connector (http://developer.intuit.com/uploadedFiles/Support/QBWebConnectorSvc.wsdl) Or, optionally, you can specify the value RPC. The RPC style is the standard encoding style used by Axis when a Java class is automatically converted to a SOAP service either through JWS or the Java2WSDL tool.
UnattendedModePref	N	Used to inform QBXMLRP2 (request processor) whether your service needs permissions to run in Unattended Mode supply the value umpRequired if it does, or umpOptional if it does not.
UserName	Y	The name your user must use to access your web service. The web connector uses this name when it invokes the authenticate call on your web service. To avoid disclosure of the password, there is no provision for any password field in the QWC file. Your user must enter the password into the web connector themselves, where it can be stored in the Windows registry securely via encryption.

- A new optional QWC parameter <CertURL> to provide means to provide certificate server for ssl certificates other than web server.

There are new optional QWC file parameters for three rp.AuthPreferences parameters IsReadOnly (true/false), UnattendedModePref (umpRequired/umpOptional), and PersonalDataPref(pdpNotNeeded/pdpOptional/pdpRequired).

Is the Order of the Tags Important?

No. You can specify the qbwcXML elements within the QWC in any order you like. However, we recommend you approximate the order shown in the sample file in order to maintain consistency for your users, in the event they should ever need to look at these files. The caveat here is that if you specify both RunEveryNMinutes and RunEveryNSeconds, the first element specified will be used.

Can I Start Developing Without All That “Cert” Stuff?

Yes. If you just want to develop the web service first without certs you can use HTTP and “localhost” instead of a domain name. Then obtain your cert and code to support certs later for use with HTTPS. The sample web service QWC file shown in Listing 4-1 shows the entry for using localhost with HTTP.

However, production use over the Internet requires HTTPS and certs.

Can I Run My Web Service in “Real Time”?

Sort of. You can specify the element <RunEveryNSeconds> within the Scheduler aggregate and set a value as low as one second. However, due to network latencies and server bandwith, this update frequency may not be attainable or even allowed. To guard against a too frequent update interval, you could implement a timer on your web service end to enforce your update frequency policy. Any web connector authenticate call that occurs too frequently as measured by the timer would be responded to by your web service with a indication that you have no requests for the web connector.

Can I Stop My Users From Running Updates in “Real Time”?

From the web connector UI, the user cannot specify a scheduled update time less than one minute. They can overwrite your “Real Time” setttings (updates less than one minute) only by specifying, from the UI, a time of one minute or greater. If they do this type of overwrite, they’ll have to remove your web service and re-add it if they want the higher frequency “real time” settings.

However, this only applies to the web connector UI. A sufficiently clever end user can always modify the QWC file simply by adding the RunEveryNSeconds field to the Scheduler aggregate in the QWC file. To guard against this, you could implement a timer on your web service end to enforce your update frequency policy. Any web connector authenticate call that occurs too frequently as measured by the timer would be responded to by your web service with a indication that you have no requests for the web connector.

Can I Specify Run EveryNSeconds and RunEveryNMinutes in one QWC File?

You can specify both of these tags within the Scheduler aggregate. However, only the first one will be used.

How Does the User Add the QWC File?

Your user can import the contents of the QWC (that is, can add the web service “pointed to” in the QWC file) in either of two ways:

- By double-clicking on the QWC file. The QWC file extension is registered so that the web connector will add it as a result of the double-click action.
- By clicking **Add an Application** and then browsing to the QWC file to open it. See Figure 4-1.

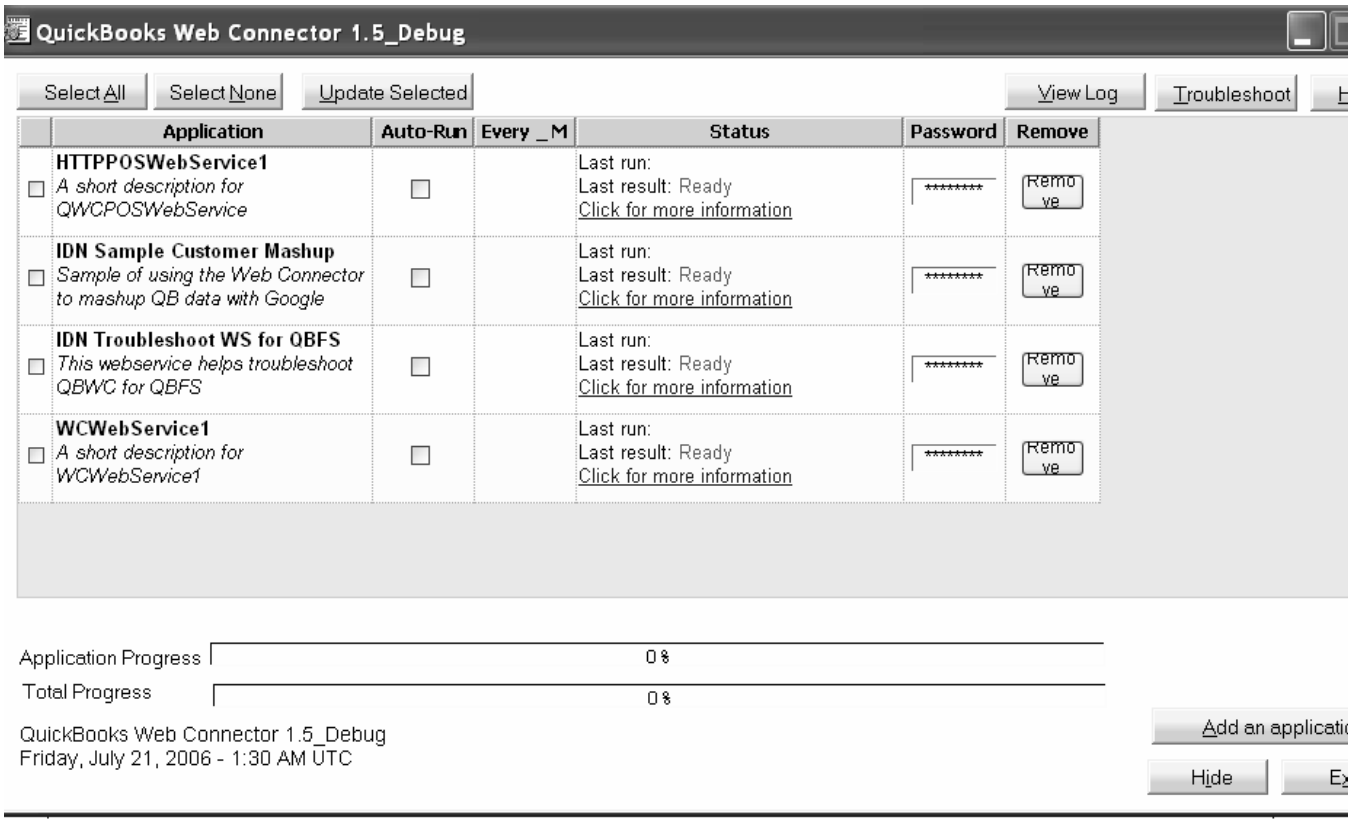


Figure 4-1 The QB Web Connector UI

CHAPTER 5
EXCHANGING DATA WITH QUICKBOOKS AND QBPOS

In many ways, once the user has setup the connection between QuickBooks and a web-based application, the heavy lifting is done, from then on, running the web connector from their desktop system should simply update QuickBooks with data from the web service.

IMPORTANT

Your web service application must not manually build SOAP headers (e.g., <soap:Envelope>, <soap:Body> etc.) before sending it to the QuickBooks Web Connector, for example, via sendRequestXML(). Your SOAP Engine should automatically wrap these headers around the xml sent by your web service.

A Note About the Required NameSpace

You must keep the namespace as http://developer.intuit.com/ for all web services that communicate with QuickBooks Web Connector.

Data Exchange Considerations

There are, however, a few areas to think about here:

1. The ability for the web connector to communicate the details of an error that occurred at the web service is limited. Therefore the web application should have a page the user can visit to examine the results of at least the most recent communication session with QuickBooks, if not an archive of previous connection sessions. We suggest that the <AppSupport> url provided in the QWC file land the user on a page that is populated with information about their previous connection, what data was exchanged and details about any errors that may have occurred with advice on how to resolve it (i.e. if a SalesReceiptAdd failed due to an item not being found, provide the user with the option to dynamically create the item on the next synch, or to choose an existing QuickBooks item to use, etc.)
2. Because the connection with QuickBooks happens based on the user's choices on the desktop, the web-based application must locally store the information that should be updated with QuickBooks upon the next connection. Many users like to preview what data will be exchanged with QuickBooks and to configure what data should and should not be exchanged. Where possible, your web application should provide a mechanism for the user to see the data queued for update to QuickBooks and for the user to enable or disable the update of specific records. So, for example, a storefront application that primarily pushes SalesReceipts might allow the user to choose not to download certain SalesReceipts until the items have been shipped, or the user explicitly marks them for update to QuickBooks.

3. Although true for any application which updates QuickBooks data, this is especially critical for web-based applications where an internet connection could go down at anytime: whenever an application is updating QuickBooks data, the QuickBooks error recovery mechanism (the NewMessageSetID, OldMessageSetID, etc. attributes on the QBXMLMsgsRq tag) should be used and the application should store any update request message until the response from that message has been fully processed. This way, if an error occurs the application can determine the status of the request it most recently sent to QuickBooks by re-sending it.
4. The SOAP interface defines a clear state model (see Figure 5-1) which applies on a per-user basis. However, the web imposes a somewhat stateless model, therefore it is important for the web service to implement an understanding of the state for a given user and the ticket that may or may not have been issued to the web connector on behalf of that user so that the web service can detect a mismatch of state with the web connector for a given user. For example, if the web service receives an authenticate() call for a particular user, but the service was expecting a call to receiveResponseXML() this is an excellent indication that information was lost and the web service must attempt to recover from the error (i.e. via the SDK's error recovery mechanism). More trivially, if the web service is waiting for a call to closeConnection() and receives a call to authenticate() then the web service can ignore the missing call, drop the old ticket for the user and behave as though it were not connected.

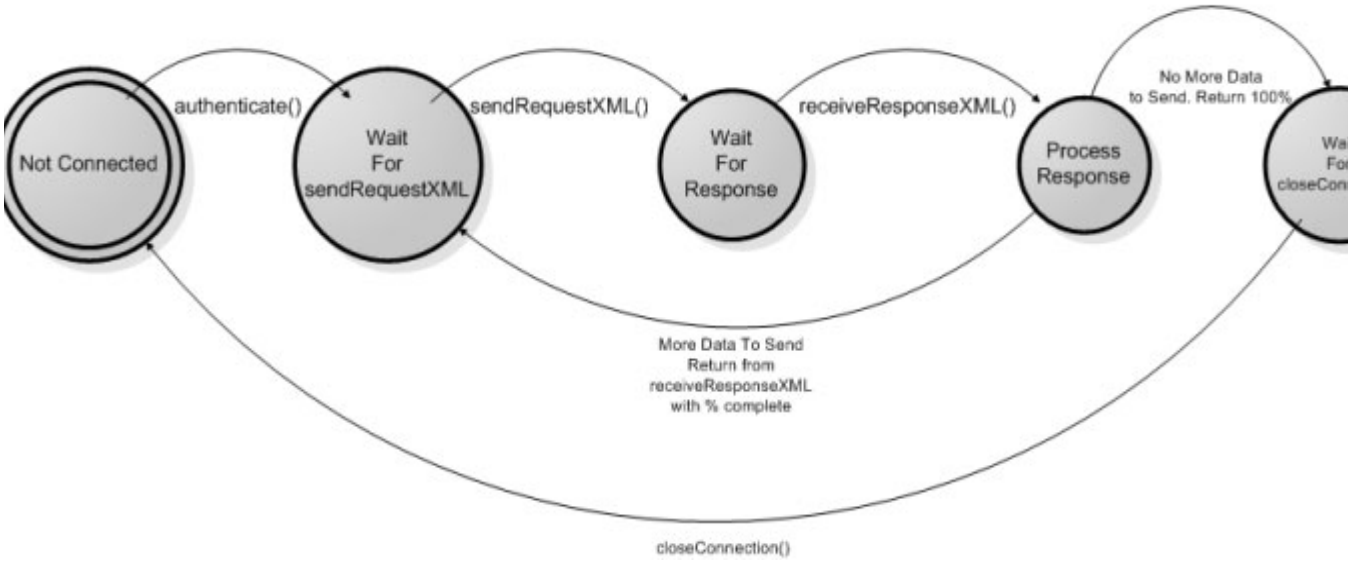


Figure 5-1 The Web Connector Service State Model

NOTE

This has been simplified: error states are not shown.

CHAPTER 6
INTERACTING DIRECTLY WITH THE WEB CONNECTOR

Beginning with QBWC 2.0, the Web Connector supports interactive mode. In interactive mode, QBWC opens a browser window to a web page specified by your web service. The user then uses the browser to do any of the available QuickBooks/QBWC activities that you choose to support.

There are two areas of activity that are available to your user:

- Your user can send information to QBWC to update web service parameters, start updates and so forth. You'll support these activities using the *docontrol* operation (which we'll describe shortly).
- Your user can invoke various pre-set SDK queries and also display certain QuickBooks list or transaction forms so the user can fill out forms right in QuickBooks. You'll use the *doquery* operation to support those.

A detailed list of supported activities is provided later in this chapter.

IMPORTANT

Version 2.0 of the web connector includes an asynchronous pluggable protocol handler that allows web-based applications to interact with the web connector via Javascript. With QBWC 2.0 the protocol handler works only with the Internet Explorer browser.

How to Implement Interactive Mode

How do you get your web service to support interactive mode? You need to do the following:

1. In the sendRequestXML when you want to start interactive mode, you need to return an empty string. This causes QBWC to call getLastError.
2. In the getLastError callback, again when interactive mode is desired, you need to return the string "Interactive mode". This causes QBWC to call getInteractiveURL so QBWC can open a browser to your web page.
3. In the getInteractiveURL callback, you need to return the URL of the web page that you want your end user to use. (Of course you'll need to develop that web page to accept user input and respond by issuing the proper docontrol and doquery operations.)
4. You need to implement the interactiveRejected callback to do whatever you want done in these cases where interactive mode is rejected. What could case such a rejection? Each time your web service attempts to start an interactive session, the user is prompted for permission by QBWC. If the user chooses NOT to give permission QBWC calls interactiveRejected. Or, suppose your thirsty user is out for coffee when your web

service tries to start an interactive session? There will be a timeout after a number of seconds and QBWC calls interactiveRejected.

5. You need to implement the interactiveDone callback. During the interactive mode session QBWC will periodically poll your web service to see if you're done yet by calling this callback. If, from your perspective, the session is done, if your web service knows the session has timed out, indicate this by returning an empty string (" ") from interactiveDone

This chapter describes how you can use the protocol handler to interact directly with QBWC, without having to wait on QBWC to call your web service callback methods.

Using docontrol to Change Web Service Behavior in QBWC

How do you use the docontrol operation to make web service changes in QBWC? You respond to user input by using qbwc:// protocol of the following form:

```
qbwc://<operation>/<request>?<parameters>
```

For <operation>, in the present case, specify docontrol, which sends control signals to QBWC to update web service parameters, start updates and so forth

Sample docontrol URLs

The following sample URLs show how the requests are placed in the URL:

```
qbwc://docontrol/WSExists?AppName=WCWebService

qbwc://docontrol/UpdateNow?AppName=WCWebService
qbwc://docontrol/UpdateNow?AppName=WCWebService3?async=true (for long
requests, this frees up the web browser)

qbwc://docontrol/SetSync?AppName=WCWebService&Interval=600

qbwc://docontrol/CountResults?AppName=WCWebService&Query=CustomerQueryRq

qbwc://docontrol/GetCompanyFile?AppName=WCWebService
```

For Time Consuming Updates, Use async=true

Normally, when web-app makes the call docontrol/Update, the Web Connector locks down some browsers for the duration of the update process. Thus, the web-app will have to wait while the Web Connector is processing the request. Setting async=true will release the web-app so it can perform other tasks while the Web Connector is processing the request.

Here is how to specify the parameter async=true in the call:

```
qbwc://docontrol/UpdateNow?AppName=WCWebService3?async=true
```

How to Get Status of the Update

For long updates, you might want to display status to the user. Here is how you do this. make the Update call async = True and keep pinging the response URL, as shown in the following example:

```
function UpdateNow() {
   var req = new Ajax.Request("qbwc://docontrol/
   UpdateNow?async=true&AppName=WCWebService3", {method:'get',
   onSuccess:processDataOne, onFailure:reportError});
}

//this code pings the response URL obtained from Web Connector
var pingURL ;
var i = "0" ;
function processDataOne(request)
{
   if ( i == "0" )
   {
      pingURL = URLDecode(request.responseText);
      i++;
   }
   if (request.status == 202)
   {
      //Url contains the Updatestatus as one of the parameter , which
      //contains the value;
      alert("Process Data: " + request.status + ': ' +
      URLDecode(request.responseText));
      setTimeout("var req1 = new Ajax.Request(pingURL,{method:'get',
      onSuccess:processDataOne, onFailure:reportError})", 1000);
   }
   else if (request.status == 200)
   {
      respURL = request.responseText;
      i= "0";
      alert("Process Data: " + request.status + ': ' +
      URLDecode(request.responseText));
   }
}
```

Requests for the docontrol operation

The *docontrol* operation offers the following requests:

Request	Description
AddWebService	POST qwcXML to qbwc://docontrol/AddWebService to snap a web service connection into the web connector. No parameters necessary in the query string.
CountResults	Returns the number of records that would satisfy a given unfiltered query. Takes two parameters: AppName and Query, where Query is the name of an SDK query (i.e. Query=CustomerQueryRq would return the number of customers in the company file). Useful for estimating sync times.

Request	Description
DisableUpdateInterval	Disables automatic update for the named web service. Takes one parameter: AppName.
GetCompanyFile	Returns the name of the company file currently open in QuickBooks. Useful for confirming the company file open is the one the user wishes to connect with your web service. Takes one parameter: AppName.
GetUpdateInterval	Gets the current update interval for the named web service. Takes one parameter: AppName.
RemoveWebService	Removes the named web service from WebConnector. Takes one parameter: AppName.
SetPassword	set the password for a web service. Takes two query string parameters: AppName and Password. The password is set for the web service specified by the AppName.
SetSync	Set the automatic sync interval for a given web service. Takes two query string parameters, AppName and Interval, with the interval provided as a number of seconds between updates.
UpdateNow	Begin an update immediately for the named web service. Takes one parameter: AppName.
WSExists	returns true if the web connector has a web service with the given AppName. Takes one parameter, AppName.

In all cases, the AppName parameter should specify the AppName (or, if supplied in the qwcXML that snapped the web service into the web connector, the AppUniqueName) of your web service.

Using doquery to Invoke Pre-Set SDK Requests

To invoke the pre-set SDK requests in response to user input, you would use this syntax:

```
qbwc://<operation>/<request>?<parameters>
```

For <operation>, in the present case, specify doquery, which uses the specified pre-set SDK queries from QuickBooks.

The following preset queries are available, all take a single parameter, a sessionID which is the GUID of an existing communication session between your web service and QuickBooks. The intent here is for support of interactive mode support during a session

The *doquery* operation offers the following requests, returning all elements of the response XML except in those cases where specific fields are listed:

Request	Description
AccountQuery	queries for active accounts
CustomerBillAddressQuery	Same as above, but including the BillingAddress for the customer
CustomerQuery	Queries for active customers returning the ListID, Name, FullName, Sublevel, Balance and TotalBalance for each customer.
EmployeeQuery	Queries for active employess returning ListID, Name, PrintAs, Phone, Mobile, Pager, PagerPIN, Email, EmployeeAddress, UseTimeDataToCreatePaychecks, IsUsingTimeDataToCreatePaychecks.
ItemDiscountQuery	queries for active discount items

Request	Description
ItemFixedAssetQuery	queries for active fixed asset items
ItemGroupQuery	queries for group items
ItemInventoryAssemblyQuery	queries for active assembly items
ItemInventoryQuery	queries for active inventory items
ItemNonInventoryQuery	queries for active non-inventory items
ItemOtherChargeQuery	queries for active other charge items
ItemPaymentQuery	queries for active payment items
ItemQuery	queries for active items, returning ListID, Name, and FullName.
ItemSalesTaxGroupQuery	queries for active sales tax group items
ItemSalesTaxQuery	queries for active sales tax items
ItemServiceQuery	queries for active service items
ItemSubtotalQuery	queries for active subtotal items
ListDisplayAdd	executes a list display add request for the list type specified in the type query parameter
ListDisplayMod	executes a list display mod request for the list type specified in the type query parameter, showing the list item provided in the ListID query parameter.
TxnDisplayAdd	a transaction display add request for the transaction type specified in the type query parameter
TxnDisplayMod	executes a Transaction display mod request for the transaction type specified in the type query parameter, showing the list item provided in the TxnID query parameter.
VendorAddressQuery	Queries for active vendors, returning ListID, Name, CreditLimit, Balance, and VendorAddress/VendorAddressBlock.
VendorQuery	Queries for active vendors, returning ListID, Name, CreditLimit, Balance, Phone, Fax , Email, and Contact fields for each.

CHAPTER 7
Understanding the End-User Experience and Setup

In order for your web-based application to be used successfully by your user, the user usually needs to do some initial setup and configuration. This chapter provides the end user view of the tasks that need to be done.

Initial End User Setup

The end user starts off by learning about your web service and the possibility of integrating it with their QuickBooks or QuickBooks POS company. The user then contacts you to establish an account to access and use your web service. At this point, the following user-driven processes must occur:

1. The user must download and install the web connector. You can host the download of the web connector yourself, but we recommend that you point the user to the appropriate Intuit site for the download because that site is guaranteed to have the latest web connector. The URL http://marketplace.intuit.com/webconnector will provide a description of the web connector as well as a link to download and install it.
2. From your web site, the user must download a QWC file provided by you. (The contents and structure of the QWC file are described in Chapter 4, “Building The QWC File for Your Users.”). This QWC file should be custom-generated for each individual user to provide the correct FileID as well as to provide the correct Username for the web connector to authenticate with.
3. After the download, the user opens the QWC file to add it to the web connector so that the web connector is able to connect to the web service. The user could just open the QWC directly without downloading it, but the recommended practice is for them to download it so they’ll have it in case they need to re-install the QWC into the web connector.
4. The user must configure the web-based application to appropriately exchange data with QuickBooks or QuickBooks POS. For example, a store front application may want to know whether items in the on-line store should be mapped 1:1 for items in QuickBooks or mapped to a single item in QuickBooks, similarly customers, sales tax, etc. To facilitate this, the web service should probably conduct an initial session with QuickBooks to obtain basic information pertinent to this configuration step (i.e. the chart of accounts, item list, etc. may be required by the application) so that user choices can be drawn directly from information in the users company file. This is important because the alternative can lead to the user entering incorrect information (i.e. typos, etc.) that can cause their subsequent data exchanges to fail or, worse, to create duplicate accounts, items, etc. that can be difficult or impossible to reverse.

CHAPTER 8
HANDLING ERRORS

In most communications between your web service and QuickBooks (via the web connector!), things for the most part will proceed happily after the web connector starts things off.

However, error conditions will occasionally occur and your web service must be able to handle them properly when they do. The error conditions your web service must handle will be one of the following types of error:

- The web connector cannot access QuickBooks for some reason, preventing further data exchange
- The web service received unexpected data from the web connector, preventing further data exchange
- The web service encounters an unexpected state: that is, it received an out-of-sequence web connector call indicating a potential disruption caused by network problems. Before further data exchange, error recovery must be performed

We'll describe how to handle these scenarios in this chapter.

The Web Connector Cannot Access QuickBooks

IMPORTANT

> Don't retry the same operation in response to the connectionError more than a couple of times. If the problem isn't resolved after a couple of tries, use getLastError to notify the user about the problem.

When the web service responds to the web connector's authenticate method call by indicating there is data to be exchanged with QuickBooks, the web connector calls the OpenConnection and BeginSession methods of the QuickBooks XML Request Processor.

If either of those calls fail for any reason, the web connector will display the error code and error message to the user, and it will let your web service know about the error via the call *connectionError*, which has the following signature:

```
string connectionError(string strTicket,
     string strHresult,
     string strMessage)
```

where the HRESULT is provided (in HEX) along with the message from the exception thrown by the request processor.

Typical causes for this type of error is that the company file requested could not be found or the file requested is not the file currently open in QuickBooks. But there could be numerous other causes: see Appendix A for a list of the possible errors.

How Your Web Service Should Respond to QB Access Errors

How should your web service respond to this category of error? Your web service should do one of two things:

(1) Return the string “done” which tells the web connector that the web service cannot proceed further and is stopping.

Or,

(2) IF your web service wants to try a different company, supply the company pathname in the returned string. (You can supply an empty string if you want to use whatever company file happens to be open.) The web connector will respond by attempting to connect to QuickBooks again using that supplied string.

Why Would a Web Service Try a Different Company?

Why would a web service perform the second of these actions instead of simply just stopping altogether? In practice this approach is used when the web service remembers the company file path from session to session (a recommended practice) and wants to have a fall-back to use whatever company file is currently open in QuickBooks (by responding to the connectionError call with an empty string).

This is not as haphazard as it might seem. When a web service is added to the web connector, the web connector stores a unique FileID as a private data extension in the specified company. As a result, the web service can always verify that it is talking to the expected company file simply by checking the CompanyRet returned to your web service in the web connector’s first sendRequestXML call in the data exchange sequence. (Check the data extension list for the expected FileID.)

The Web Service Gets Unexpected Data from Web Connector

In some cases, your web service may receive a sendRequestXML or a receiveResponseXML call that contains unexpected data. For example, there may be an XML parse error, or an expired ticket, or other unexpected data from QuickBooks. If this happens, your web service must first tell the web connector that an error has occurred and then handle the follow-up getLastError call from the web connector.

How Your Web Service Should Respond to Unexpected Data

How do you tell the web connector that an error occurred, in the "opinion" of the web service? If the problem data was sent in the sendRequestXML call, simply return an empty string to the sendRequestXML call. If the problem data was sent in the receiveResponseXML call, simply return a negative value to the receiveResponseXML call.

The web connector responds to this error condition by calling the *getLastError* method:

```
string getLastError(string strTicket)
```

Your web service should respond to this call by returning a message string describing the problem and any other information that you want your user to see. The web connector writes this message to the web connector log for the user and also displays it in the web connector's Status column. The web connector will then terminate the connection to the web service by calling closeConnection.

The Web Service Encounters an Unexpected State

Your web service must maintain a certain state information during the current session. For example, if the web connector has just called sendRequestXML, then the web service should expect the next call to be either receiveResponseXML or getLastError (if your web service indicated an error when it responded to the sendRequestXML).

If this expected call sequence does not occur, for example, if sendRequestXML is called instead, or authenticate is called, this indicates some type of communication failure. The communication between the web connector and the web service has been disrupted in some unexpected way such as a network failure.

If you just sent in some queries, you can simply resend them. But if you sent requests that wrote data to QuickBooks, you don't want to blindly just send those same data-writing requests again. How do you know whether the requests you sent actually made it into QuickBooks, or whether the failure occurred before this happened?

To determine this, use the error-recovery capabilities provided in the QB SDK, as documented in the QB SDK Programmer's Guide, using the oldMessageSetID and newMessageSetID attributes as described in that document.

CHAPTER 9
HOW DO I TROUBLESHOOT PROBLEMS?

When the QB web connector runs into an error condition originating from web connector operations or from the web service itself, the web connector displays an error message. The various possible error messages, descriptions, and suggested remedies are listed in Appendix A, “Understanding and Responding to QBWC Error Codes.”

However, in many cases, you will need to start your troubleshooting by determining whether your user has a valid and working web connector and a working connection to the outside world (internet). To help you and your user test for a working installation, Intuit hosts a troubleshooting page that contains a dummy web service and a corresponding QWC file for it. This chapter briefly describes this troubleshooting page.

About Logging

The Web Connector supports three log levels:

- NONE = No logging
- DEBUG (default setting) = Logging + first 50 characters of request/response xml
- VERBOSE = Logging + complete request/response xml

How Do I Get to the Troubleshooting Page?

On the theory that it never hurts to state the obvious, let’s put it on record that you have to have a working Internet connection in order to get to the troubleshooting page. If this is the case, then you get to the QBWC troubleshooting main page by clicking on the Troubleshoot button in the web connector UI, as shown in Figure 9-1 on page 56.

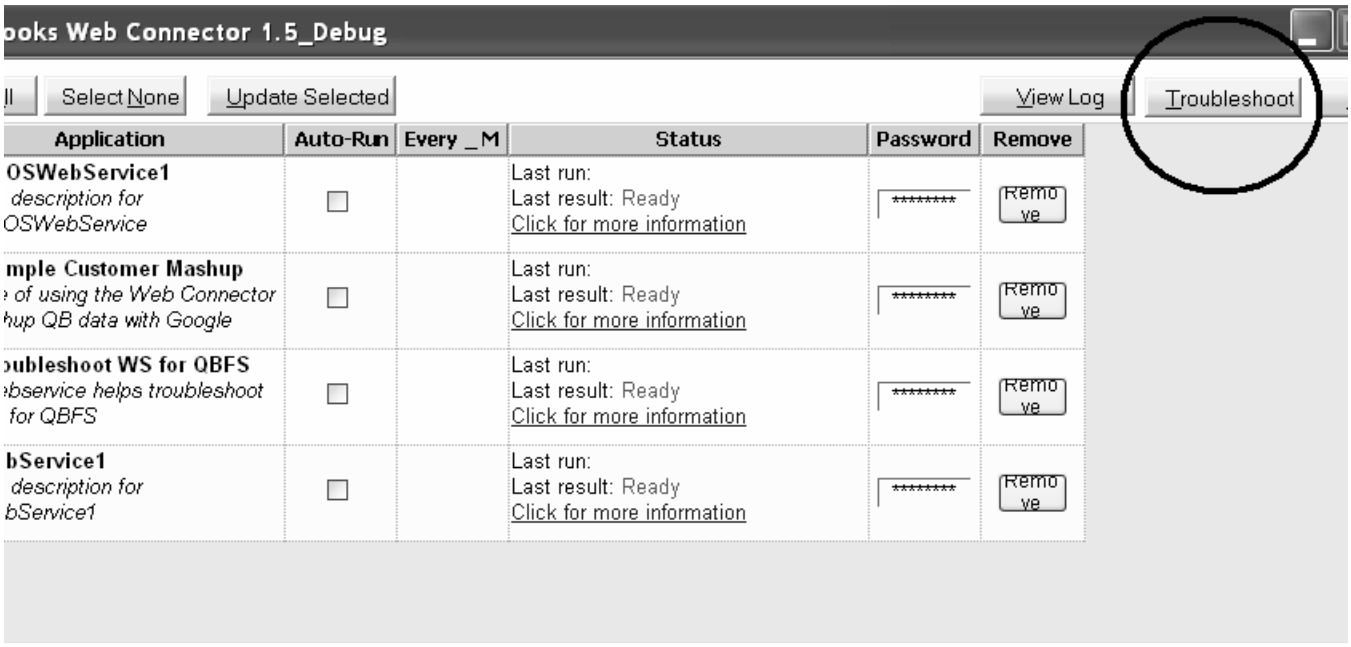


Figure 9-1 The Web Connector Troubleshoot button

The main troubleshooting page should pop up for you in the web browser (Figure 9-2 on page 57):

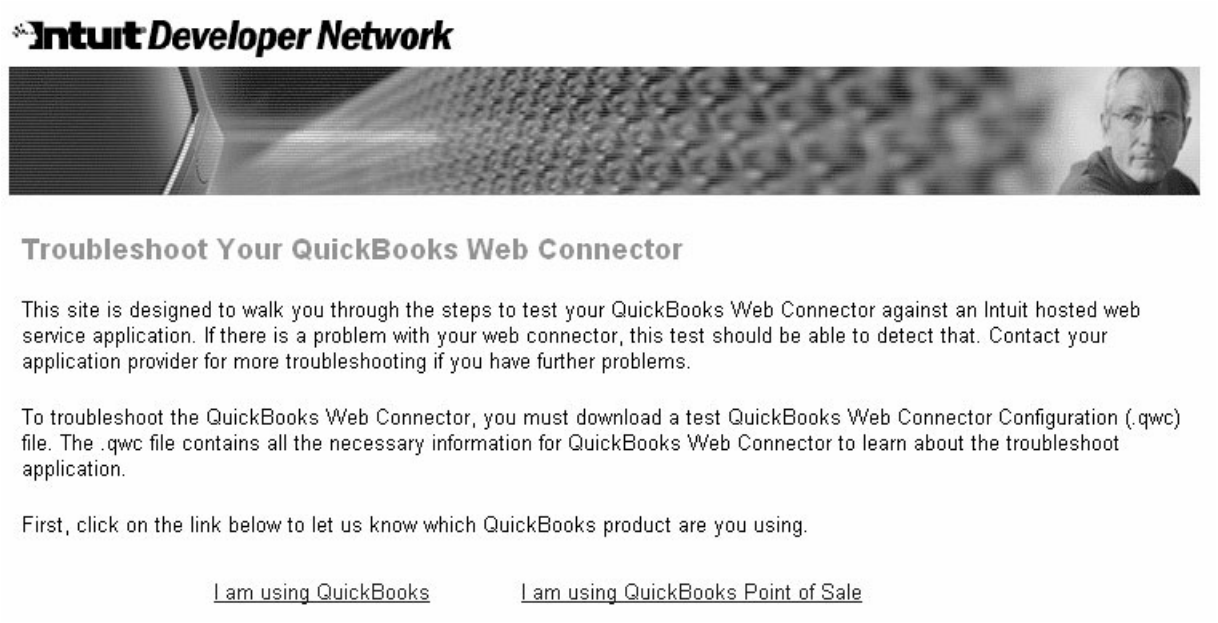
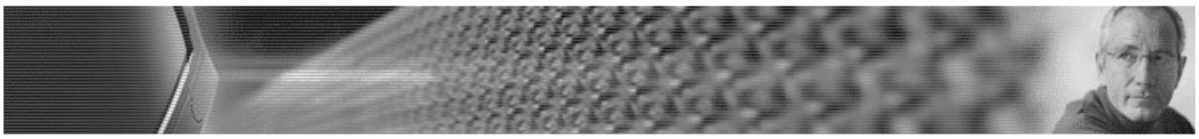


Figure 9-2 QBWC Troubleshooting Main Page

This routing page is pretty self explanatory. The QuickBooks link leads to the page where you get the QWC for a QuickBooks-oriented web service and the QuickBooks POS link leads to the page where you get the QWC for a QuickBooks POS-oriented web service. Take a look at the web “landing” pages corresponding to each of these links in Figure 9-3 on page 58and Figure 9-4 on page 59.



Note: Once the application is loaded in the QuickBooks Web Connector, in the password text box for this application enter the word **password** (all lower case)

For QuickBooks:

1. Make sure QuickBooks is started on your system before you proceed with this test.
2. Download the .qwc file.

   This .qwc file is configured to point to an intuit-hosted test application (web service). When loaded into the QuickBooks Web Connector, the web service will show up as "IDN Troubleshoot WS for QBFS."
3. From the QuickBooks Web Connector, click the check box next to "IDN Troubleshoot Web Service for QBFS" and click "Update Selected."

Select All | Selec: None | Update Selected | Troubleshoot | Help

	Application	Auto-Run	Every _ Min	Status	Password	Remove
☐	**IDN Troubleshoot WS for QBFS** *This webservice helps troubleshoot QBWC for QBFS* For support	☐		Last run: Last result: Ready Click for more information	********	Remove
☐	**IDN Troubleshoot WS for QBPOS** *This webservice helps troubleshoot QBWC for QBPOS* For support	☐		Last run: Last result: Ready Click for more information	********	Remove

4. View the web service log to verify that the web service received data from QuickBooks Web Connector.

   Result:

Figure 9-3 Troubleshooting page for QuickBooks

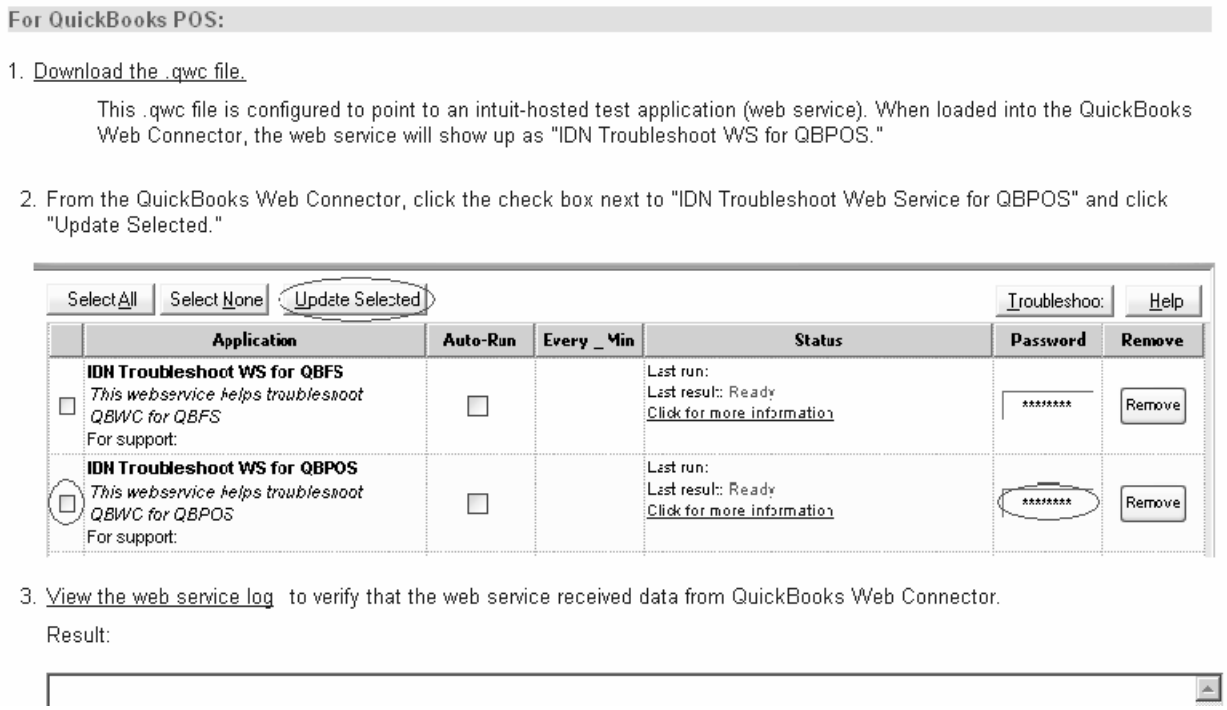


Figure 9-4 Troubleshooting page for QuickBooks POS

What Is Provided at the Troubleshooting Pages?

The test page for QuickBooks provides a dummy web service and QWC file that works with any QuickBooks company. All you have to do is double-click the QWC file to add it to the web connector and manually add the password displayed on the web page.

The test page for QuickBooks POS provides a dummy web service and QWC file that works with any QuickBooks POS company. As with the QuickBooks page, simply double-click the QWC file to add it to the web connector and manually add the password displayed on the troubleshooting page.

After you add the dummy web service and password, make sure QuickBooks or QuickBooks POS is running with a company open. Then perform an update on the web service and read the results. The results of the update indicate success or failure and are shown on the troubleshooting website.

CHAPTER 10
QBWC CALLBACK WEB METHOD REFERENCE

This chapter contains descriptions of each of the callback web methods your web service must implement. Notice that although the clientVersion is optional, we strongly recommend that you implement this as well.

IMPORTANT

Your web service application must not manually build SOAP headers (e.g., <soap:Envelope>, <soap:Body> etc.) before sending it to the QuickBooks Web Connector, for example, via sendRequestXML(). Your SOAP Engine should automatically wrap these headers around the xml sent by your web service.

The following callback methods should be implemented in your web service:

- "authenticate"
- "clientVersion"
- "closeConnection"
- "connectionError"
- "getInteractiveURL"
- "getLastError"
- "getServerVersion"
- "interactiveDone"
- "interactiveRejected"
- "receiveResponseXML"
- "sendRequestXML"

Each of these callback methods is described in the following sections.

IMPORTANT

The parameter names listed for the callback methods are important. You must use the parameter names as given in the method signatures.

authenticate

```
string[] authenticate(string strUserName, string strPassword)
```

Prompts the web service to authenticate the specified user and specify the company to be used in the session.

Parameters

strUserName	The web connector supplies the user name that you provided to your user in the QWC file to allow that username to access your web service.
strPassword	The web connector supplies the user password that you provided to your user and which was stored by the user in the web connector.

Return Value

Your callback must return A string array with 4 possible elements. In this returned string array:

- *The first element* contains either NONE or NVU (invalid user) or BUSY., or empty string, or a string that is the QB company file name. If your web service returns an empty string or any other string that is NOT *nvu* or *none*, or *busy*, that string will be used as the qbCompanyFileName parameter in the web connector's BeginSession call to QuickBooks.
- *The second element* enables the web service to postpone the update process. The value in this parameter determines the number of seconds by which the update will be postponed. For example if authRet[2]=60, the WebConnector will postpone the update process by 60 seconds. That is, the current update process is discontinued and will resume after 60 seconds.
- *The third elemen*t (optional) sets the lower limit for the Every_Min parameter (this parameter determines the interval the scheduler uses to run the updates when autorun is enabled). For example, if the third element =300 seconds, suppose a tries to set Every_Min=2 min using the UI of the WebConnector instance. In this case the result would be a popup that informs the user that the lower limit for this parameter is 300 seconds.and the Web Connector will automatically set the Every_Min parameter to 5 min (300 seconds).
- *The fourth element* (optional) contains the number of seconds to be used as the MinimumRunEveryNSeconds time.

IMPORTANT

In order to enable the web service to use authRet[2] & authRet[3], 'Auto Run' has to be enabled in the WebConnector

Usage

When a scheduled update occurs for your web service, or when the user clicks Update Selected in the web connector, the web connector calls your web service's authenticate method, supplying the user name and password required for your user to access your web service. Your web service validates the user specified in the authenticate call and returns a string array containing the values described previously under "Return Values".

Sample behavior of authRet[2] & authRet[3] is as follows:

Example 1. Suppose authRet[1]="", authRet[2]="x" seconds and authRet[3]=""
Result: the update is postponed by x seconds. In the QBWC status window this will be shown:

Last Result- Update postponed by application.

Example 2. authRet[1]="", authRet[2]="", and authRet[3]="y" seconds.
Result: The minimum limit for the Every_Min parameter is set. In the QBWC status window, the value of the Every_Min field will be shown set to the value in authRet[3], if the previous Every_Min value is lesser than the current value in authRet[3]. The Last Run time and the Next Run time is also shown. (Here Next Run Time= Last Run Time + y seconds)

Example 3. authRet[1]="", authRet[2]= "x" seconds and authRet[3]= "y" seconds
Result: The update is postponed by x seconds and the minimum limit for the Every_Min parameter is set. In the QBWC status window, the following are shown: Last Result- Update postponed by application.Last Run time and the Next Run time is also shown. (Here Next Run Time= Last Run Time + x seconds). Every_Min field is set to the value in authRet[3], if the previous Every_Min value is lesser than the value in authRet[3].

Example 4. authRet[1]="NONE/NVU/BUSY" , authRet[2]= "x" seconds and authRet[3]=""
Result: The update is postponed by x seconds. In the QBWC status window the following will be shown: Last Result- Update postponed by application.

Example 5. authRet[1] ="NONE/NVU/BUSY", authRet[3]= "y" seconds and authRet[2]=""
Result: The minimum limit on the Every_Min parameter is set and the update is stopped. In the QBWC status window the following will be shown: Last Result- No Data Exchange/ Invalid password for username/ Application Busy is displayed based on the value in authRet[1].Last Run Time and Next Run Time is displayed, Here Next Run Time=Last Run Time +y seconds. Every_Min field is set to the value in authRet[3], if the previous Every_Min value was lesser than the value in authRet[3].

Example 6. authRet[1]= "NONE/NVU/BUSY", authRet[3]= "y" seconds and authRet[2]="x" seconds.
Result: The minimum limit on the Every_Min parameter is set and the update is postponed. In the QBWC status window the following will be shown: Last Result- Update postponed by application. Every_Min field is set to the value in authRet[3], if the previous Every_Min value was lesser than the value in authRet[3]. Last Run Time and Next Run Time is displayed. Here Next Run Time=Last Run Time +x seconds

Example 7. authRet[1] =””, authRet[3]= “” seconds and authRet[2]=””
Result: Update will complete successfully.

Sample Code (C-sharp)

The following code sample is taken from the QB SDK sample program WCWebService. It responds to the authenticate method by creating a session GUID to be used as the session token and stores it in the first element in the string array to be returned to the web connector. The sample then compares the username and password to the expected hardcoded value, which is a bit hokey, but good enough for the purposes of this sample! If the supplied values are valid, the second element in the returned string array is set to empty string, which tells the web connector to use the currently open company.

```
[WebMethod]
      public string[] authenticate(string strUserName, string strPassword)
      {
         string evLogTxt="WebMethod: authenticate() has been called by QBWebconnector" + "
         evLogTxt=evLogTxt+"Parameters received:
         evLogTxt=evLogTxt+"string strUserName = " + strUserName + "
         evLogTxt=evLogTxt+"string strPassword = " + strPassword + "
         evLogTxt=evLogTxt+"

         string[] authReturn = new string[2];
         // Code below uses a random GUID to use as session ticket
         // An example of a GUID is {85B41BEE-5CD9-427a-A61B-83964F1EB426}
         authReturn[0]= System.Guid.NewGuid().ToString();

         // For simplicity of sample, a hardcoded username/password is used.
         // In real world, you should handle authentication in using a standard way.
         // For example, you could validate the username/password against an LDAP
         // or a directory server
         string pwd="password";
         evLogTxt=evLogTxt+"Password locally stored = " + pwd + "
         if (strUserName.ToUpper().Trim().Equals("USERNAME") &&
                           strPassword.ToUpper().Trim().Equals(pwd.ToUpper()))
         {
            // An empty string for authReturn[1] means asking QBWebConnector
            // to connect to the company file that is currently openned in QB
            authReturn[1]="";
         }
         else
         {
            authReturn[1]="nvu";
         }
         // You could also return "none" to indicate there is no work to do
         // or a company filename in the format C:\full\path   o\company.qbw
         // based on your program logic and requirements.
         evLogTxt=evLogTxt+"
         evLogTxt=evLogTxt+"Return values: " + "
```

```
evLogTxt=evLogTxt+"string[] authReturn[0] = " + authReturn[0].ToString() + "
evLogTxt=evLogTxt+"string[] authReturn[1] = " + authReturn[1].ToString();
logEvent(evLogTxt);
return authReturn;
```

clientVersion

```
string clientVersion(string strVersion)
```

Optional callback allows the web service to evaluate the current web connector version and react to it. Not currently required to support backward compatibility but strongly recommended.

Parameters

strVersion	The version of the QB web connector supplied in the web connector's call to clientVersion.

Return Value

A string telling the web connector what to do next. Supply one of the following strings:

- Specify an empty string or Null if you want the web connector to proceed with the update.
- Specify a text string that begins with the characters "W:" if you want the web connector to display a WARNING dialog prompting the user to continue with the update or cancel it. The text string after the "W:" will be displayed in the warning dialog.
- Specify a text string that begins with the characters "E:" if you want the web connector to cancel the update and display an ERROR dialog. The text string after the "E:" will be displayed in the error dialog. The user will have to download a new version of the web connector to continue with the update.
- Supply a value of O: (O as in Okay, not zero, followed by the QBWC version supported by the web service). For example O:2.0. This tells the user that the server expects a newer version of QBWC than the user currently has but also tells the user which version is needed.

Usage

When the web connector user clicks on Update Selected with your web service selected, or when a scheduled update occurs, the web connector begins the communication by calling clientVersion.

If your web service does not implement this callback method, the web connector simply proceeds to the update by calling *authenticate*.

If your web service does implement the clientVersion callback, the web connector will continue with the update, cancel it, or warn the user, depending on the information it receives from your web service. (See above under "Return Value".)

Sample Code (C-sharp)

The following code sample is taken from the QB SDK sample program WCWebService. It responds to the clientVersion method by comparing the web connector version to a minimum value and a recommended value. If the version is less than the recommended value, a warning string is returned; if the version is less than the minimum supported value, and error string is returned. Otherwise, an empty string is returned to allow the web connector to continue.

```
   [WebMethod]
   public string clientVersion(string strVersion)
   {
      string evLogTxt="WebMethod: clientVersion() has been called " +
         "by QBWebconnector" + "
      evLogTxt=evLogTxt+"Parameters received:
      evLogTxt=evLogTxt+"string strVersion = " + strVersion + "
      evLogTxt=evLogTxt+"

      string retVal=null;
      double recommendedVersion  = 1.5;
      double supportedMinVersion = 1.0;
      double suppliedVersion=Convert.ToDouble(this.parseForVersion(strVersion));
      evLogTxt=evLogTxt+"QBWebConnector version = " + strVersion + "
      evLogTxt=evLogTxt+"Recommended Version = " + recommendedVersion.ToString() + "
      evLogTxt=evLogTxt+"Supported Min Version = " + supportedMinVersion.ToString() + "
      evLogTxt=evLogTxt+"SuppliedVersion = " + suppliedVersion.ToString()+"
      if(suppliedVersion<recommendedVersion) {
         retVal="W:We recommend that you upgrade your QBWebConnector";
      }
      else if(suppliedVersion<supportedMinVersion){
         retVal="E:You need to upgrade your QBWebConnector";
      }
      evLogTxt=evLogTxt+"
      evLogTxt=evLogTxt+"Return values: " + "
      evLogTxt=evLogTxt+"string retVal = " + retVal;
      logEvent(evLogTxt);
      return retVal;
}
```

closeConnection

```
string closeConnection(string ticket)
```

Tells your web service that the web connector is finished with the update session.

Parameters

ticket	The ticket from the web connector. This is the session token your web service returned to the web connector's *authenticate* call, as the first element of the returned string array.

Return Value

Specify a string that you want the web connector to display to the user showing the status of the web service action on behalf of your user. This string will be displayed in the web connector UI in the status column.

Usage

When the update with the web service is completed, the web connector will notify the web service that it is done with the session it started by calling *closeConnection.* fifth of the six required methods for your web service:

Sample Code (C-sharp)

The following code sample is taken from the QB SDK sample program WCWebService. It doesn't do anything very interesting, just returns an "OK" message.

```
  [WebMethod]

        public string closeConnection(string ticket) {
      string evLogTxt="WebMethod: closeConnection() has been called by QBWebconnector" + "
      evLogTxt=evLogTxt+"Parameters received:
      evLogTxt=evLogTxt+"string ticket = " + ticket + "
      evLogTxt=evLogTxt+"
      string retVal=null;

      retVal="OK";
      evLogTxt=evLogTxt+"
      evLogTxt=evLogTxt+"Return values: " + "
      evLogTxt=evLogTxt+"string retVal= " + retVal + "
      logEvent(evLogTxt);
      return retVal;
}
```

connectionError

```
string connectionError(string ticket, string hresult, string message)
```

Tells your web service about an error the web connector encountered in its attempt to connect to QuickBooks or QuickBooks POS.

Parameters

ticket	The ticket from the web connector. This is the session token your web service returned to the web connector's *authenticate* call, as the first element of the returned string array.
hresult	The HRESULT (in HEX) from the exception thrown by the request processor.
message	The error message that accompanies the HRESULT from the request processor.

Return Value

Specify the string value "done" to indicate that your web service is finished. Or, if you want to retry the connection attempt on a different QuickBooks or QuickBooks POS company, specify the full pathname of that company in the return string. Any string other than "done" will be interpreted as the company name to be used in a retry attempt.

Usage

IMPORTANT

Don't retry the same operation in response to the connectionError more than a couple of times. If the problem isn't resolved after a couple of tries, use getLastError to notify the user about the problem.

When the web service responds to the web connector's authenticate method call by indicating there is data to be exchanged with QuickBooks, the web connector calls the OpenConnection and BeginSession methods of the QuickBooks XML request processor.

If either of those calls fail for any reason, the web connector will display the error code and error message from the request processor to the user, and it will let your web service know about the error via the *connectionError* call.

Sample Code (C-sharp)

The following code sample is taken from the QB SDK sample program WCWebService. Depending on the error, it either returns "Done" indicating it doesn't want to continue, or returns an empty string, meaning retry the connection attempt with the currently open company. (A real-world web service might want to instead maintain a record of the company file name or file names it expects and try to specify an expected filename.)

```
[WebMethod]
```

```
public string connectionError(string ticket, string hresult, string message)
{
   string evLogTxt="WebMethod: connectionError() has been called by QBWebconnector" + "
   evLogTxt=evLogTxt+"Parameters received:
   evLogTxt=evLogTxt+"string ticket = " + ticket + "
   evLogTxt=evLogTxt+"string hresult = " + hresult + "
   evLogTxt=evLogTxt+"string message = " + message + "
   evLogTxt=evLogTxt+"
   string retVal=null;
   // 0x80040400 - QuickBooks found an error when parsing the provided XML text stream.
   const string QB_ERROR_WHEN_PARSING="0x80040400";
   // 0x80040401 - Could not access QuickBooks.
   const string QB_COULDNT_ACCESS_QB="0x80040401";
   // 0x80040402 - Unexpected error. Check the qbsdklog.txt file
   const string QB_UNEXPECTED_ERROR="0x80040402";
   // Add more as you need...
   if(hresult.Trim().Equals(QB_ERROR_WHEN_PARSING)){
      evLogTxt=evLogTxt+ "HRESULT = " + hresult + "
      evLogTxt=evLogTxt+ "Message = " + message + "
      retVal = "DONE";

   }
   else if(hresult.Trim().Equals(QB_COULDNT_ACCESS_QB)){
      evLogTxt=evLogTxt+ "HRESULT = " + hresult + "
      evLogTxt=evLogTxt+ "Message = " + message + "
      retVal = "DONE";

   }
   else if(hresult.Trim().Equals(QB_UNEXPECTED_ERROR)){
      evLogTxt=evLogTxt+ "HRESULT = " + hresult + "
      evLogTxt=evLogTxt+ "Message = " + message + "
      retVal = "DONE";
   }

   else {
      // Depending on various hresults return different value
      // Try again with this company file
      evLogTxt=evLogTxt+ "HRESULT = " + hresult + "
      evLogTxt=evLogTxt+ "Message = " + message + "
      retVal = "";
   }

   evLogTxt=evLogTxt+"
   evLogTxt=evLogTxt+"Return values: " + "
   evLogTxt=evLogTxt+"string retVal = " + retVal + "
   logEvent(evLogTxt);
   return retVal;
}
```

getInteractiveURL

```
string getInteractiveURL(string wcTicket, string sessionID)
```

Lets your web service tell QBWC where to get the web page to display in the browser at the start of interactive mode.

Parameters

ticket — The ticket from the web connector. This is the session token your web service returned to the web connector's *authenticate* call, as the first element of the returned string array.

sessionID

Return Value

Your web service should return a message string containing the URL of the web page you want opened in the browser.

Usage

Used to support interactive mode. To start interactive mode, your web service indicates to QBWC that it wants to start interactive mode by returning an empty string from sendRequestXML, which causes QBWC to invoke getLastError. Then from getLastError you return the string "Interactive mode" to kick off the interactive session.

QBWC responds to this string by calling getInteractiveURL and opens a browser with the web page you specify in your return to that call.

getLastError

```
string getLastError(string ticket)
```

Allows your web service to return the last web service error, normally for display to the user, before causing the update action to stop.

Parameters

ticket	The ticket from the web connector. This is the session token your web service returned to the web connector's *authenticate* call, as the first element of the returned string array.

Return Value

Your web service should return a message string describing the problem and any other information that you want your user to see. The web connector writes this message to the web connector log for the user and also displays it in the web connector's Status column.

If you want your web service to go into interactive mode, you return the string "Interactive mode" and QBWC will respond by calling your web service's getInteractiveURL method, and open a web browser to the URL that you provide via this callback.

If you want the Web Connector to pause for an interval of time (currently 5 seconds) return the string "NoOp" from your sendRequestXML callback, followed by the string "NoOp" returned from your "getLastError callback. This will cause the QBWC to pause updates for 5 seconds before attempting to call sendRequestXML() again.

Usage

In some cases, your web service may receive a sendRequestXML or a receiveResponseXML call that contains unexpected data. For example, there may be an XML parse error, or an expired ticket, or other unexpected data from QuickBooks. If this happens, your web service must first tell the web connector that an error has occurred and then handle the follow-up getLastError call from the web connector.

How do you tell the web connector that an error occurred, in the "opinion" of the web service? If the problem data was sent in the sendRequestXML call, simply return an empty string to the sendRequestXML call. If the problem data was sent in the receiveResponseXML call, simply return a negative value to the receiveResponseXML call.

The web connector responds to this error condition by calling the *getLastError* method. After you return a string indicating the nature of the problem, the web connector will then terminate the connection to the web service by calling closeConnection.

Sample Code (C-sharp)

The following code sample is taken from the QB SDK sample program WCWebService. It just does a simple check then returns an error.

```
    [WebMethod]
    public string getLastError(string ticket)
    {
       string evLogTxt="WebMethod: getLastError() has been called by QBWebconnector" + "
       evLogTxt=evLogTxt+"Parameters received:
       evLogTxt=evLogTxt+"string ticket = " + ticket + "
       evLogTxt=evLogTxt+"

       int errorCode=0;
       string retVal=null;
       if(errorCode==-101){
          retVal="QuickBooks was not running!"; // just an example of custom user errors
       }
       else{
          retVal="Error!";
       }
       evLogTxt=evLogTxt+"
       evLogTxt=evLogTxt+"Return values: " + "
       evLogTxt=evLogTxt+"string retVal= " + retVal + "
       logEvent(evLogTxt);
       return retVal;
    }
}
```

getServerVersion

```
string getServerVersion(string ticket)
```

Provides a way for web-service to notify QBWC of it's version. This version string shows up in the More Information pop-up dialog in QBWC.

Parameters

ticket	The ticket from the web connector. This is the session token your web service returned to the web connector's *authenticate* call, as the first element of the returned string array.

Return Value

Your web service should return a message string describing the server version and any other information that you want your user to see.

Usage

Sample Code (C-sharp)

interactiveDone

```
string interactiveDone(string wcTicket)
```

Allows your web service to indicate to QBWC that it is done with interactive mode.

Parameters

ticket	The ticket from the web connector. This is the session token your web service returned to the web connector's *authenticate* call, as the first element of the returned string array.

Return Value

Your web service should return a message string with the value "Done" when the interactive session is over.

Usage

interactiveRejected

```
string interactiveRejected(string wcTicket, string reason)
```

Allows your web service to take alternative action when the interactive session it requested was rejected by the user or by timeout in the absence of the user.

Parameters

ticket	The ticket from the web connector. This is the session token your web service returned to the web connector's *authenticate* call, as the first element of the returned string array.
reason	The reason for the rejection of interactive mode.

Return Value

Return a message string to be displayed.

Usage

receiveResponseXML

```
int receiveResponseXML(string ticket, string response, string hresult,
string message)
```

Returns the data request response from QuickBooks or QuickBooks POS.

Parameters

ticket	The ticket from the web connector. This is the session token your web service returned to the web connector's *authenticate* call, as the first element of the returned string array.
response	Contains the qbXML response from QuickBooks or qbposXML response from QuickBooks POS.
hresult	The hresult and message could be returned as a result of certain errors that could occur when QuickBooks or QuickBooks POS sends requests is to the QuickBooks/QuickBooks POS request processor via the ProcessRequest call. If this call to the request processor resulted in an error (exception) instead of a response, then the web connector will return the corresponding HRESULT and its text message in the *hresult* and *message* parameters. If no such error occurred, *hresult* and *message* will be empty strings.
message	See above under *hresult*.

Return Value

A positive integer less than 100 represents the percentage of work *completed*. A value of 1 means one percent complete, a value of 100 means 100 percent complete--there is no more work. A negative value means an error has occurred and the Web Connector responds to this with a getLastError call. The negative value could be used as a custom error code.

Usage

When the web connector gets a response from QuickBooks or QuickBooks POS, it sends the response to the web service through *receiveResponseXML*. The web service should process the response and return an integer. A positive integer if you want it to serve as the estimated percent complete for the session, a negative integer if you want to indicate to the web connector that an error has occurred.

If the return value is positive, but less than 100 then the web connector knows that the web service has additional requests to be sent to QuickBooks, the connection status bar will be updated based on the percentage returned by the web service and the connector will call sendRequestXML again (this time leaving the strHCPResponse parameter as an empty string).

If the return value is negative, meaning an error occurred, then the Web Connector will call the web service's getLastError method (the fourth of the six required methods for your web service to implement). The getLastError method returns to the error message that should be presented to the user.

If the web service indicated it was not 100% complete, then the web connector will call sendRequestXML again, the qbXML returned by the web service will be sent to QuickBooks and the response sent to the web service via the receiveResponseXML method. This will repeat until an error occurs or the web service indicates that it is done exchanging data with QuickBooks.

There is no limit on the number of messages to QuickBooks Web Connector (QBWC). It depends on your application -- when in response to receiveResponseXML() you send a return value of 100 (which means 100% completed) then QBWC will stop calling sendRequestXML().

Sample Code (C-sharp)

The following code sample is taken from the QB SDK sample program WCWebService.

```
[ WebMethod(Description="response XML from QuickBooks",EnableSession=true) ]
public int receiveResponseXML(string ticket, string response, string hresult,
                              string message)
{
   string evLogTxt="WebMethod: receiveResponseXML() called by QBWebconnector" + "
   evLogTxt=evLogTxt+"Parameters received:
   evLogTxt=evLogTxt+"string ticket = " + ticket + "
   evLogTxt=evLogTxt+"string response = " + response + "
   evLogTxt=evLogTxt+"string hresult = " + hresult + "
   evLogTxt=evLogTxt+"string message = " + message + "
   evLogTxt=evLogTxt+"

   int retVal=0;
   if(!hresult.ToString().Equals("")){
      // if error in the response, web service should return a negative int
      evLogTxt=evLogTxt+ "HRESULT = " + hresult + "
      evLogTxt=evLogTxt+ "Message = " + message + "
      retVal=-101;
   }
   else{
      evLogTxt=evLogTxt+ "Length of response received = " + response.Length + "
      ArrayList req=buildRequest();
      int total=req.Count;
      int count=Convert.ToInt32(Session["counter"]);
      int percentage=(count*100)/total;
      if (percentage>=100){
         count=0;
         Session["counter"]=0;
      }
      retVal=percentage;
   }
   evLogTxt=evLogTxt+"
   evLogTxt=evLogTxt+"Return values: " + "
```

```
evLogTxt=evLogTxt+"int retVal= " + retVal.ToString() + "
logEvent(evLogTxt);
return retVal;
```

sendRequestXML

```
string sendRequestXML(string ticket,
                      string strHCPResponse,
                      string strCompanyFileName,
                      string qbXMLCountry,
                      int qbXMLMajorVers,
                      int qbXMLMinorVers)
```

The web connector's invitation to the web service to send a request.

Parameters

ticket The ticket from the web connector. This is the session token your web service returned to the web connector's *authenticate* call, as the first element of the returned string array

strHCPResponse Only for the first sendRequestXML call in a data exchange session will this parameter contains response data from a HostQuery, a CompanyQuery, and a PreferencesQuery request. This data is provided at the outset of a data exchange because it is normally useful for a web service to have this data. In the ensuing data exchange session, subsequent sendRequestXML calls from the web processor do not contain this data, (only an empty string is supplied) as it is assumed your web service already has it for the session.

strCompanyFileName The company file being used in the current data exchange.

qbXMLCountry The country version of QuickBooks or QuickBooks POS product being used to access the company. For example, US, CA (Canada), or UK.

qbXMLMajorVers The major version number (corresponding to the qbXML or qbposXML spec level) of the request processor being used. For example, the major number of the request processor released to support qbXML spec 6.0 would be "6".

qbXMLMinorVers The minor version number (corresponding to the qbXML or qbposXML spec level) of the request processor being used. For example, the major number of the request processor released to support qbXML spec 6.0 would be "0".

Return Value

If the web service has no requests to send, specify an empty string. If you want the Web Connector to pause for an interval of time (currently 5 seconds) return the string "NoOp", which will cause the Web Connector to call your getLastError callback: a "NoOp" returned from GetLastError will cause the QBWC to pause updates for 5 seconds before attempting to call sendRequestXML() again.

Any other string will be taken as a qbXML for QuickBooks or a qbposXML request for QuickBooks POS. The Web Connector sends the qbXML or qbposXML to QuickBooks or QuickBooks POS via the request processor's ProcessRequest method call.

Usage

After receiving the session token (ticket) returned from the web service in response to the authenticate call, the web connector establishes a connection to QuickBooks using QBXML Request Processor. The web connector then calls *sendRequestXML*, supplying in that call certain information about the QuickBooks or QuickBooks POS connection that the web connector has established.

If there is a problem establishing the connection the web connector does not call *sendRequestXML*, but instead calls *connectionError.*

Sample Code (C-sharp)

The following code sample is taken from the QB SDK sample program WCWebService. It logs the incoming HostQuery, CompanyQuery, and PreferencesQuery data and then invokes *buildRequest* (which is defined in the sample program WCWebService) to build a hardcoded set of requests.

```
[ WebMethod(Description="send request XML ",EnableSession=true) ]
public string sendRequestXML(string ticket, string strHCPResponse,
                             string strCompanyFileName,
                             string Country,
                             int qbXMLMajorVers,
                             int qbXMLMinorVers)
{
   if (Session["counter"] == null) {
      Session["counter"] = 0;
   }
   string evLogTxt="WebMethod: sendRequestXML() has been called by QBWebconnector" + "
   evLogTxt=evLogTxt+"Parameters received:
   evLogTxt=evLogTxt+"string ticket = " + ticket + "
   evLogTxt=evLogTxt+"string strHCPResponse = " + strHCPResponse + "
   evLogTxt=evLogTxt+"string strCompanyFileName = " + strCompanyFileName + "
   evLogTxt=evLogTxt+"string Country = " + Country + "
   evLogTxt=evLogTxt+"int qbXMLMajorVers = " + qbXMLMajorVers.ToString() + "
   evLogTxt=evLogTxt+"int qbXMLMinorVers = " + qbXMLMinorVers.ToString() + "
   evLogTxt=evLogTxt+"

   ArrayList req=buildRequest();
   string request="";
   int total = req.Count;
   count=Convert.ToInt32(Session["counter"]);
   if(count<total) {
      request=req[count].ToString();
      evLogTxt=evLogTxt+ "sending request no = " + (count+1) + "
      Session["counter"] = ((int) Session["counter"]) + 1;
   }
   else{
      count=0;
      Session["counter"]=0;
```

```
      request="";
   }
   evLogTxt=evLogTxt+"
   evLogTxt=evLogTxt+"Return values: " + "
   evLogTxt=evLogTxt+"string request = " + request + "
   logEvent(evLogTxt);
   return request;
}
```

APPENDIX A
UNDERSTANDING AND RESPONDING TO QBWC ERROR CODES

The following table lists the error codes and messages that can be returned from the QBWC during normal operation. A Description column is also provided with notes for you, the developer, and notes for your customers, in the event you wish to provide these details to them.

Table A-1 QBWC Errors and How to Handle Them

Error Code	Error Message	More Information
QBWC1000	The domain names for <AppName>'s service and support URLs do not match.	The AppURL and AppSupport URLs must use the same domain name. **Developer:** Please check your application's QWC file to make sure the <AppURL> and <AppSupport> both have same domain name. **End user:** There is an error in the web application definition. Please send the following to your application provider: - - Capture a screenshot (hit Alt+PrtSc while the error window is selected) of this error message. - Include the QWCLog.txt file (generally in C:\Documents and Settings\All Users\Application Data\Intuit\Quickbooks Web Connector\version directory) - Send a dump of the web connector registry settings. You can do this from a command prompt by running "regedit /E <AnyFilename>.reg HKEY_CURRENT_USER/Software/Intuit/QB web connector/<YourAppNameHere>", where AnyFilename is any name you want to give this dump file, and YourAppNameHere is the name of the provider's application.
QBWC1001	The application <AppName>'s service URL is an IP Address, it will not be loaded.	No AppURL can be IP-address based. The URL must contain a symbolic host name. **Developer:** In your application's QWC file, instead of an IP address, please use a host name for your <AppURL> value. **End user:** There is an error in the web application definition. Please send the same information requested above for error QBWC1000.

Error Code	Error Message	More Information
QBWC1002	The application <AppName>'s support URL is an IP Address, it will not be loaded.	No AppSupport URL can be IP-address based. The URL must contain a symbolic host name. **Developer:** In your application's QWC file, instead of IP address, use a host name for your <AppSupport> value. **End user:** There is an error in the web application's support address. Please send the same information requested above for error QBWC1000.
QBWC1003	The application <AppName>'s support URL (AppSupport) could not be reached.	No exception was thrown when trying to reach AppSupport URL. However, HttpWebResponse.StatusCode returned from web server was neither OK (Equivalent to HTTP status 200) or Accepted (HTTP status 202) **Developer:** Type the support URL in the error message in a web browser and see if you can reach the web page. If not, you need to make sure the URL is accessible from a web browser. **End user:** There is a possible problem in accessing the web application support page. Please send the same information requested above for error QBWC1000.
QBWC1004	The application <AppName>'s support URL (AppSupport) could not be reached.	An exception was thrown when trying to reach the AppSupport URL. In case of a WebException, a possible problem and status description should be shown with the error message. **Developer:** Type the support URL in the error message in a web browser and see if you can reach the web page. If not, you need to make sure the URL is accessible from a web browser. **End user:** There is a possible problem in accessing the web application support page. Please send the same information requested above for error QBWC1000.

Error Code	Error Message	More Information
QBWC1005	QuickBooks Web Connector failed to run. Some trace information was captured during the failure. Please see the QWCLog.txt file for trace information.	A System.IO.FileNotFoundException is caught at this point. QB web connector makes an attempt to re-create a new QWCLog.txt and dump the stack trace for debugging. **Developer:** QB web connector failed to run possibly because of not finding the right log file. Check QWCLog.txt for more information. Generally a stack trace is captured. QB web connector makes an attempt to re-create a new QWCLog.txt and dump the stack trace for debugging. One possibility is that for some reason the expected log directory was not found, and the current Windows user doesn't have permissions to create directories. **End user:** QB web connector failed to run possibly because it couldn't create the log file. One possibility is that for some reason the expected log directory was not found, and the current Windows user doesn't have permissions to create directories. Make sure your current Windows logon has sufficient permissions to create directories.
QBWC1006	QuickBooks Web Connector failed to run. Some trace information was captured during the failure. Please see QWCLog.txt file for trace information.	Any exception other than System.IO.FileNotFoundException has been caught at this point. **Developer:** QB web connector failed to run possibly because of any cause other than finding the right log file. Reproduce the issue with logging enabled (system tray - right click menu - enable logging). Then, check QWCLog.txt for more information. Generally a stack trace is captured in QWCLog.txt. **End user:** Make sure the log file directory and/or log file are not write protected. Otherwise, send the same information requested above for error QBWC1000.

Error Code	Error Message	More Information
QBWC1007	An error occurred when connecting to QuickBooks. <A Message from QuickBooks will appear here>. Please fix the problem and click Retry to try again.	Any time QB web connector connects to QuickBooks it will try the connection twice. This error code represents a failure during the second connection attempt. **Developer:** This is an error when QB web connector tried to connect again to QuickBooks. Have your user follow the instruction in the message (from QuickBooks) in the error screen. Most common causes are that QuickBooks is not running, and you need to start it, or make sure a company file is open and no modal dialog box is open. Otherwise, a message from QuickBooks is displayed. Finally, retry the task (generally an update operation, double-click/ download a QWC or load an application operation). **End user:** This is an error when QB web connector tried to connect to QuickBooks. Follow the instruction in the message (from QuickBooks) in the error screen. Most common causes are that QuickBooks is not running, and you need to start it, or make sure a company file is open and no modal dialog box is open. Otherwise, a message from QuickBooks is displayed. Finally, retry the task (generally an update operation, double-click/download a QWC or load an application operation)
QBWC1008	Unable to connect to QuickBooks. Task could not be completed. Reason: <A Message from QuickBooks will appear here>	Any time QB web connector connects to QuickBooks it will try the connection twice. This error code represents a failure during the second connection attempt. **Developer:** This is an error when QB web connector tried to connect again to QuickBooks. Have your user follow the instruction in the message (from QuickBooks) in the error screen. Most common causes are that QuickBooks is not running, and you need to start it, or make sure a company file is open and no modal dialog box is open. Otherwise, a message from QuickBooks is displayed. Finally, retry the task (generally an update operation, double-click/ download a QWC or load an application operation). **End user:** This is an error when QB web connector tried to connect to QuickBooks. Follow the instruction in the message (from QuickBooks) in the error screen. Most common causes are that QuickBooks is not running, and you need to start it, or make sure a company file is open and no modal dialog box is open. Otherwise, a message from QuickBooks is displayed. Finally, retry the task (generally an update operation, double-click/download a QWC or load an application operation).

Error Code	Error Message	More Information
QBWC1009	Unable to connect to QuickBooks. Task could not be completed. Reason: <A Message from QuickBooks will appear here>	Any time QB web connector connects to QuickBooks it will try the connection twice. This error code represents the fact that a failure occurred during the first connection attempt and then user chose to Cancel instead of Retry. **Developer:** Have your end user follow the instruction in the message (from QuickBooks) in the error screen. Most common causes are that QuickBooks is not running, and you need to start it, or make sure a company file is open and no modal dialog box is open. Otherwise, a message from QuickBooks is displayed. And finally retry the task (generally an update operation, double-click/download a QWC or load an application operation). **End user:** Follow the instruction in the message (from QuickBooks) in the error screen. Most common causes are that QuickBooks is not running, and you need to start it, or make sure a company file is open and no modal dialog box is open. Otherwise, a message from QuickBooks is displayed. And finally, retry the task (generally an update operation, double-click/download a QWC or load an application operation).
QBWC1010	Application <AppName> cannot be loaded. For security reasons only SSL (https) based applications are allowed.	AppURL needs to be SSL (https) based. **Developer:** Make sure to setup your application to support https. You could use http for development purposes in some case but certainly not in production. Currently QB web connector allows http and https for localhost only -- to provide ease with your development effort. However, it does not allow for http for any remote web servers. **End user:** Please send the same information requested above for error QBWC1000.
QBWC1011	Application named <AppName> does not exist in registry. It is possible that the view and registry is out of sync. Restart QuickBooks Web Connector.	Generally, QB web connector UI shows a snapshot of the applications listed in the system registry. Somehow the QB web connector is out of sync with this system registry. **Developer:** Have the end user exit from QB web connector and re-start it. Once re-started, if user does not see the application in the list, user may also need to manually add the application QWC file to the Web Connector. **End user:** You need to exit from QB web connector and re-start it. Once re-started, if you do not see the application in the list, you may also need to add the application's QWC file you received from the service provider to the Web Connector.

Error Code	Error Message	More Information
QBWC1012	Authentication failed due to error message: <An error message appears here>.	An exception was caught during the authenticate() call to application. This could be either a client or a server side problem. **Developer:** A call to the WebMethod authenticate() failed with an exception. Have your end user send you the QWCLog.txt file just after the error occurred. It should contain a stack trace for the exception caught. It's very possible the error originated on your server, please check your server logs for any web service exceptions. **End user:** Please send the same information requested above for error QBWC1000.
QBWC1013	Error connecting to QuickBooks. Returning error message to application. <Error message from QuickBooks appears here>.	QB web connector makes an attempt to connect to QuickBooks during update operation. This error code represents the fact that a failure occurred during this connection attempt. Notice that you can get this error if QuickBooks is running on Vista with UAC turned off. **Developer:** Your web application should have received a connectionError() call with the error message that QB web connector received from QuickBooks. You may need to have your end user do the steps according to the instruction in the error message from QuickBooks. Most common causes are that QuickBooks is not running, and you need to start it, or make sure a company file is open and no modal dialog box is open. And then retry the update operation. **End user:** There should be some information in the status window. Take a screenshot (press Alt+PrtSc while the window is selected) of the information and send it along with the QWCLog.txt file (generally in C:\Documents and Settings\All Users\Application Data\Intuit\Quickbooks Web Connector\version directory) to your application provider.
QBWC1014	Could not get Host/ Company/Preference Query response from QuickBooks. Job ending. <An error message from QuickBooks appears here>.	During update in process, QB web connector was unable to get and/or parse the Host/Company/Preference Query response from QuickBooks. **Developer:** Have your end user send you the QWCLog.txt file. It should contain a stack trace for the exception caught. **End user:** Please send the same information requested above for error QBWC1000.

Error Code	Error Message	More Information
QBWC1015	<An error message from QuickBooks>	An exception was caught during EndSession and CloseConnection call from QB web connector to QuickBooks. **Developer:** Have your end user send you the QWCLog.txt file. It should contain a stack trace for the exception caught. **End user:** pdate completed at this time but QB web connector could not end its communication with QuickBooks. Please send the same information requested above for error QBWC1000.
QBWC1016	No application was selected for update.	User clicked **Update** in Web Connector without first selecting any applications to update. **Developer:** Have end user select the application by checking the checkbox at the left of the application name and then hit "Update Selected" **End user:** Select the application by checking the checkbox at the left of the application name and then click **Update Selected**.
QBWC1017	The following applications could not be updated. Applications named <A list of web applications that were not updated at this time>.	When there are multiple applications being updated, this error shows a list of applications that was not updated successfully. **Developer:** This error code just lists the applications when multiple applications are being updated. You would need to get QWCLog.txt file from the user to determine what happened to each individual web applications. **End user:** Please send the same information requested above for error QBWC1000.
QBWC1018	No application available to select at this time.	User clicked on "Select All" button when there is no application loaded in QB web connector. **Developer:** User clicked on "Select All" button when there is no application loaded in QB web connector. User needs to load an application first. **End user:** You clicked on "Select All" button when there is no application loaded in QB web connector. You need to load an application first. You could either locate the QWC file that your application provide gave you and double-click it to load it in QB web connector or you could use Load button from QB web connector and browse to the QWC file your application provider sent you

Error Code	Error Message	More Information
QBWC1019	No application available to un-select at this time.	User clicked on "Select None" button when there is no application loaded in QB web connector. **Developer:** User clicked on "Select None" button when there is no application loaded in QB web connector. User need to load an application first. **End user:** You clicked on "Select None" button when there is no application loaded in QB web connector. You need to load an application first. You could either locate the QWC file that your application provide gave you and double-click it to load it in QB web connector or you could use Load button from QB web connector and browse to the QWC file your application provider sent you.
QBWC1020	There are scheduled jobs. Web Connector will not be able to run these jobs if you exit. Do you still want to exit?	User has a scheduled update service set in QB web connector when user attempted to exit. **Developer:** Either user need to un-select scheduling (Auto-Run) before exiting QB web connector or do a forced exit by choosing "Yes. **End user:** You need to either un-select scheduling (Auto-Run) before exiting QB web connector or do a forced exit by choosing "Yes".
QBWC1021	<An error message from QuickBooks>	An exception was thrown when trying to determine latest version of QBXMLRP supported by the QuickBooks running. **Developer:** There may be a problem trying to determine the latest version of QBXMLRP supported by the QuickBooks running. The error screen shot should show a hint and the QWCLog.txt file should show a stack trace. **End user:** Please send the same information requested above for error QBWC1000.

Error Code	Error Message	More Information
QBWC1022	An error occurred when connecting to QuickBooks. <An error message from QuickBooks appears here>. Please fix the problem and click OK to try again.	The error occurred when trying to search for FileID (while adding this application to registry) for this application. Any time QB web connector connects to QuickBooks it will try the connection twice. This error code represents a failure during the first connection attempt. **Developer:** This is an error when QB web connector tried to connect to QuickBooks. Have your user follow the instruction in the message (from QuickBooks) in the error screen. Most common causes are that QuickBooks is not running, and you need to start it, or make sure a company file is open and no modal dialog box is open. Otherwise, a message from QuickBooks is displayed. **End user:** This is an error when QB web connector tried to connect to QuickBooks. Follow the instruction in the message (from QuickBooks) in the error screen. Most common causes are that QuickBooks is not running, and you need to start it, or make sure a company file is open and no modal dialog box is open. Otherwise, a message from QuickBooks is displayed.
QBWC1023	Unable to connect to QuickBooks. Task could not be completed. Reason: <An error message from QuickBooks appears here>.	The error occurred when trying to search for FileID (while adding this application to registry) for this application. Any time QB web connector connects to QuickBooks it will try the connection twice. This error code represents a failure during the second connection attempt. **Developer:** This is an error when QB web connector tried to connect again to QuickBooks. Have your user follow the instruction in the message (from QuickBooks) in the error screen. Most common causes are that QuickBooks is not running, and you need to start it, or make sure a company file is open and no modal dialog box is open. Otherwise, a message from QuickBooks is displayed. And finally, retry the task (generally double-click/download a QWC or load an application operation). **End user:** This is an error when QB web connector tried to connect to QuickBooks. Follow the instruction in the message (from QuickBooks) in the error screen. Most common causes are that QuickBooks is not running, and you need to start it, or make sure a company file is open and no modal dialog box is open. Otherwise, a message from QuickBooks is displayed. And finally, retry the task (generally double-click/download a QWC or load an application operation).

Error Code	Error Message	More Information
QBWC1024	Unable to connect to QuickBooks. Task could not be completed. Reason: <An error message from QuickBooks appears here>.	The error occurred when trying to search for FileID (while adding this application to registry) for this application. Any time QB web connector connects to QuickBooks it will try the connection twice. This error code represents the fact that a failure occurred during the first connection attempt and then user chose to Cancel instead of Retry. **Developer:** Have your user follow the instruction in the message (from QuickBooks) in the error screen. Most common causes are that QuickBooks is not running, and you need to start it, or make sure a company file is open and no modal dialog box is open. Otherwise, a message from QuickBooks is displayed. And finally, retry the task (generally double-click/download a QWC or load an application operation). **End user:** Follow the instruction in the message (from QuickBooks) in the error screen. Most common causes are that QuickBooks is not running, and you need to start it, or make sure a company file is open and no modal dialog box is open. Otherwise, a message from QuickBooks is displayed. And finally, retry the task (generally double-click/download a QWC or load an application operation).
QBWC1025	Exiting the application. <An error message from QuickBooks appears here>.	The error occurred when trying to search for FileID (while adding this application to registry) for this application. A connection from QB web connector to QuickBooks was successful. However, an exception was thrown during the FileID find operation. **Developer:** Have your end user enable logging and send you the QWCLog.txt file. It should contain an exception message. **End user:** Please send the same information requested above for error QBWC1000.
QBWC1026	There was some problem adding fileID.	When trying to register FileID (while adding this application to registry) for a new application, QuickBooks did not return statusCode as 0 which indicates a possible problem during creation of the FileID. **Developer:** It may be because the FileID is already in use. Try using a different FileID value in your applications QWC file. **End user:** Please send the same information requested above for error QBWC1000.

Error Code	Error Message	More Information
QBWC1027	Exiting the application. <An error message from QuickBooks>.	When trying to register FileID (while adding this application to registry) for a new application, an exception was caught. **Developer:** Have your end user enable logging and send you the QWCLog.txt file. It should contain an exception message. **End user:** Please send the same information requested above for error QBWC1000.
QBWC1028	Exception encrypting. <A message describing the cause of error>.	There was a problem encrypting the password entered by user. The error message should show useful information. **Developer:** The error message should show useful information. Have your end user enable logging and send you the QWCLog.txt file. It should contain an exception message. **End user:** Please send the same information requested above for error QBWC1000.
QBWC1029	Exception decrypting. <A message describing the cause of error>	There was a problem decrypting the password. **Developer:** The error message should show useful information. Have your end user enable logging and send you the QWCLog.txt file. It should contain an exception message. **End user:** Please send the same information requested above for error QBWC1000.
QBWC1030	Password is not available for application named <AppName>. Please set the password for this application.	User forgot to set password before requesting an update. **Developer:** Have your user set the password in QB web connector for this application. **End user:** Set the password in QB web connector for this application. If you don't know your password, please contact your application provider the support url available at the QB web connector UI or QWC file.
QBWC1031	Operation completed with some error. Application has been notified of the error accordingly. See local log for further information.	This error indicates that there was an error during scheduled update operation. **Developer:** Have your user send you the QWCLog.txt file. It should contain an exception message. **End user:** Please send the same information requested above for error QBWC1000.

Error Code	Error Message	More Information
QBWC1032	Could not find application <AppName> in registry to complete scheduled update.	This error indicates that QB web connector was unable to find the application in registry during scheduled update operation. **Developer:** It may be possible that the application has been removed or deleted from windows registry. Have your user uncheck Auto-run, exit and re-start QB web connector so that it reloads the application again. User may need to reload the application by loading QWC file. **End user:** Uncheck Auto-run, exit and re-start QB web connector so that it reloads the application again. If the problem persists restore the application by reloading the application's QWC file.
QBWC1033	QB web connector failed to initialize QWCLog.txt file and will not run. Please make sure QWCLog.txt file is writable and then try again.	QB web connector was unable to use the QWCLog.txt file. The QWCLog.txt file may not have read-write permission set. **Developer:** Check the properties of QWCLog.txt file (generally in C:\Documents and Settings\All Users\Application Data\Intuit\Quickbooks Web Connector\version directory). Possible causes could be that the directory for QWCLog.txt file does not exist, file attributes are set to ReadOnly, QWCLog.txt is a file and not a directory, or user's hard disk file system is full. **End user:** Check the properties of QWCLog.txt file (generally in C:\Documents and Settings\All Users\Application Data\Intuit\Quickbooks Web Connector\version directory). Possible causes could be that the directory for QWCLog.txt file does not exist, File attributes are set to ReadOnly, QWCLog.txt is a file and not a directory, or your hard disk file system is full.
QBWC1034	Error setting AuthFlags - <An error from QuickBooks appears here>.	This error indicates that there was an error during the attempt to set AuthFlags for the QBXMLRP2 Request Processor. Possible problem in QBXMLRP2 installation. **Developer:** Examine the QWCLog.txt for a root cause. Have your end user enable logging and send you the QWCLog.txt file. It should contain an exception message. If none available, try re-installing QBXMLRP2 on end user's system. **End user:** Please send the same information requested above for error QBWC1000.

Error Code	Error Message	More Information
QBWC1035	Dns.Resolve(localhost) failed due to Exception -- <An eror message describing the possible root cause of the failure>.	Either Dns.Resolve() on localhost is not working. Possibly, a "ping localhost" will fail. Or, the QB Web Connector system failed to process Dns.Resolve() call. **Developer:** Try doing a host file entry to %WINDIR%\system32\drivers\etc\hosts: 127.0.0.1 localhost localhost.localdomain.com localhost.home.localdomain.com If the possible root cause indicate a failure to process Dns.Resolve() call, you may need to contact IDN Developer Support. You should never send a "localhost" qwc file out to your end user. This problem should come up only during development. **End user:** Your application provider most likely provided the wrong QWC file. In the QWC file, the AppURL or AppSupport with localhost is intended for only development purposes. Your QWC file should contain a qualified web address for AppURL and AppSupport. Contact your application provider to get the correct QWC file.
QBWC1036	Error countered during version check: <An error message describing possible root cause of the failure>.	There was a problem while WebMethod clientVersion() was being processed. It could be a possible SOAP problem. Stack trace in QWCLog.txt file should reveal further information. **Developer:** There was a problem while WebMethod clientVersion() was being processed. It could be a possible SOAP problem. Stack trace in QWCLog.txt file should reveal some clue. have your end user send you the QWCLog.txt file. It should contain an exception message. This is an exception that was caught during processing of the clientVersion() call. It could be a SOAP server or client (QB web connector) problem. This error has nothing to do with the clientVersion() warning or error your application is trying to send via this method. **End user:** Please send the same information requested above for error QBWC1000.

Error Code	Error Message	More Information
QBWC1037	Application sent following error or warning message when checking version of QB web connector. Update aborted. <An error or warning message from web application appears here>	This is an error that web application sent to QB web connector when processing the QB. Generally, "E:<any text>" instructs QB web connector to abort this data processing and force user to download a new version of QB web connector. "W:<any text>" instructs QB web connector to give user a choice to continue this update or not. **Developer:** Explain to your end user why your application sent this message with E: or W: and then have them upgrade QB web connector, if needed. **End user:** Follow the instruction in the message box -- generally shows some instruction from the web application. Most of the cases, application is asking you to upgrade your QB web connector. If confused about the message, take a screen shot of the message box and contact your application provider.
QBWC1038	User cancelled from master key input. Need master key to continue. If you forgot master key, you would need to re-enter password again to reset the master key.	User was prompted for master key to decrypt the password. At this point, user either entered an invalid master key or cancelled out from master key input. **Developer:** Have your end user enter the correct master key. If they forgot the master key, they would need to reset the master key by resetting their password. No need to repeat this master key input for all other web application passwords. They will be automatically encrypted using the new master key. **End user:** Enter the correct master key. If you forgot the master key, you would need to reset the master key by resetting your password. No need to repeat this master key input for all other web application passwords. They will be automatically encrypted using the new master key.
QBWC1039	There was a problem adding the application to registry. Check QWCLog.txt for details.	The QWC file may be missing some required elements. **Developer:** Run the QWC file through the validator. It may have missing elements. Also, the QWCLog.txt file should contain some information. **End user:** Please send the same information requested above for error QBWC1000.
QBWC1040	Web connector did not provide a valid username and/or password.	The application is not set with a password or set with an incorrect password **Developer:** Make sure your user knows the correct password to type in the Web Connector UI. **End user:** You need to obtain the correct password from your application provider and then type it in password text box and hit enter to set the password.

Error Code	Error Message	More Information
QBWC1041	SendRequestXML failed due to error message: <An error message appears here>.	An exception was caught during the sendRequestXML() call to application. This could be either a client or a server side problem. **Developer:** A call to the WebMethod sendRequestXML() failed with an exception. Have your end user send you the QWCLog.txt file just after the error occurred. It should contain a stack trace for the exception caught. It's very possible the error originated on your server, please check your server logs for any web service exceptions. **End user:** Please send the same information requested above for error QBWC1000.
QBWC1042	ReceiveResponseXML failed due to error message: <An error message appears here>.	An exception was caught during the receiveResponseXML() call to application. This could be either a client or a server side problem. **Developer:** A call to the WebMethod receiveResponseXML() failed with an exception. Have your end user send you the QWCLog.txt file just after the error occurred. It should contain a stack trace for the exception caught. It's very possible the error originated on your server, please check your server logs for any web service exceptions. **End user:** Please send the same information requested above for error QBWC1000.
QBWC1043	getLastError failed due to error message: <An error message appears here>.	An exception was caught during the getLastError() call to application. This could be either a client or a server side problem. **Developer:** A call to the WebMethod getLastError() failed with an exception. Have your end user send you the QWCLog.txt file just after the error occurred. It should contain a stack trace for the exception caught. It's very possible the error originated on your server, please check your server logs for any web service exceptions. **End user:** Please send the same information requested above for error QBWC1000.

Error Code	Error Message	More Information
QBWC1044	CloseConnection failed due to error message: <An error message appears here>.	An exception was caught during the closeConnection() call to application. This could be either a client or a server side problem. **Developer:** A call to the WebMethod closeConnection() failed with an exception. Have your end user send you the QWCLog.txt file just after the error occurred. It should contain a stack trace for the exception caught. It's very possible the error originated on your server, please check your server logs for any web service exceptions. **End user:** Please send the same information requested above for error QBWC1000.
QBWC1045	ConnectionError failed due to error message: <An error message appears here>.	An exception was caught during the connectionError() call to application. This could be either a client or a server side problem. **Developer:** A call to the WebMethod connectionError() failed with an exception. Have your end user send you the QWCLog.txt file just after the error occurred. It should contain a stack trace for the exception caught. It's very possible the error originated on your server, please check your server logs for any web service exceptions. **End user:** Please send the same information requested above for error QBWC1000.
QBWC1046	Application sent incorrect syntax return value for clientVersion(). Error message: < An error message describing possible root cause of the failure>. Update cannot continue.	There was a problem while WebMethod clientVersion() was being processed. Most likely problem is that the syntax of the return value from application was incorrect. **Developer:** QWCLog.txt should contain an exception message. This is an exception that was caught during processing of the returned value for clientVersion() call. **End user:** Please send the same information requested above for error QBWC1000.
QBWC1048	QuickBooks Web Connector could not verify the web application server certificate.	A web service is using a cert URL which requires authentication. One solution could be removing authentication from your cert URL.