Intuit QuickBooks® SDK

Technical Overview

Version 16.0

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/.

CONTENTS

Chapter 1: Introduction

Chapter 2: Communicating with QuickBooks

CHAPTER 1
INTRODUCTION

This overview describes the Intuit QuickBooks Software Development Kit (SDK). It is written for both prospective and new users of the QuickBooks SDK, including application developers, system integrators, software product managers, consultants, and other technical professionals interested in developing applications that share data with QuickBooks.

The purpose of this document is to provide you with a general understanding of the high-level concepts relating to the components of the QuickBooks SDK and how to use them. It also provides a roadmap to the SDK documentation to guide you through your initial work as you learn about the QuickBooks SDK.

Why an SDK for QuickBooks?

QuickBooks is Intuit's financial management system for small and medium-sized businesses. With QuickBooks, data such as accounts receivable, accounts payable, customer lists, vendor lists, employee lists, and expense and time tracking can be managed easily and safely. Because of its flexibility and power, QuickBooks is used for a wide variety of businesses—from construction firms to dental offices, from florists to property leasing agencies.

Often, additional applications are used to supplement QuickBooks. Sometimes, a complementary application is used to manage data in ways that are unique to a particular industry. In other cases, the adjunct application enters and stores the QuickBooks data differently, for example, on a per-client or per-location basis. In these examples, large quantities of important data need to be entered twice, resulting in extra work and potential errors, both of which cost time and money.

Sharing Data with QuickBooks

With the QuickBooks SDK, the small-business owner no longer needs to enter the same data twice. Because your application can share and modify QuickBooks data, the small-business owner has the best of both worlds—the power, ease, and comprehensiveness of QuickBooks combined with the benefits of an application tailored to the unique needs of a particular small-business concern.

The QuickBooks SDK allows your application to integrate directly with any of the following QuickBooks products (Release 2002 and beyond):

- QuickBooks Pro
- QuickBooks Premier (and industry-specific editions)
- QuickBooks Enterprise Edition (and industry-specific editions)
- QuickBooks Simple Start Edition (beginning with QuickBooks 2006 and SDK 5.0)

The SDK supports *U.S.*, *Canadian,* and UK *editions* of QuickBooks products. For convenience, this document uses the shortened name "QuickBooks" to refer to these products, because they all use the same APIs to communicate with QuickBooks. (The SDK does not support QuickBooks Basic.) Also see the IDG website for detailed information on supported QuickBooks products (*http://developer.intuit.com*).

Support for Payments

You can also save Payments credit card transaction data into QuickBooks starting with qbXML specification 4.1 and QuickBooks versions 2005 R5 or greater (and QuickBooks Enterprise Solutions 5.0 R4 and greater).

However, because QuickBooks 2007 supports the latest payment application best practices (PABP) requirements such as masking credit card numbers, we encourage the use of QuickBooks 2007 and later, and Intuit Payments API.

Support for Web Services

Web services can use the QuickBooks Web Connector to communicate with Quickbooks over the Internet. For more information, see the *QuickBooks Web Connector Programmer's Guide*.

A Common SDK Foundation

The SDK provides a common methodology for integrating an application with QuickBooks: once you've developed an application for one QuickBooks product, modifying your application to support any (or all) of the other QuickBooks products is a straightforward task. This shared SDK approach is based on qbXML, a version of XML (eXtensible Markup Language) designed specifically for QuickBooks.

Support for Multiple Programming Languages

The QuickBooks SDK is designed for use by a wide variety of developers in many different development environments. Its application programming interfaces (APIs) can be used by any programming language that is compatible with Microsoft's Component Object Model (COM). For web services designed to work with the QuickBooks Web Connector, compatibility with COM is not required because HTTPS is used for communication.

Design Principles

The QuickBooks SDK was designed with the following principles in mind:

- *Keep the small-business owner in control.*
  The small-business owner (or the administrator for the business) authorizes the connection between your application and QuickBooks. The small-business owner also sets up permissions

for authorized users and access rights for each third-party application.

- *Provide robust mechanisms to protect data.*
  The QuickBooks SDK provides strong error recovery, data logging, and synchronization facilities to ensure that data is not lost or destroyed and that application data remains synchronized with QuickBooks data.
- *Use existing programming standards.*
  The QuickBooks SDK uses standard COM interfaces and XML formats. As a result, it is compatible with most programming languages.
- *Adhere to a single specification for the entire SDK.*
  Except where noted, the QuickBooks products use the same qbXML specification. The content and behavior of the request and response messages is consistent across products, which simplifies your task if you decide to port your application from one QuickBooks product to another.

Runtime Components of the QuickBooks SDK

As shown in Figure 1, you can choose between one of two APIs when you create an integrated application with the QuickBooks SDK:

- The qbXML Request Processor, which requires you to create and parse documents written in qbXML. qbXML is a form of XML that deals with QuickBooks data.
- The QuickBooks Foundation Class (QBFC) Library, a library of COM objects that implement the qbXML specification. This API eliminates the need to create and parse qbXML directly in your code.

Regardless of which approach you take, you'll need a basic understanding of the elements, aggregates, and attributes defined in the qbXML specification.

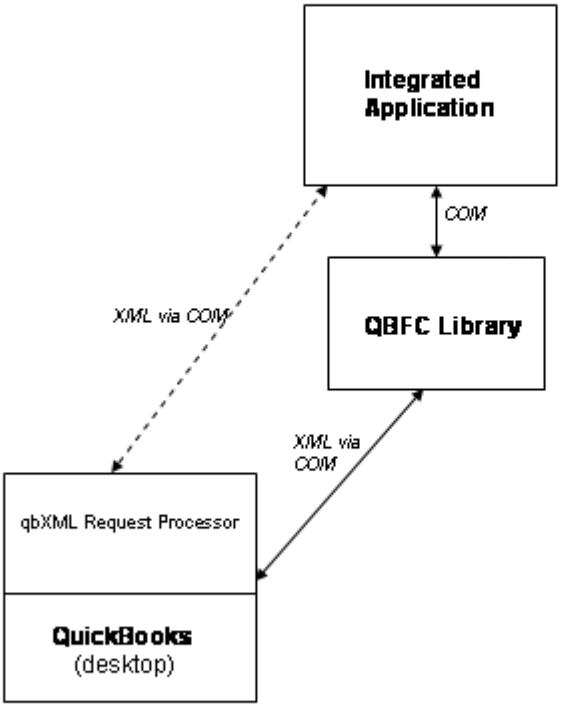

Figure 1 Components of the QuickBooks SDK

Figure 2 shows how web services applications access QuickBooks via the QuickBooks Web Connector.

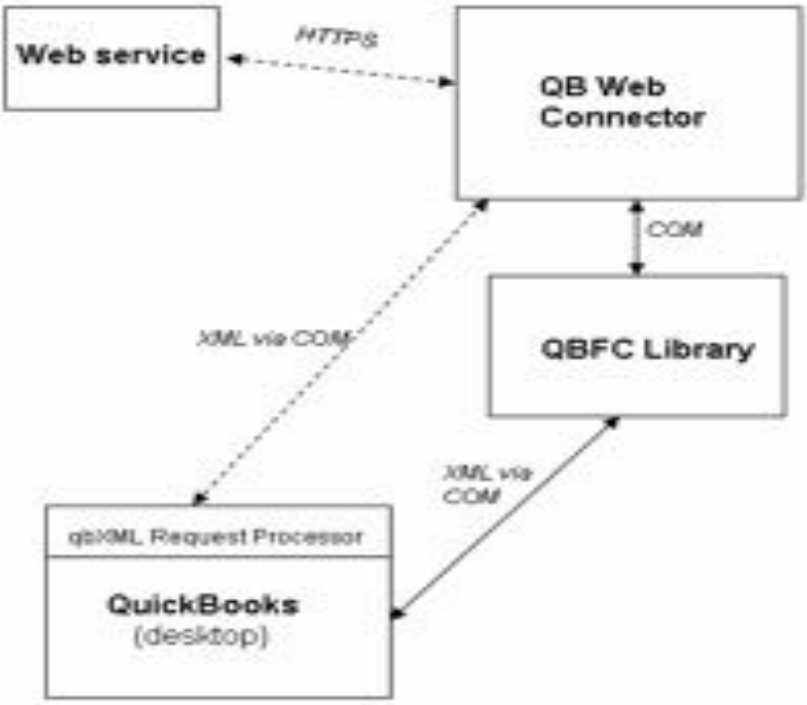

Figure 2 Accessing QuickBooks via web services

What's Included in the QuickBooks SDK?

Conceptually, the QuickBooks SDK includes the following software libraries, manuals, utilities, and examples. Some, but not all, of these items are provided in the QuickBooks SDK package. Others, such as the qbXML Request Processor interface, are actually runtime components of QuickBooks itself.

- **Software libraries.** APIs for creating, sending, and receiving QuickBooks messages. These libraries, shown in Figure 1 (page 9), include

- qbXML Request Processor Interface
- QuickBooks Foundation Class (QBFC) Library

- **qbXML specification.** The qbXML specification is described in qbXML schema files that are distributed with QuickBooks. QuickBooks validates your application's requests against the qbXML specification.
- **QuickBooks Web Connector(QBWC).** The QuickBooks Web Connector is a redistributable component that polls remote web services that have been registered in QBWC and issues requests on behalf of the web services to the target QuickBooks company. QBWC returns QuickBooks responses to the web services.
- **Example qbXML files.** This file (*qbxmlops*.xml*) includes examples of all qbXML request and response messages.
- **Documentation.** In addition to this *Technical Overview*, the following QuickBooks SDK documentation is available:
  - *QuickBooks Desktop API Reference*
  - *QuickBooks SDK Release Notes*
  - *QuickBooks SDK Programmer's Guide*
  - *QuickBooks Web Connector Programmer's Guide*
- **Utilities.** The SDK includes several utilities to aid in your development cycle. To verify that a given qbXML document conforms to the qbXML specification before it is sent to QuickBooks, use the *qbXML Validator* utility.

  To test the request/response cycle, use the *SDKTest* utility, which accepts a qbXML request, sends it to QuickBooks, and returns the response. Source code for *SDKTest* is available in multiple languages for the desktop version of QuickBooks.
- **Sample applications.** The SDK includes a large set of sample application programs, including both source code and executable files in Visual Basic, C, C++, and Java.

Which API to Use

An important initial decision you'll need to make is whether to use the qbXML Request Processor API or the QBFC API. The main difference between the two approaches is that the qbXML Request Processor API requires you to create and parse qbXML documents explicitly. With the QBFC interface, you are relieved of the task of parsing the qbXML content because you specify the data in terms of parameter/value pairs within COM methods.

Appendix A includes sample code that illustrates parallel use of both APIs. The examples, written in Visual Basic, illustrate a typical QuickBooks task: adding an invoice.

Supported Versions and Products

Refer to the QuickBooks SDK Release Notes for a comprehensive list of supported versions and products.

Notice that your application can also run against earlier versions of QuickBooks (QB 2002 and newer), but you must specify the qbXML version that those QuickBooks support, so functionality will be limited to that of the earlier specifications.

The SDK provides methods for querying which version of the qbXML specification is supported by the version of QuickBooks that is currently running on the user's system. (The QBXMLVersionsForSession) method can be called after the session begins.) If you write "smart" code that checks the version and responds accordingly, your application can run against multiple versions of QuickBooks.

FAQs about Versioning

The following paragraphs answer some frequently asked questions and provide further detail on versioning. Also see the QB SDK *Programmer's Guide*, which includes diagrams and sample code related to this topic.

Q: I am writing an application for QuickBooks 2022. Will it work with older versions, for example, QuickBooks 2019, 2018, 2017, 2016, 2015, 2014, 2013 or 2012?

A: Yes, if your application does the following:

1. Check which version of the qbXML specification is supported by the version of QuickBooks that is currently running.
2. In the qbXML prolog to each request message set, specify this version of the qbXML specification. (Your code will need to make this decision at runtime, after it checks the version.)
3. In certain cases, your code will need to branch and deal intelligently with differences between different versions of the qbXML specification. For example, certain requests include additional data for later versions of the specification. Your application can use the data when it is running against the later version of QuickBooks but won't have access to the data when it is running against an earlier version of QuickBooks.

   The prolog of the qbXML messages must reference the version of the specification they "understand," as listed in the SDK release notes. Because of this, new features associated with later versions cannot be invoked when your application is running against an earlier version of QuickBooks. If your code anticipates and deals with these version differences, it can run against earlier versions of QuickBooks.

Q: I wrote an application for an earlier version of QuickBooks. Will it work with QuickBooks 2022?

A: As shown in the release notes, QuickBooks 2022 supports all of the previous versions of the qbXML specification, so any message sent to earlier QuickBooks versions that support the SDK will also work with QuickBooks 2022.

Q: I want to add some SDK 15.0 functionality to my application, which was written for QuickBooks 2003. How do I proceed?

A: First of all, be sure to use the **QBXMLRP2** request processor, not the old and obsolete QBXMLRP request processor.

Next, when you update your application, be sure to separate the new functionality in your code and invoke it only when you're running

against QuickBooks 2022 (or later). After a session has begun, check the qbXML version. You can keep the old qbXML version numbers

for older messages, but new 15.0 requests will require an 15.0 prolog. In areas where functionality has changed, your application

needs to include a branch for the original functionality (in your case, versions14.0,13.0,12.0,11.0,10.0,9.0,8.0,7.0,6.0,5.0,4.1,4.0,3.0,2.1,2.0 ,1.1, 1.0) and a separate branch taking advantage of new functionality (version 15.0). Be sure to consider version differences both when you're creating requests and when you're parsing responses.

Before You Begin

The SDK documentation assumes you have a sound understanding of the following:

- QuickBooks (This is critical: you must know what you want to do in QuickBooks and whether you can do it in the QuickBooks User Interface before you start designing your application.)
- Programming language of your choice
- Microsoft's Component Object Model (COM) (in most cases; see Figure 1)

Knowledge of XML is useful if you plan to use the qbXML Request Processor API. In this case, you may also want to install and use an XML parser such as Microsoft's XML Core Services (MSXML).

The Getting Started Video

Many developers plunge right into coding before getting a firm grasp on what needs to be done in the QB SDK programming environment. As a result, their initial progress is slower than it should be.

A great way to jumpstart your progress and get a good overview of programming in this environment is to watch the free video from IDG, available either streaming or in downloadable files. The video lasts about one hour, but is well worth the time. The QuickBooks SDK Essentials Video is available at the *IDG website*.

QuickBooks desktop API Reference

Once you're ready to program, you'll use the *QuickBooks Desktop API Reference* often to find out the exact syntax of a given request or response. This online reference provides detailed information for all developer libraries—qbXML and QBFC for QuickBooks (U.S), QuickBooks (Canada), and QuickBooks UK.

QuickBooks Desktop API Reference is the documentation for programming with the QB SDK. It includes descriptions of each object, aggregate, and element in the SDK. For each element, it includes the data type, maximum length (for strings) or range (for numerical values), whether it is required or optional, and whether it is restricted to a particular release.

The API Reference provides sample code for qbXML and also QBFC in C# and VB.Net.The request descriptions include links to in-depth information in the SDK Programmer's Guide. There is also an error code lookup feature that accepts codes either in decimal or hex.

https://developer.intuit.com/app/developer/qbdesktop/docs/api-reference/qbdesktop

Programmer's Guide

The QBSDK *Programmer's Guide* contains instructions and concepts you need to know to program with the QuickBooks SDK. It explains the details of message requests and responses, describes how to create queries and reports, and guides you through dealing with complex transactions such as receive payment and bill payment. It includes information event notification, integrating with the QuickBooks user interface, using BuildAssemblies, SalesReceipts, and credit card refunds. It also focuses on general application concerns such as error recovery, how to synchronize application data with QuickBooks, and how to anticipate typical user problems in your application.

What's Next?

When you're finished reading the rest of this *Technical Overview*, you'll probably want to consult these additional sources of information:

- QuickBooks Desktop API Reference
- The QBSDK Programmer's Guide
- The QuickBooks Web Connector Programmer's Guide
- Web site for the Intuit Developer Group (IDG): *http://developer.intuit.com*

CHAPTER 2
COMMUNICATING WITH QUICKBOOKS

Basic communication between your application and QuickBooks is based on a client/server model, as shown in Figure 3. The application sends a request message to QuickBooks, and QuickBooks sends back a response message. With this model, there is always a one-to-one correspondence between request and response messages, and the communication is synchronous. For efficiency, messages of the same type (that is, either request or response) are grouped into *message sets*.

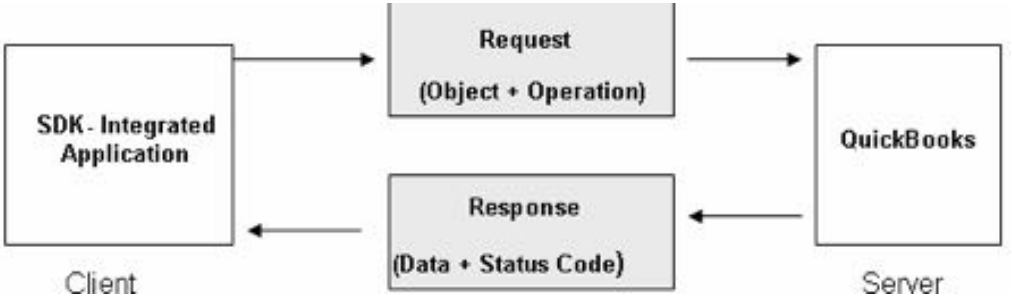

Figure 3 Client/server model for communication

Notice that for web services, the same model is used, but with a slight twist. The QB Web Connector becomes the integrated application in this case, handling the communication with QuickBooks on behalf of the web service (see Figure 4).

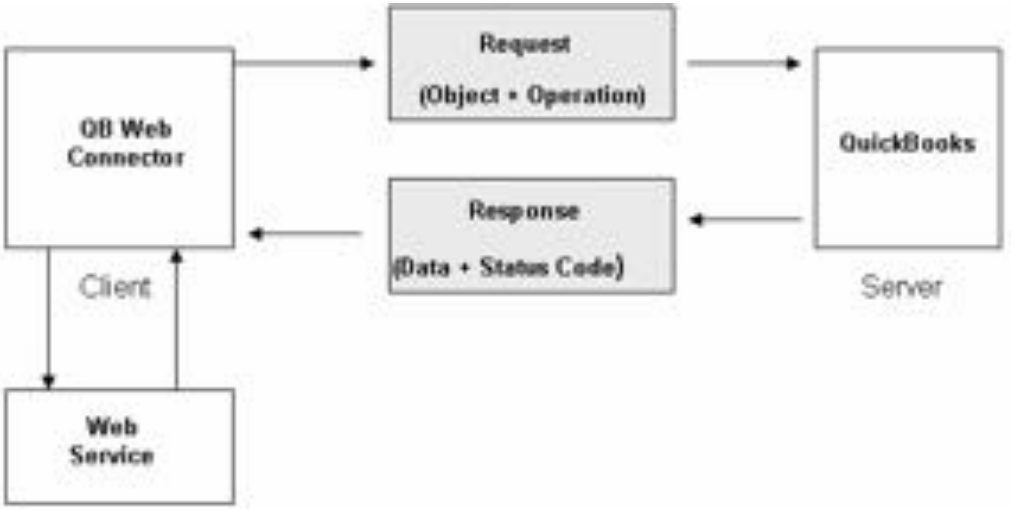

Figure 4 QuickBooks communication from a web service

For QuickBooks, there are several possible ways to set up your application:

- Your application runs as a separate process and resides on the same desktop system as QuickBooks. Optionally, your application can access QuickBooks remotely on a different machine inside the local network if Remote Data Sharing (RDS) is used.
- Your web service application runs on a remote server and accesses QuickBooks via the QuickBooks Web Connector.

Typical Sequence

In its simplest form, your application typically performs these tasks:

1. Establish a communications channel with QuickBooks
2. Build a set of request messages
3. Send the request message set to QuickBooks

4. Check the status codes and data in the response message set received from QuickBooks
5. Close the communications channel with QuickBooks

Setting Up the Communications Channel

In the list above, Steps 1, 3, and 5 deal with how an application communicates with QuickBooks. These steps can be different for each QuickBooks product. Desktop applications that integrate with QuickBooks, QuickBooks Canada, QuickBooks UK, or QuickBooks Simple Start, use either the qbXML Request Processor API or QBFC (both COM APIs) to accomplish this task. The following sections provide a few more details concerning these APIs.

What's in a Message?

Steps 2 and 4 require an understanding of the content of specific request and response message objects. With minor exceptions, these message components are the same across all QuickBooks products. The specifics of how to name, construct, and parse messages differ for the qbXML developer and the QBFC developer, but the general principles—as well as the specific content elements—are essentially the same for all developers using the QuickBooks SDK. These principles and specifics are covered in - of this document and, in further detail, in the *Programmer's Guide*.

Key Concepts

Keep in mind these general concepts as you look at the specific API methods listed in the following sections. First, a *connection* is the "handshake" made between QuickBooks and your application. Before anything else can happen, QuickBooks allows the small-business owner to authorize the connection between QuickBooks and your application.

QuickBooks

For QuickBooks, a *session* occurs within the context of a connection and is tied to a specific company data file. As shown in Figure 5 and Figure 6, a connection can contain a single session, which deals with one company data file, or multiple sessions dealing with multiple company data files. Since opening and closing connections and sessions is associated with a certain amount of overhead, you should perform these operations only when necessary.

Once a session has been successfully established, a session handle, called a *ticket*, is created and returned to you. This ticket is passed to QuickBooks along with each message request sent during the session. (With QBFC, this mechanism is handled internally.)

An application can open multiple company files, with only one company open at a time, under the following circumstances:

- QuickBooks can be running before the application starts, but it cannot have a company file open.
- The application must have permission to access all needed company files using automatic login.
- When the application is ready to close a company file and open a new one, no other applications on the same computer can have that same company file open.
- The application must end the session with the currently open company file before it begins a session with a different company file.

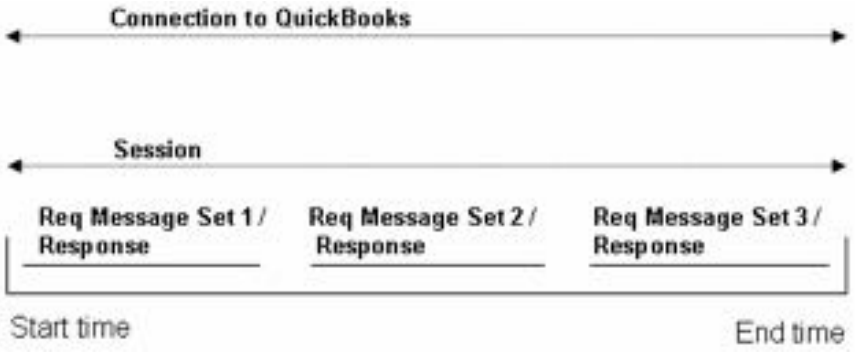

Figure 5 Single session within a connection

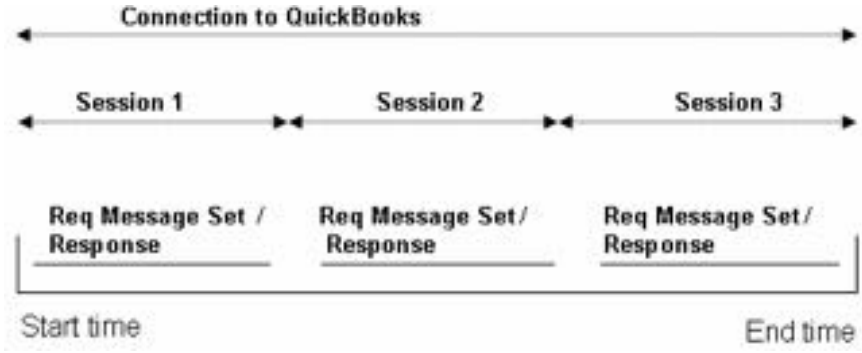

Figure 6 Multiple sessions (for multiple companies) within a connection

Communicating with Web Services

Remote web services can communicate with desktop editions of QuickBooks by implementing interfaces that enable communication with the QuickBooks Web Connector. The Web Connector is a redistributable component that runs on the same machine as the local QuickBooks.

The Web Connector is an SDK-integrated application that polls the web service at time intervals specified by the QuickBooks user. If the web service has work for the customer, it tells the Web Connector what it wants (SDK requests) and the Web Connector passes the request on to QuickBooks, returning any responses to the web service.

For complete details on this topic, including how to create the required interfaces in your web service, see the *QuickBooks Web Connector Programmer's Guide*.

Integration with the QuickBooks User Interface

Your application can add a menu with associated subitems to QuickBooks (see the *Programmer's Guide* for details), and can obtain context information about QuickBooks (which QuickBooks form was open) when your menu items were clicked. Your application can also display reports as well as some list elements and transactions inside QuickBooks, and can prefill transaction create forms with customer, vendor, employee, or other name data.

Receiving Events from QuickBooks

In addition to the request/response model, the SDK provides a supplementary event model that allows your application to receive notification of events that occur within QuickBooks. Your application subscribes to certain types of events in QuickBooks, and then QuickBooks sends an event message to your application

whenever one of these events occurs. Events your application can subscribe to are of three types:

- **Data events** - occur when the QuickBooks user or an integrated application adds, modifies, or deletes data
- **User interface events** - occur when the QuickBooks user interacts with the QuickBooks user interface (excluding menu items that have been added by developers); examples of user interface events are when the user opens or closes the company file
- **User interface extension events** - occur when the user selects a menu item added by your application

QuickBooks (U.S. and Canadian Editions)

Developers for QuickBooks integrated applications (both U.S., Canadian, and UK editions of QuickBooks) can use either the qbXML Request Processor API or the QBFC API.

qbXML Request Processor API

The qbXML Request Processor API provides the following COM methods for communicating with QuickBooks:

- *OpenConnection2*
- *BeginSession*
- *ProcessRequest*
- *EndSession*
- *CloseConnection*
- *Various methods supporting AuthPreferences (allows you to set or get authorization preferences)*
- *Various methods supporting AuthFlags (allows you to specify the QB editions that your application supports)*

The OpenConnection2 method provides flexibility through its connection type parameter, which can be used to cause various types

of connections, such as launching the QuickBooks UI from an SDK application.

QBFC API

In QBFC, which is also COM-based, the Session Manager (QBSessionManager) handles all communication with QuickBooks. The Session Manager provides these methods for communicating with QuickBooks:

- *OpenConnection2*
- *BeginSession*
- *DoRequests*
- *EndSession*
- *CloseConnection*
- *Various methods supporting AuthPreferences (allows you to set or get authorization preferences)*
- *Various methods supporting AuthFlags (allows you to specify the QB editions that your application supports)*

The Session Manager creates the message set request object with its *CreateMsgSetRequest* method. This message set is sent to QuickBooks using the Session Manager's *DoRequests* method (which is analogous to the *ProcessRequest* method in the qbXML Request Processor API).

Security Concerns

A number of mechanisms combine to ensure that your application remains tamperproof and that the small-business owner controls access to the company's QuickBooks files. QuickBooks provides a user interface for authentication and authorization that is invoked at certain points in the connection process on behalf of your application.

Authentication

For applications that integrate with QuickBooks, the best way for you to assure the small-business owner that your application has not been tampered with, and is not a rogue application masquerading as your application, is to obtain a digital code-signing certificate for the application. This certificate contains information such as your public key, your name, an expiration date, the name of the certificate authority that issued the certificate, and a serial number. A digital signature provides the end user with assurance that the code has originated with you, the software publisher, and that it has not been tampered with.

QuickBooks notifies the end user that an application is digitally signed when the application attempts to access QuickBooks. The Intuit Developer Group website *(http:// developer.intuit.com)* describes how to obtain a digital certificate using the Microsoft Authenticode™ technology.

Authorization

For QuickBooks (both U.S. and Canadian editions), a BeginSession call initiates the authorization process. The first time your application is used by a small-business owner, the user must have QuickBooks running and be logged on as the administrative user. The administrator authorizes further use of your application by other users and specifies permissions when the application starts QuickBooks in unattended mode. The QuickBooks SDK supports the user privilege structure as established by QuickBooks (wherein, for example, certain users can access only certain types of data).

Authorization for Web Services

A web service must be authorized by the QuickBooks administrative user just like other applications. In addition, the user can grant or remove the web service's permission to access the QuickBooks Web Connector itself.

Access to Personal Data

The QuickBooks administrator controls whether a given integrated application has access to personal data—for example, social security number—contained in the QuickBooks company file. Without this permission, certain personal data will not be returned to the application.

Access to Payroll Data

Certain payroll data is available only if your application is accessing a company file that is currently subscribed to a payroll service. To access this data, the following two conditions must be met:

- The administrative user must have granted the application permission to view personal data, as described in the previous section, and
- The application must be accessing a company file that is currently signed up for a subscription to a payroll service

See the *Programmer's Guide* for more information.

Error Handling

Errors are returned on two general levels:

- Result codes related to the establishment of the connection and session and the general communications process between QuickBooks and the application
- Status information (codes, severity levels, and messages) contained within individual message responses sent by QuickBooks

Your application needs to inspect and respond to both types of error codes and respond appropriately.

Synchronization

One of the most important features of the SDK is that it allows you to ensure that the data in your application is always synchronized with QuickBooks data. Whenever you send a request to QuickBooks to modify data, QuickBooks checks whether you have the most up-to-date version of the data before granting your request. If your data is not current, an error code is returned and the data is not modified. You can then resolve the problem and resend the modify request if necessary. In many cases, your application may need to query the end user before taking further steps.

Synchronization problems can be complex, and their resolution is usually dependent on the individual application. The *Programmer's Guide* provides further information on this topic.

CHAPTER 3
REQUEST AND RESPONSE MESSAGES

This chapter provides introductory material on the types of data exchanged between your application and QuickBooks and the operations you can request QuickBooks to perform on the data. The QuickBooks SDK can currently perform most, but not all, operations on most kinds of QuickBooks data. Table 1 at the end of this chapter lists objects and operations currently supported.

Currently, an application can interact with QuickBooks in one of two ways. Most importantly, it can send requests to QuickBooks and initiate data responses from QuickBooks. Second, your application can effect various changes in QuickBooks user interface behavior, such as causing a report to be displayed in QuickBooks or displaying a QuickBooks form to edit or create list elements or transactions. (You can even prefill the transaction creation forms with data from existing customers, vendors, employee, or other name entities.)

"Objects" in the SDK

In the QuickBooks SDK documentation, the term *object* is used to refer to a transaction or list item that is commonly found in QuickBooks, such as an account, payment, credit, vendor, invoice, or purchase order. Objects fall into two general categories: *lists* and *transactions*. In addition, there are a few miscellaneous objects—namely, *company, host,* and *preferences*—that can be queried, as well as various *reports* that can be obtained from QuickBooks.

Lists

List objects correspond to lists of information in QuickBooks. Lists are identified by a list ID or by a descriptive name (FullName). The following types of lists are supported:

- *Account Lists*
  Account
- *Entity Lists*
  Customer, Vendor, Employee, Other Names
- *Item Lists*
  Inventory, Non-inventory, Inventory Assembly (Premier Edition and above), Service, Group, Payment, Discount, Sales Tax, Sales Tax Group, Other Charge, Subtotal, Fixed Asset, Vehicle
- *Simple Lists*
  Sales Tax Code (U.S. Edition), Class, Standard Terms, Date Driven Terms, Ship Method, Payment Method, Sales Rep, Job Type, Customer Type, Vendor Type, Customer Message, Template, To Do, Currency, PriceLevel, BillingRate, UnitOfMeasureSet
- *Payroll Lists*
  Wage Payroll, NonWage Payroll, WorkersCompCode

Transactions

Transaction objects correspond to the basic accounting entities in a business. Adding a transaction object (except a non-posting transaction object) affects the account balance of the business. Transactions are identified by a transaction ID (TxnID). Beginning with SDK 3.0, your application can also query for templates and use them when adding or modifying transactions in QuickBooks. The following types of transactions are supported:

- *Accounts Receivable Transactions:*
  Invoice, Credit Memo, Receive Payment, Charge

- *Accounts Payable Transactions:*
  Vendor Credit, Bill, Bill Payment Check, Bill Payment Credit Card, Item Receipt, Sales Tax Payment
- *Non-Posting Transactions:*
  Purchase Order, Estimate, Sales Order (US Premier Edition and above)
- *Bank Transactions and Sales Receipts:*
  Check, Deposit, Sales Receipt
- *Credit Card Transactions:*
  Credit Card Charge, Credit Card Credit, Credit Card Refund
- *Other Transactions:*
  Journal Entry
- Time Tracking
- Vehicle Mileage
- Inventory Adjustment
- BuildAssembly

Operations

The SDK supports the following operations:

- *Query* - obtains information about one or more objects according to specified criteria
- *Add* - adds an object to QuickBooks
- *Modify* - modifies an existing QuickBooks object
- *ListMerge* - merges two accounts, or two vendors, or two customers/jobs, or two classes.
- *Delete* - removes the list or transaction object from QuickBooks
- *Void* - changes the transaction amount to zero but leaves a record of the transaction in QuickBooks (does not apply to list items)

Naming Conventions for Messages

Names of request and response messages are concatenations of an object name, an operation, and an abbreviation that indicates whether it is a request (Rq) or a response (Rs), as described in the following sections.

Request Messages

The naming conventions for messages are slightly different between APIs, but the basic name for a request is created as follows:

object + operation + **Rq**

Examples of request messages are

InvoiceAddRq
CustomerModRq
CheckAddRq
EstimateQueryRq

The QBFC API follows the same naming convention, but it inserts an "I" (for *Interface*) before the message name. For example, IInvoiceAddRq, ICustomerModRq, and so on.

Void requests and Delete requests are slightly different from Add, Modify, and Query requests. There is only one form for a list delete request (*ListDelRq*), which is used for all list types. Similarly, there is only one form for a transaction delete request (*TxnDelRq*) and one form of transaction void request (*TxnVoidRq*).

To see how requests and responses match up and what information they contain, look at the *QuickBooks Desktop API Reference.*

Response Messages

The naming conventions for response messages is similar to that for requests:

object + operation + **Rs**

Examples of response messages are

InvoiceAddRs
CustomerModRs
CheckAddRs
EstimateQueryRs

Responses contain slightly different data from their corresponding requests, so you'll always need to look up their exact syntax in the *QuickBooks Desktop API Reference* that corresponds to your API and product.

Return Status

In addition to accounting data that may be part of a response, the response message sent by QuickBooks contains status information regarding the processing of the request. Three types of status information are returned in each response:

- Status code - indicates, in numerical form, what happened to the request.
- Status severity - indicates whether request processing succeeded or failed. This indicator can be *Info, Warning,* or *Error.* If an Error, the request was not successfully processed and the response contains no data.
- Status message - explains the error or warning condition that is indicated by the status code.

Each request message set contains an onError attribute that specifies whether to continue processing requests contained in the same message set if an error is encountered.

Elements and Aggregates

The basic building block of any message is an element. An *element* is a name/value pair. Each element has an associated data type. In qbXML, an element corresponds to an XML field. In QBFC, an element corresponds to a COM property.

Elements are grouped together into aggregates. An *aggregate* is simply a container object that holds elements and other aggregates.

Each element and aggregate may be required or optional, according to the qbXML specification. Also, each element and aggregate may or may not be repeatable, according to the specification. The *QuickBooks Desktop API Reference* contains the details for the elements and aggregates contained in each qbXML message.

Object References

Each list object has a corresponding *object reference*, which is a pointer to the actual object. Object references are used to refer to objects of a particular type—for example, VendorRef, DepositToAccountRef, PayeeEntityRef. Both list objects and transactions objects can contain object references.

qbXML Sample Request and Response

In the following sample request, as in all requests, the request message set is named QBXMLMsgsRq. The onError attribute indicates to stop processing if an error is encountered. The request message is named CustomerAddRq. An example of an aggregate is CustomerAdd (the "*message aggregate*," since it is the container for the message data). The other parts of the request are all elements (Name, FirstName, LastName, Phone). (This request message has two required elements: the message aggregate, CustomerAdd, and the Name element. It also includes three optional elements: FirstName, LastName, and Phone.)

```
<?xml version="1.0" ?>
<?qbxml version="6.0"?>
<QBXML>
  <QBXMLMsgsRq onError="StopOnError">
    <CustomerAddRq requestID = "1">
      <CustomerAdd>
        <Name>Sally Smith</Name>
        <FirstName>Sally</FirstName>
        <LastName>Smith</LastName>
        <Phone>123-2345</Phone>
      </CustomerAdd>
```

```
    </CustomerAddRq>
  </QBXMLMsgsRq>
</QBXML>
```

Here is the corresponding response to this request:

```
<?xml version="1.0" ?>
<QBXML>
<QBXMLMsgsRs>
  <CustomerAddRs requestID="1" statusCode="0"
        statusSeverity="Info"
        statusMessage="Status OK">
    <CustomerRet>
      <ListID>30000-1029522127</ListID>
      <TimeCreated>2006-08-16T11:22:07-08:00
      </TimeCreated>
      <TimeModified>2006-08-16T11:22:07-08:00
      </TimeModified>
      <EditSequence>1029522127</EditSequence>
      <Name>Sally Smith</Name>
      <FullName>Sally Smith</FullName>
      <IsActive>true</IsActive>
      <Sublevel>0</Sublevel>
      <FirstName>Sally</FirstName>
      <LastName>Smith</LastName>
      <Phone>123-2345</Phone>
      <Balance>0.00</Balance>
      <TotalBalance>0.00</TotalBalance>
      <JobStatus>None</JobStatus>
    </CustomerRet>
  </CustomerAddRs>
</QBXMLMsgsRs>
</QBXML>
```

Queries and Filters

You can perform a query on most QuickBooks objects. A number of *filters* enable you to specify characteristics and parameters for the queried objects. For companies with large amounts of data, accurate

filtering of queries is essential so that the returned data is of a manageable size.

For example, list filters allow you to specify the actual list IDs, a range of dates, or an alphabetical range of names, as well as a maximum number of objects to return. The following qbXML example shows a customer query request that asks for all customer objects that are currently enabled for use by QuickBooks and that also have been modified (or created) on or between the beginning of day on October 12, 2003, and end of day on October 15, 2003:

```
<CustomerQueryRq requestID"541">
  <ActiveStatus>All</ActiveStatus>
  <FromModifiedDate>2003-10-12</FromModifiedDate>
  <ToModifiedDate>2003-10-15</ToModifiedDate>
</CustomerQueryRq>
```

Transaction queries can also include filters for specific IDs, for date ranges, or for particular entities, among others, as well as a maximum number of objects to return. The following qbXML example shows an invoice query request that asks QuickBooks to return all invoice objects modified (or created) between January 2, 2003, and January 5, 2003. These invoice objects must all apply to the Jones kitchen job (for example, they could be for work completed or for goods purchased).

```
<InvoiceQueryRq request ID=”73”>
  <ModifiedDateRangeFilter>
    <FromModifiedDate>2003-01-02</FromModifiedDate>
    <ToModifiedDate>2003-01-05</ToModifiedDate>
  </ModifiedDateRangeFilter>
  <EntityFilter>
    <FullName>Jones:Kitchen</FullName>
  </EntityFilter>
</InvoiceQueryRq>
```

Reports

Report queries, like object queries, are a useful way for your application to retrieve data from QuickBooks. The SDK supports

most of the report types found in the QuickBooks user interface. Much report data is based on changes over a period of time so that clients can compare results for a wide range of periods. Reports also span multiple data types, so it is possible to obtain a consolidated view in a single request. You would request a report, for example, to learn about all transactions that have been posted to an account in the past fiscal year. An account query, in contrast, only returns information about an account, including the current balance.

Categories of Reports

For convenience, the SDK groups QuickBooks reports into a number of logical categories based on shared general characteristics among the reports. Categories for reports are as follows:

- Aging
- Budget
- Custom Detail
- Custom Summary
- General Detail
- General Summary
- Job
- Payroll Detail
- Payroll Summary
- Time

The Standard Profit and Loss Report, for example, is in the General Summary category. All reports in this category have columns that are period-based (for example, every two weeks, last fiscal year). General Detail reports consist exclusively of transaction detail reports such as Profit and Loss Detail, Income by Customer Detail, Estimate by Job, and so on.

Report data is described as a table of rows and columns. In addition to the actual data contained in the report, the response contains meta-data regarding the report as a whole as well as the individual pieces of the report (for example, report title, number of rows and columns, whether the report is cash- or accrual-based, and the type of each column and row: data, text, total, or subtotal).

Customization

You have the option of customizing the requested report in a number of ways. If you request a report without any customization, the report uses the default parameters employed by the QuickBooks user interface. Depending on the report type, you can customize some or many of the following features:

- Date, based on date range macro or custom date
- Columns to include, for transaction reports
- Period data (previous period, current period, etc.) for period reports
- Fiscal, calendar, or tax year for summary reports
- Active, non-active, or all accounts for summary reports
- Account types or specific accounts to filter by
- Item types or specific items to filter by
- Named entity types or specific entities to filter by
- Transaction types to filter by
- Class to filter by
- Report basis
- Detail level to filter by
- Posting status to filter by

Messages Supported by the SDK for QuickBooks

The following table lists messages supported by this release of the QuickBooks SDK for US QuickBooks. See the *Programmer's Guide* for details on messages supported by QB Canada, QB United Kingdom, and Simple Start Edition.

Table 1 Messages Supported by the SDK for QuickBooks

Object	Query	Add	Modify	Delete	Void	Merge
Account	yes	yes	yes	yes	no	yes, via List-Merge
AgingReport	yes	no	no	no	no	NA
ARRefundCreditCard	yes	yes	no	yes	yes	NA
Bill	yes	yes	yes	yes	yes	NA
BillingRate	yes	yes	no	yes	no	NA
Bill payment check	yes	yes	yes	yes	yes	NA
Bill payment credit card	yes	yes	no	yes	yes	NA
Bill to pay	yes	no	no	no	no	NA
BudgetSummary-Report	yes	no	no	no	no	NA
BuildAssembly	yes	yes	yes	yes	yes	NA
Charge	yes	yes	yes	yes	yes	NA
Check	yes	yes	yes	yes	yes	NA
Class	yes	yes	yes	yes	no	yes
Company	yes	no	no	no	no	NA
CompanyActivity	yes	no	no	no	no	NA
Credit card charge	yes	yes	yes	yes	yes	NA
Credit card credit	yes	yes	yes	yes	yes	NA
Credit memo	yes	yes	yes	yes	yes	NA
Currency	yes	yes	yes	yes	no	NA
CustomDetailReport	yes	no	no	no	no	NA
Customer	yes	yes	yes	yes	no	yes, via List-Merge
Customer message	yes	yes	no	yes	no	NA
Customer type	yes	yes	no	yes	no	NA
CustomSummary-Report	yes	no	no	no	no	NA
DataEventRecovery-Info	yes	no	no	yes	no	NA
DataEventSubscrip-tion	yes	yes	no	yes	no	NA
DataExt	no	yes	yes	yes	no	NA

Object	Query	Add	Modify	Delete	Void	Merge
DataExtDef	yes	yes	yes	yes	no	NA
Date-driven terms	yes	yes	no	yes	no	NA
Deposit	yes	yes	yes	yes	yes	NA
Discount item	yes	yes	yes	yes	no	NA
Employee	yes	yes	yes	yes	no	NA
Entity	yes	no	no	no	no	NA
Estimate	yes	yes	yes	yes	no	NA
Fixed asset item	yes	yes	yes	yes	no	NA
GeneralDetailReport	yes	no	no	no	no	NA
GeneralSumaryReport	yes	no	no	no	no	NA
Group item	yes	yes	yes	yes	no	NA
Host application	yes	no	no	no	no	NA
Inventory adjustment	yes	yes	no	yes	no	NA
Inventory assembly item	yes	yes	yes	yes	no	NA
Inventory item	yes	yes	yes	yes	no	NA
Invoice	yes	yes	yes	yes	yes	NA
Item	yes	no	no	no	no	NA
Item Receipt	yes	yes	yes	yes	yes	NA
Job type	yes	yes	no	yes	no	NA
JobReport	yes	no	no	no	no	NA
Journal entry	yes	yes	yes	yes	yes	NA
ListDisplay	no	yes	yes	no	no	NA
Non-inventory item	yes	yes	yes	yes	no	NA
Non-wage payroll item (see Note at end of table)	yes	no	no	no	no	NA
Other charge item	yes	yes	yes	yes	no	NA
Other name	yes	yes	yes	yes	no	NA
Payment item	yes	yes	yes	yes	no	NA
Payment method	yes	yes	no	yes	no	NA
PayrollDetailReport (See Note below)	yes	no	no	no	no	NA
PayrollSummary Report (See Note below)	yes	no	no	no	no	NA

Object	Query	Add	Modify	Delete	Void	Merge
Preferences	yes	no	no	no	no	NA
PriceLevel	yes	yes	yes	yes	no	NA
Purchase order	yes	yes	yes	yes	no	NA
Receive payment	yes	yes	yes	yes	no	NA
Receive payment to deposit	yes	no	no	no	no	NA
Sales order (US Premier Edition and above)	yes	yes	yes	yes	no	NA
Sales receipt	yes	yes	yes	yes	yes	NA
Sales rep	yes	yes	yes	yes	no	NA
Sales tax code (US only)	yes	yes	yes	yes	no	NA
Sales tax group item	yes	yes	yes	yes	no	NA
Sales tax item	yes	yes	yes	yes	no	NA
Sales tax payment check	yes	no	no	no	no	NA
Service item	yes	yes	yes	yes	no	NA
Ship method	yes	yes	no	yes	no	NA
SpecialAccountAdd	yes	yes	yes	yes	no	NA
SpecialItemAdd	yes	yes	yes	yes	no	NA
Standard terms	yes	yes	no	yes	no	NA
Subtotal item	yes	yes	yes	yes	no	NA
Tax code (Canada only)	yes	yes	yes	yes	no	NA
Template	yes	no	no	no	no	NA
Terms	yes	no	no	no	no	NA
Time tracking	yes	yes	yes	yes	no	NA
TimeReport	yes	no	no	no	no	NA
To do	yes	yes	no	yes	no	NA
TransactionQuery	yes	no	no	no	no	NA
TxnDisplay	no	yes	yes	no	no	NA
UIEventSubscription	yes	yes	no	yes	no	NA
UIExtensionSubscrip-tion	yes	yes	no	yes	no	NA

Object	Query	Add	Modify	Delete	Void	Merge
UnitOfMeasureSet	yes	yes	no	yes	no	NA
Vehicle	yes	yes	yes	yes	no	NA
VehicleMileage	yes	yes	no	yes	no	NA
Vendor	yes	yes	yes	yes	no	yes, via List-Merge
Vendor credit	yes	yes	yes	yes	yes	NA
Vendor type	yes	yes	no	yes	no	NA
Wage payroll item	yes	yes	no	no	no	NA
WorkersCompCode*	yes	yes	yes	yes	no	NA

NOTE * Requires that the company file being accessed is currently subscribed to a payroll service.

GLOSSARY

aggregate
A container object that holds elements and/or other aggregates. It does not itself contain data. An aggregate can have one or more **attributes**.

API
Application programming interface.

attribute
A name-value pair that specifies additional data in a qbXML element or aggregate.

connection
A "handshake" between the application and QuickBooks.

data event
Notification, sent to an integrated application, that data within Quickbooks has been added, modified, or deleted. If your application has subscribed to data events, it will receive notification when these operations occur. (See Chapter 5 of the *Concepts Manual*.)

delete
Deleting a transaction or list removes it altogether.

digital certificate
A set of data that uniquely identifies a software publisher. It is issued by a certificate authority and contains information including a public key, name of the software publisher, an expiration date, the name of the authority that issued the certificate, and a serial number.

digitally signed application
An application that contains a digital certificate.

element
A name-value pair that is the basic building block of a message. An element has an associated data type.

event notification
Delivery of event information to an integrated application when a QuickBooks data event, user interface event, or user interface extension event occurs. See also *subscription*.

filter
A specification of selection criteria that allows you to characterize and delimit the data returned by QuickBooks in response to a query.

list
A collection of objects of the same type. A list in the SDK corresponds to a list in QuickBooks.

message
Information exchanged between the application and QuickBooks. A *request* message is sent by the application to QuickBooks. A *response* message is returned by QuickBooks to the application. There is a one-to-one correspondence between requests and responses.

message set
A collection of **messages** of the same type (either requests or responses).

object
An entity that is commonly found in QuickBooks, such as an account, payment, credit, vendor, invoice, or purchase order. Objects fall into two general categories: *lists* and *transactions*. In addition, there are a few miscellaneous objects that can be queried—namely, *company, host, preferences*, and *reports*.

operation
An action that is performed on an object. Operations are Add, Modify, Query, Delete, and Void. Not all operations can be performed on all objects (see Chapter 3).

QBFC
QuickBooks Foundation Class Library, a library of COM objects that implement the qbXML specification.

qbXML
A collection of XML elements, aggregates, and attributes designed for interacting with QuickBooks data.

query
An operation contained in a request message that asks QuickBooks to return specified data. Reports are a special type of query.

Remote Data Sharing
A feature that allows an integrated application to be installed on a system without QuickBooks and to communicate remotely with QuickBooks. (No changes to the application are required to take advantage of this feature.)

request
A message sent by the application to QuickBooks.

request message set
A collection of request messages.

response
A message sent by QuickBooks to the application. A response message contains both data and status code information.

response message set
A collection of response messages.

SDK
Software Development Kit.

SDK runtime
The portion of QuickBooks that processes application request message sets and builds response message sets.

session
Within a connection, a period during which a particular company file is open.

subscription
(several uses) 1. Can refer to enrollment in a payroll service—required for access to certain data such as payroll information, or 2. Can refer to the process of registering a callback function with QuickBooks to notify interest in certain types of events. (See Chapter 5 of the *Concepts Manual*.)

tag
In qbXML, a tag is the generic name for either a start tag or an end tag. A start tag consists of an **element** or **aggregate** name surrounded by angled brackets. An end tag is the same as a start tag except that it includes a forward slash immediately preceding the name.

ticket
A session handle that is sent to QuickBooks along with a message request.

transaction
An object that corresponds to the basic accounting entities in a business.

user interface (UI) event
Occur when the QuickBooks user interacts with the QuickBooks user interface (excluding menu items that have been added by integrated applications). Examples of user interface events are when the user opens or closes the company file.

user interface (UI) extension event
Occurs when the QuickBooks user clicks on a custom menu item (that is, a menu item that was added by an integrated application).

void
Voiding a transaction sets its amount to zero but maintains a record of the transaction. List items cannot be voided. *See also* delete.

XML
eXtensible Markup Language.

APPENDIX A
EXAMPLES

This chapter includes sample Visual Basic code that illustrates the different APIs you can use for creating and interpreting QuickBooks messages. The module includes two routines that add an invoice to QuickBooks. The first uses the Microsoft XML DOM parser and raw qbXML to build the add request and parse the response. The second routine (page 55) uses the QBFC Library to build and parse the messages.

This example adds a new shipping address for the customer Kristy Abercrombie. It also adds a service item, a non-inventory item, and an item group to the invoice.

This sample is also included in the online examples for the SDK.

```
Public Sub qbXML_AddInvoice()

'Declare various utility variables
    Dim boolSessionBegun As Boolean
'We want to know if we've begun a session so we can end it if an
'error sends us to the exception handler
    boolSessionBegun = False

  Dim strTicket As String
  Dim strXMLRequest As String
  Dim strXMLResponse As String

'Set up an error handler to catch exceptions
On Error GoTo Errs

'Create the RequestProcessor object using QBXMLRP2Lib
Dim qbXMLCOM as QBXMLRP2Lib.RequestProcessor

'Set up the RequestProcessor object and connect to QuickBooks
  Set qbXMLCOM = New QBXMLRP2Lib.RequestProcessor

'Our App will be named Add Invoice Sample
```

```
  qbXMLCOM.OpenConnection2 "", "Add Invoice Sample", localQBD

'Use the currently open company file
  strTicket = qbXMLCOM.BeginSession("", QBXMLRP2Lib.qbFileOpenDoNotCare)
  boolSessionBegun = True

'Create a DOM document object for creating our request.
  Dim xmlInvoiceAdd As MSXML2.DOMDocument
  Set xmlInvoiceAdd = new MSXML2.DOMDocument

'Create the QBXML aggregate
  Dim rootElement As IXMLDOMNode
  Set rootElement = xmlInvoiceAdd.createElement("QBXML")
  xmlInvoiceAdd.appendChild rootElement

'Add the QBXMLMsgsRq aggregate to the QBXML aggregate
  Dim QBXMLMsgsRqNode As IXMLDOMNode
  Set QBXMLMsgsRqNode = xmlInvoiceAdd.createElement("QBXMLMsgsRq")
  rootElement.appendChild QBXMLMsgsRqNode

'If we were writing a real application this is where we would add
'a newMessageSetID so we could perform error recovery. Any time a
'request contains an add, delete, modify or void request developers
'should use the error recovery mechanisms.

'Set the QBXMLMsgsRq onError attribute to continueOnError
  Dim onErrorAttr As IXMLDOMAttribute
  Set onErrorAttr = xmlInvoiceAdd.createAttribute("onError")
  onErrorAttr.Text = "stopOnError"
  QBXMLMsgsRqNode.Attributes.setNamedItem onErrorAttr

'Add the InvoiceAddRq aggregate to QBXMLMsgsRq aggregate
  Dim InvoiceAddRqNode As IXMLDOMNode
  Set InvoiceAddRqNode = _
    xmlInvoiceAdd.createElement("InvoiceAddRq")
  QBXMLMsgsRqNode.appendChild InvoiceAddRqNode

'Note - Starting requestID from 0 and incrementing it for each
'request allows a developer to load the response into a QBFC and
'parse it with QBFC

'Set the requestID attribute to 0
  Dim requestIDAttr As IXMLDOMAttribute
  Set requestIDAttr = xmlInvoiceAdd.createAttribute("requestID")
  requestIDAttr.Text = "0"
  InvoiceAddRqNode.Attributes.setNamedItem requestIDAttr
```

```
'Add the InvoiceAdd aggregate to InvoiceAddRq aggregate
  Dim InvoiceAddNode As IXMLDOMNode
  Set InvoiceAddNode = _
    xmlInvoiceAdd.createElement("InvoiceAdd")
  InvoiceAddRqNode.appendChild InvoiceAddNode

'Add the CustomerRef aggregate to InvoiceAdd aggregate
  Dim CustomerRefNode As IXMLDOMNode
  Set CustomerRefNode = _
    xmlInvoiceAdd.createElement("CustomerRef")
  InvoiceAddNode.appendChild CustomerRefNode

'Add the FullName element to CustomerRef aggregate
  Dim FullNameElement As IXMLDOMElement
  Set FullNameElement = xmlInvoiceAdd.createElement("FullName")
  FullNameElement.Text = "Abercrombie, Kristy"
  CustomerRefNode.appendChild FullNameElement

'Add the RefNumber element to InvoiceAdd aggregate
  Dim RefNumberElement As IXMLDOMElement
  Set RefNumberElement = xmlInvoiceAdd.createElement("RefNumber")
  RefNumberElement.Text = "qbXMLAdd"
  InvoiceAddNode.appendChild RefNumberElement

'Add the ShipAddress aggregate to InvoiceAdd aggregate
  Dim ShipAddressNode As IXMLDOMNode
  Set ShipAddressNode = _
    xmlInvoiceAdd.createElement("ShipAddress")
  InvoiceAddNode.appendChild ShipAddressNode

'Add the Addr1 element to ShipAddress aggregate
  Dim Addr1Element As IXMLDOMElement
  Set Addr1Element = xmlInvoiceAdd.createElement("Addr1")
  Addr1Element.Text = "Kristy Abercrombie"
  ShipAddressNode.appendChild Addr1Element

'Add the Addr2 element to ShipAddress aggregate
  Dim Addr2Element As IXMLDOMElement
  Set Addr2Element = xmlInvoiceAdd.createElement("Addr2")
  Addr2Element.Text = "c/o Enviromental Designs"
  ShipAddressNode.appendChild Addr2Element

'Add the Addr3 element to ShipAddress aggregate
  Dim Addr3Element As IXMLDOMElement
  Set Addr3Element = xmlInvoiceAdd.createElement("Addr3")
  Addr3Element.Text = "1521 Main Street"
  ShipAddressNode.appendChild Addr3Element
```

```
'Add the Addr4 element to ShipAddress aggregate
  Dim Addr4Element As IXMLDOMElement
  Set Addr4Element = xmlInvoiceAdd.createElement("Addr4")
  Addr4Element.Text = "Suite 204"
  ShipAddressNode.appendChild Addr4Element

'Add the City element to ShipAddress aggregate
  Dim CityElement As IXMLDOMElement
  Set CityElement = xmlInvoiceAdd.createElement("City")
  CityElement.Text = "Culver City"
  ShipAddressNode.appendChild CityElement

'Add the State element to ShipAddress aggregate
  Dim StateElement As IXMLDOMElement
  Set StateElement = xmlInvoiceAdd.createElement("State")
  StateElement.Text = "CA"
  ShipAddressNode.appendChild StateElement

'Add the PostalCode element to ShipAddress aggregate
  Dim PostalCodeElement As IXMLDOMElement
  Set PostalCodeElement = xmlInvoiceAdd.createElement("PostalCode")
  PostalCodeElement.Text = "90139"
  ShipAddressNode.appendChild PostalCodeElement

'We're going to add three line items, so use the same code and just
'change the values on each iteration
'Declare all of the DOM objects we'll be using here
  Dim InvoiceLineNode As IXMLDOMNode
  Dim ItemRefNode As IXMLDOMNode
  Dim DescElement As IXMLDOMElement
  Dim QuantityElement As IXMLDOMElement
  Dim RateElement As IXMLDOMElement

'Declare the other variables we'll be using in the loop
  Dim strDesc As String
  Dim strItemRefName As String
  Dim strItemFullName As String
  Dim strQuantity As String
  Dim strRate As String

  Dim i As Integer
  For i = 1 To 3
  'We'll add standard items for the first two lines and a group item
  'for the third
    If i <> 3 Then
    'Add the InvoiceLine aggregate to InvoiceAdd aggregate
```

```
    Set InvoiceLineNode = _
      xmlInvoiceAdd.createElement("InvoiceLineAdd")
    InvoiceAddNode.appendChild InvoiceLineNode
    strItemRefName = "ItemRef"

    If i = 1 Then
      strItemFullName = "Installation"
      strDesc = ""
      strQuantity = "2"
      strRate = ""
    Else
      strItemFullName = "Lumber:Trim"
      strDesc = "Door trim priced by the foot"
      strQuantity = "26"
      strRate = "1.37"
    End If
  Else
  'Add the InvoiceLineGroupAdd aggregate to InvoiceAdd aggregate
    Set InvoiceLineNode = _
      xmlInvoiceAdd.createElement("InvoiceLineGroupAdd")
    InvoiceAddNode.appendChild InvoiceLineNode
    strItemRefName = "ItemGroupRef"

    strItemFullName = "Door Set"
    strQuantity = ""
    strRate = ""
  End If

'Add the ItemRef aggregate to InvoiceAdd aggregate
  Set ItemRefNode = _
    xmlInvoiceAdd.createElement(strItemRefName)
  InvoiceLineNode.appendChild ItemRefNode

'Add the FullName element to ItemRef aggregate
  Set FullNameElement = xmlInvoiceAdd.createElement("FullName")
  FullNameElement.Text = strItemFullName
  ItemRefNode.appendChild FullNameElement

'We don't want to cause QuickBooks to do more work than it needs
'to so only include elements where we want to set the value
  If strDesc <> "" Then
    Set DescElement = xmlInvoiceAdd.createElement("Desc")
    DescElement.Text = strDesc
    InvoiceLineNode.appendChild DescElement
  End If

  If strQuantity <> "" Then
```

```
      Set QuantityElement = xmlInvoiceAdd.createElement("Quantity")
      QuantityElement.Text = strQuantity
      InvoiceLineNode.appendChild QuantityElement
    End If

    If strRate <> "" Then
      Set RateElement = xmlInvoiceAdd.createElement("Rate")
      RateElement.Text = strRate
      InvoiceLineNode.appendChild RateElement
    End If

  Next

'We're adding the prolog using text strings
  strXMLRequest = _
    "<?xml version=""1.0"" ?>" & _
    "<?qbxml version=""3.0""?>" & _
    rootElement.xml

  strXMLResponse = qbXMLCOM.ProcessRequest(strTicket, strXMLRequest)

' Uncomment the following to see the request and response XML for debugging
' MsgBox strXMLRequest, vbOKOnly, "RequestXML"
' MsgBox strXMLResponse, vbOKOnly, "ResponseXML"

  qbXMLCOM.EndSession strTicket
  boolSessionBegun = False
  qbXMLCOM.CloseConnection

'Set up a DOM document object to load the response into
  Dim xmlInvoiceAddResponse As MSXML2.DOMDocument
  Set xmlInvoiceAddResponse = new MSXML2.DOMDocument

'Parse the response XML
  xmlInvoiceAddResponse.async = False
  xmlInvoiceAddResponse.loadXML (strXMLResponse)

'Get the status for our add request
  Dim nodeInvoiceAddRs As IXMLDOMNodeList
  Set nodeInvoiceAddRs = xmlInvoiceAddResponse.getElementsByTagName
                         ("InvoiceAddRs")

  Dim rsStatusAttr        As IXMLDOMNamedNodeMap
  Set rsStatusAttr = nodeInvoiceAddRs.Item(0).Attributes
  Dim strAddStatus As String
  strAddStatus = rsStatusAttr.getNamedItem("statusCode").nodeValue
```

```
'If the status is bad, report it to the user
  If strAddStatus <> "0" Then
    MsgBox "qbXML_AddInvoice unexpexcted Error - " &
            rsStatusAttr.getNamedItem("statusMessage").nodeValue
    Exit Sub
  End If

'Parse the Add response and add the InvoiceAdds to the InvoiceAdd text box
  Dim InvoiceAddRsNode As IXMLDOMNode
  Set InvoiceAddRsNode = nodeInvoiceAddRs.Item(0)

'We only expect one InvoiceRet aggregate in an InvoiceAddRs
  Dim InvoiceRetNode    As IXMLDOMNode
  Set InvoiceRetNode = InvoiceAddRsNode.selectSingleNode("InvoiceRet")

'Get the data for the InvoiceDisplay form and set it
  frmInvoiceDisplay.txtInvoiceNumber = _
    InvoiceRetNode.selectSingleNode("RefNumber").Text
  frmInvoiceDisplay.txtInvoiceDate.Text = _
    InvoiceRetNode.selectSingleNode("TxnDate").Text
  frmInvoiceDisplay.txtDueDate.Text = _
    InvoiceRetNode.selectSingleNode("DueDate").Text
  frmInvoiceDisplay.txtSubtotal.Text = _
    InvoiceRetNode.selectSingleNode("Subtotal").Text
  frmInvoiceDisplay.txtSalesTax.Text = _
    InvoiceRetNode.selectSingleNode("SalesTaxTotal").Text
  frmInvoiceDisplay.txtInvoiceTotal.Text = _
    Str(CDbl(frmInvoiceDisplay.txtSubtotal.Text) + _
        CDbl(frmInvoiceDisplay.txtSalesTax.Text))

'Get the invoice lines and display information about them

'We know there are three invoice lines, but lets pretend we didn't
'know that.  We'll start at the last child node and move back
'until we first get an InvoiceLineRet or InvoiceLineGroupRet, then
'stop when we stop finding them and move forward again until we
'don't.

'Get the child nodes of the InvoiceAddRs
  Dim InvoiceNodeList As IXMLDOMNodeList
  Set InvoiceNodeList = InvoiceRetNode.childNodes

  Dim l As Long
  l = InvoiceNodeList.length - 1

  Do While _
    InvoiceNodeList.Item(l).nodeName <> "InvoiceLineRet" And _
```

```
    InvoiceNodeList.Item(l).nodeName <> "InvoiceLineGroupRet"
    l = l - 1
  Loop

  Do While _
    InvoiceNodeList.Item(l).nodeName = "InvoiceLineRet" Or _
    InvoiceNodeList.Item(l).nodeName = "InvoiceLineGroupRet"
    l = l - 1
  Loop
  l = l + 1

  Do While l < InvoiceNodeList.length
    If InvoiceNodeList.Item(l).nodeName = "InvoiceLineRet" Then
      frmInvoiceDisplay.txtInvoiceLines.Text = _
        frmInvoiceDisplay.txtInvoiceLines.Text & _
        InvoiceNodeList.Item(l).selectSingleNode("ItemRef").selectSingleNode(
                "FullName").Text & vbCrLf l = l + 1
    ElseIf InvoiceNodeList.Item(l).nodeName = "InvoiceLineGroupRet" Then
      frmInvoiceDisplay.txtInvoiceLines.Text = _
        frmInvoiceDisplay.txtInvoiceLines.Text & _
        InvoiceNodeList.Item(l).selectSingleNode("ItemGroupRef").selectSingleNode(
                "FullName").Text & vbCrLf l = l + 1
    Else
      l = InvoiceNodeList.length
    End If
  Loop

  Load frmInvoiceDisplay
  frmInvoiceDisplay.Show

  Exit Sub

Errs:
  If Err.Number = &H80040416 Then
    MsgBox "You must have QuickBooks running with the company" & vbCrLf & _
           "file open to use this program."
    qbXMLCOM.CloseConnection
    End
  ElseIf Err.Number = &H80040422 Then
    MsgBox "This QuickBooks company file is open in single user mode and"
           & vbCrLf & _ "another application is already accessing it.  Please exit
           the" & vbCrLf & _
           "other application and run this application again."
    qbXMLCOM.CloseConnection
    End
  Else
    MsgBox "HRESULT = " & Err.Number & " (" & Hex(Err.Number) & _
```

```
          ") " & vbCrLf & vbCrLf & Err.Description

    If boolSessionBegun Then
      qbXMLCOM.EndSession strTicket
      qbXMLCOM.CloseConnection
    End If

    End
  End If
End Sub



'QBFC routine begins here...

Public Sub QBFC_AddInvoice()

'Declare various utility variables
  Dim boolSessionBegun As Boolean
  boolSessionBegun = False

On Error GoTo ErrHandler

'We want to know if we've begun a session so we can end it if an
'error sends us to the exception handler
  boolSessionBegun = False

' Create the session manager object using QBFC 3.0, and use this
' object to open a connection and begin a session with QuickBooks.
  Dim sessionManager As New QBSessionManager
  sessionManager.OpenConnection "", "Add Invoice Sample"
  sessionManager.BeginSession "", omDontCare
  boolSessionBegun = True

' Create the message set request object
  Dim requestMsgSet As IMsgSetRequest
  Set requestMsgSet = sessionManager.CreateMsgSetRequest("US",3,0)

' Initialize the message set request's attributes
  requestMsgSet.Attributes.OnError = roeStop

' Add the request to the message set request object
  Dim invoiceAdd As IInvoiceAdd
  Set invoiceAdd = requestMsgSet.AppendInvoiceAddRq

' Set the IinvoiceAdd field values
```

```
  invoiceAdd.CustomerRef.FullName.setValue "Abercrombie, Kristy"
  invoiceAdd.RefNumber.setValue "QBFCAdd"

' Set up the shipping address for the invoice
  invoiceAdd.ShipAddress.Addr1.setValue "Kristy Abercrombie"
  invoiceAdd.ShipAddress.Addr2.setValue "c/o Enviromental Designs"
  invoiceAdd.ShipAddress.Addr3.setValue "1521 Main Street"
  invoiceAdd.ShipAddress.Addr4.setValue "Suite 204"
  invoiceAdd.ShipAddress.City.setValue "Culver City"
  invoiceAdd.ShipAddress.State.setValue "CA"
  invoiceAdd.ShipAddress.PostalCode.setValue "90139"

' Create the first line item for the invoice
  Dim invoiceLineAdd As IInvoiceLineAdd
  Set invoiceLineAdd = invoiceAdd.ORInvoiceLineAddList.Append.invoiceLineAdd

' Set the values for the invoice line
  invoiceLineAdd.ItemRef.FullName.setValue "Installation"
  invoiceLineAdd.Quantity.setValue 2

'Create the second line item for the invoice
  Set invoiceLineAdd = invoiceAdd.ORInvoiceLineAddList.Append.invoiceLineAdd

' Set the values for the invoice line
  invoiceLineAdd.ItemRef.FullName.setValue "Lumber:Trim"
  invoiceLineAdd.Desc.setValue "Door trim priced by the foot"
  invoiceLineAdd.Quantity.setValue 26
  invoiceLineAdd.ORRate.Rate.setValue 1.37

'Create the third line item with is a item group line
  Dim invoiceGroupLineAdd As IInvoiceLineGroupAdd
  Set invoiceGroupLineAdd = invoiceAdd.ORInvoiceLineAddList.
                            Append.InvoiceLineGroupAdd

' Set the values for the invoice line
  invoiceGroupLineAdd.ItemGroupRef.FullName.setValue "Door Set"

' Perform the request and obtain a response from QuickBooks
  Dim responseMsgSet As IMsgSetResponse
  Set responseMsgSet = sessionManager.DoRequests(requestMsgSet)

' Close the session and connection with QuickBooks.
  sessionManager.EndSession
  boolSessionBegun = False
  sessionManager.CloseConnection

' Uncomment the following to see the request and response XML for debugging
```

```
' MsgBox requestMsgSet.ToXMLString, vbOKOnly, "RequestXML"
' MsgBox responseMsgSet.ToXMLString, vbOKOnly, "ResponseXML"

' Interpret the response
  Dim response As IResponse

' The response list contains only one response,
' which corresponds to our single request
  Set response = responseMsgSet.ResponseList.GetAt(0)

  msg = "Status: Code = " & CStr(response.StatusCode) & _
        ", Message = " & response.StatusMessage & _
        ", Severity = " & response.StatusSeverity & vbCrLf

' The Detail property of the IResponse object
' returns a Ret object for Add and Mod requests.
' In this case, the Ret object is IInvoiceRet.

'For help finding out the Detail's type, uncomment the following
'line:
'MsgBox response.Detail.Type.GetAsString

  Dim invoiceRet As IInvoiceRet
  Set invoiceRet = response.Detail

'Declare variables for subtotal and tax so we can compute the
'invoice total.  Although we could use BalanceRemaining in this
'case, lets compute it anyway.
  Dim dblSubtotal As Double
  Dim dblTax As Double

'Fill in the text fields of the Invoice Display form
  frmInvoiceDisplay.txtInvoiceNumber = invoiceRet.RefNumber.getValue
  frmInvoiceDisplay.txtInvoiceDate = invoiceRet.TxnDate.getValue
  frmInvoiceDisplay.txtDueDate = invoiceRet.DueDate.getValue
  dblSubtotal = invoiceRet.Subtotal.getValue
  frmInvoiceDisplay.txtSubtotal = invoiceRet.Subtotal.GetAsString
  dblTax = invoiceRet.SalesTaxTotal.getValue
  frmInvoiceDisplay.txtSalesTax = invoiceRet.SalesTaxTotal.GetAsString
  frmInvoiceDisplay.txtInvoiceTotal = Str(dblSubtotal + dblTax)

'Put the lines in the display
  Dim orInvoiceLineRetList As QBFC3Lib.IORInvoiceLineRetList
  Set orInvoiceLineRetList = invoiceRet.orInvoiceLineRetList

  Dim i As Integer
  For i = 0 To orInvoiceLineRetList.Count - 1
```

```
    If invoiceRet.orInvoiceLineRetList.GetAt(i).ortype = _
         orilrInvoiceLineRet Then
      frmInvoiceDisplay.txtInvoiceLines.Text = _
        frmInvoiceDisplay.txtInvoiceLines.Text & _
        invoiceRet.orInvoiceLineRetList.GetAt(i).InvoiceLineRet.ItemRef.FullName.g
etValue & _
        vbCrLf
    Else
      frmInvoiceDisplay.txtInvoiceLines.Text = _
        frmInvoiceDisplay.txtInvoiceLines.Text & _
        invoiceRet.orInvoiceLineRetList.GetAt(i).InvoiceLineGroupRet.ItemGroupRef.
FullName.getValue & _
        vbCrLf
    End If
  Next

'Show the Invoice Display form
  Load frmInvoiceDisplay
  frmInvoiceDisplay.Show
  Exit Sub

ErrHandler:
  If Err.Number = &H80040416 Then
    MsgBox "You must have QuickBooks running with the company" & vbCrLf & _
           "file open to use this program."
    sessionManager.CloseConnection
    End
  ElseIf Err.Number = &H80040422 Then
    MsgBox "This QuickBooks company file is open in single user mode and"
            & vbCrLf & _"another application is already accessing it.  Please exit
            the" & vbCrLf & _
           "other application and run this application again."
    sessionManager.CloseConnection
    End
  Else
    MsgBox "HRESULT = " & Err.Number & " (" & Hex(Err.Number) & _
           ") " & vbCrLf & vbCrLf & Err.Description

    If boolSessionBegun Then
      sessionManager.EndSession
      sessionManager.CloseConnection
    End If

    End
  End If
End Sub
```

INDEX

S

T

U

V

W

X