QuickBooks® SDK

Technical Overview

Version 3.0

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/.

As well as

http://www.quicken.com.au/Partners/SoftwareDP/default.aspx

CONTENTS

CHAPTER 1 INTRODUCTION 1

WHY AN SDK FOR QUICKBOOKS? 1
SHARING DATA WITH QUICKBOOKS 1
A COMMON SDK FOUNDATION 2
SUPPORT FOR MULTIPLE PROGRAMMING LANGUAGES 2
DESIGN PRINCIPLES 2
RUNTIME COMPONENTS OF THE QUICKBOOKS SDK 3
WHAT'S INCLUDED IN THE QUICKBOOKS SDK? 3
Software libraries. *3*
qbXML specification. *3*
Example qbXML file. *3*
Documentation. *3*
Utilities. *3*
Sample applications. *4*
WHICH API TO USE? 4
SUPPORTED VERSIONS AND PRODUCTS 4
FAQS ABOUT VERSIONING 5
BEFORE YOU BEGIN 6
Documentation Roadmap *6*
Concepts Manual *6*
Developer's Guides *7*
Onscreen Reference *7*
WHAT'S NEXT? 7

CHAPTER 2 COMMUNICATING WITH QUICKBOOKS 2 8

TYPICAL SEQUENCE 8
SETTING UP THE COMMUNICATIONS CHANNEL 8
WHAT'S IN A MESSAGE? 8
KEY CONCEPTS 9
RECEIVING EVENTS FROM QUICKBOOKS 9
INTEGRATION WITH THE QUICKBOOKS USER INTERFACE 10
SECURITY CONCERNS 10
AUTHENTICATION 10
AUTHORISATION 10
ACCESS TO PERSONAL DATA 11
ERROR HANDLING 11
SYNCHRONISATION 11

CHAPTER 3 REQUEST AND RESPONSE MESSAGES 3 12

"OBJECTS" IN THE SDK 12
Lists *12*
Transactions *13*
Operations *14*
NAMING CONVENTIONS FOR MESSAGES 14
REQUEST MESSAGES 14
RESPONSE MESSAGES 14
RETURN STATUS 15
ELEMENTS AND AGGREGATES 15
OBJECT REFERENCES 15
QBXML SAMPLE REQUEST AND RESPONSE 16
QUERIES AND FILTERS 17
REPORTS 17
CATEGORIES OF REPORTS 18
CUSTOMISATION 19
MESSAGES SUPPORTED BY THE SDK FOR QUICKBOOKS 19

GLOSSARY 21

APPENDIX A 25

INDEX 30

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. (For simplicity and directness, the
"you" in this document refers to the third party application developer.)
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 (Version 2003 and beyond):

QuickBooks Pro
QuickBooks Accountant's Edition
QuickBooks Premier (and industry-specific editions)
QuickBooks Enterprise Edition (and industry-specific editions)

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).

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) authorises the connection between your application and QuickBooks. The small-business owner also sets up permissions for authorised 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 synchronisation facilities to ensure that data is not lost or destroyed and that application data remains synchronised 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 behaviour 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

When you create an integrated application with the QuickBooks SDK you must use the qbXML Request Processor API, as the QuickBooks Foundation Class (QBFC) Library is not supported for the Australian/New Zealand/Singapore versions of QuickBooks

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.

You will need a basic understanding of the elements, aggregates, and attributes defined in the qbXML specification.

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 include: qbXML Request Processor Interface

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.

Example qbXML file.

This file (*qbxmlops30.xml*) includes examples of all qbXML request and response messages.

Documentation.

In addition to this *Technical Overview*, the following QuickBooks SDK manuals are available:
Concepts Manual
Developer's Guide for QuickBooks (U.S. and Canadian Editions)
Onscreen Reference

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 desktop versions 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?

The only API that you can use with the Australian/New Zealand/Singapore version of QuickBooks is the qbXML Request Processor API. The QBFC API is not supported.

The qbXML Request Processor API requires you to create and parse qbXML documents explicitly. Appendix A includes sample code illustrating the use of the qbXML request processor.

Supported Versions and Products

The qbXML specification is built into QuickBooks. Table 1 lists the different versions of QuickBooks and corresponding versions of the qbXML specification that each version contains. An application written for SDK 3.0 (qbXML specification version 3.0) is supported by QuickBooks 2004. (It can also run against QuickBooks 2003, but functionality will be limited to that of the earlier specifications, as listed in Table 1.)

Starting with the current release (SDK 3.0), a new version of the Request Processor is available, namely QBXMLRP2. Backward compatibility is maintained for the older Request Processor (QBXMLRP), but new functionality such as event notification and integration with the QuickBooks user interface is available only with the new Request Processor. (The QBXMLRP2 Request Processor is redistributable and can be included in the installation for your application.)

Table 1 QuickBooks and Supported qbXML Specification Versions

Version of QuickBooks	qbXML Specification Version
QuickBooks 2004 Premier, Pro, Accountant's Edition, and Enterprise 4.0 (using QBXMLRP2)	3.0, 2.1, 2.0
QuickBooks 2004 Premier, Pro, Accountant's Edition, and Enterprise 4.0 (using QBXMLRP)	2.0
QuickBooks 2003 Pro, Accountant's Edition, and Enterprise 2.0 (using QBXMLRP)	2.0

QBXMLRP2 or QBXMLRP? We recommend you upgrade existing applications to use QBXMLRP2. Features new in SDK 3.0 are available only with QBXMLRP2; in addition, this Request Processor solves some problems found in the older Request Processor.

FAQs about Versioning

The following paragraphs answer some frequently asked questions and provide further detail on versioning. Also see the *Concepts Manual*, which includes diagrams and sample code related to this topic.

Q: I am writing an application for QuickBooks 2004. Will it work with QuickBooks 2003?

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 Table 1. 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 QuickBooks 2003. Will it work with QuickBooks 2004?

A: As shown in Table 1, QuickBooks 2004 supports all of the previous versions of the qbXML specification, so any message sent to QuickBooks 2003 will also work with QuickBooks 2004.

Q: I want to add some SDK 3.0 functionality to my application, which was written for QuickBooks 2003. How do I proceed?

A: First of all, be sure to use the new Request Processor, **QBXMLRP2**. If you want your application to work with QuickBooks 2003, you also need to redistribute the QBXMLRP2 installer or merge module with your application (include it in your installation) so that it will be available on systems with these earlier versions of QuickBooks 2003.
Next, when you update your application, be sure to separate the 3.0 functionality in your code and invoke it only when you're running against QuickBooks 2004 (or later). After a session has begun, check the qbXML version. You can keep the old qbXML version numbers for older messages, but new 3.0 requests will require a 3.0 prolog. In areas where functionality has changed, your application needs to include a branch for the original functionality (in your case, version
2.0) and a separate branch taking advantage of new functionality (version 3.0). 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
Programming language of your choice
Microsoft's Component Object Model (COM)
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 4.0 (MSXML).

Documentation Roadmap

In addition to this *Technical Overview*, you will also need to become familiar with at least three other pieces of documentation.

Concepts Manual

The *Concepts Manual* is your starting point for practical information on how 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. With this release, it includes information on new features including event notification and integrating with the QuickBooks user interface. It also focuses on general application concerns such as error recovery, how to synchronise application data with QuickBooks, and how to anticipate typical user problems in your application.

Developer's Guides

Each *Developer's Guide* describes the communication and authentication models used by QuickBooks. It presents a simple "Hello, QuickBooks" sample that shows

1. Setting up a connection and the communication infrastructure with QuickBooks
2. Sending a simple request to QuickBooks
3. Receiving a response
4. Closing the connection

Onscreen Reference

Once you're ready to program, you'll use the *Onscreen Reference* often to find out the exact syntax of a given request or response. This online tool provides detailed reference information for developer libraries. 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. This online tool is also a handy place to look up the meaning of any SDK error code.

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:

- The manuals listed in the "Documentation Roadmap" section of this *Technical Overview*
- Web site for the Intuit Developer Network (IDN): *http://developer.intuit.com*

CHAPTER 2
COMMUNICATING WITH QUICKBOOKS

Basic communication between your application and QuickBooks is based on a client/server model. 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*.

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

- Your application runs as a separate process and resides on the same desktop system as QuickBooks.

- Using Remote Data Sharing, your application resides on a system with the Remote Data Sharing Client software. QuickBooks resides on a remote system, which has the Remote Data Sharing Server software installed on it. Remote Data Sharing is transparent to you, the developer, and to your application. Your application functions as if it were installed locally with QuickBooks.

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

QuickBooks uses either the qbXML Request Processor API 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.

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 authorise the connection between QuickBooks and your application.

For QuickBooks, a *session* occurs within the context of a connection and is tied to a specific company data file. 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.

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.

Receiving Events from QuickBooks

In addition to the request/response model, SDK 3.0 provides a new 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:

A data event - occurs 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

Integration with the QuickBooks User Interface

Beginning with SDK 3.0, your application can also add a menu with associated subitems to QuickBooks (see the *Concepts Manual* for details). Your application can also display reports as well as some list elements and transactions inside QuickBooks.

QuickBooks

Developers for QuickBooks Australia, New Zealand, and Singapore integrated applications can only use the qbXML Request Processor API.

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

OpenConnection
BeginSession
ProcessRequest
EndSession
CloseConnection

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 authorisation 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 Network website *(http:// developer.intuit.com)* describes how to obtain a digital certificate using the Microsoft Authenticode™ technology.

Authorisation

For QuickBooks a BeginSession call initiates the authorisation 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 authorises 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).

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.

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.

Synchronisation

One of the most important features of the SDK is that it allows you to ensure that the data in your application is always synchronised 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. Synchronisation problems can be complex, and their resolution is usually dependent on the individual application. The *Concepts*
Manual 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 3 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, beginning with SDK 3.0, your application can effect various changes in QuickBooks user interface behavior, such as causing a report to be displayed in QuickBooks, displaying a form to create a new list element or transaction, or displaying a form to edit an existing list element or transaction.

"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, Tax Code, Other Charge, Subtotal, Fixed Asset
- *Simple Lists* - Tax Code, Class, Standard Terms, Date Driven Terms, Ship Method, Payment Method, Sales Rep, Job Type, Customer Type, Vendor Type, Customer Message, Template, To Do
- *Payroll Lists* - Wage Payroll, NonWage Payroll

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 Cheque
- Bill Payment Credit Card
- Item Receipt
- Tax Payment

Non-Posting Transactions:

- Purchase Order
- Estimate
- Sales Order

Bank Transactions and Sales Receipts:

- Cheque
- Deposit
- Sales Receipt

Credit Card Transactions:

- Credit Card Charge
- Credit Card Credit

Other Transactions:

- Journal Entry
- Time Tracking
- Inventory Adjustment

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
- *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

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 *Onscreen Reference*, which uses this paradigm of building up a message name: first, you select an object name and operation, then *request* or *response*.

Response Messages

The naming convention 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 *Onscreen 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 *Onscreen 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="AU3.0"?>
<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>2003-08-16T11:22:07-08:00
                                </TimeCreated>
                                <TimeModified>2003-08-16T11:22:07-08:00
                                </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).

Customisation

You have the option of customising the requested report in a number of ways. If you request a report without any customisation, the report uses the default parameters employed by the QuickBooks user interface. Depending on the report type, you can customise 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 QuickBooks

Table 3 Messages Supported by the SDK (v 3.0) for QuickBooks

Object	Type	Query	Add	Modify	Delete	Void
Account	List	yes	yes	no	yes	no
Class	List	yes	yes	no	yes	no
Company	Special	yes	no	no	no	no
CompanyActivity	Special	yes	no	no	no	no
Currency	List	yes	yes	yes	yes	no
Customer	List	yes	yes	yes	yes	no
Customer message	List	yes	yes	no	yes	no
Customer type	List	yes	yes	no	yes	no
Date-driven terms	List	yes	yes	no	yes	no
Discount item	List	yes	yes	yes	yes	no
Employee	List	yes	yes	yes	yes	no
Entity	List	yes	no	no	no	no
Fixed asset item	List	yes	yes	yes	yes	no
Group item	List	yes	yes	yes	yes	no
Host application	Special	yes	no	no	no	no
Inventory item	List	yes	yes	yes	yes	no
Inventory assembly item	List	yes	yes	yes	yes	no
Job type	List	yes	yes	no	yes	no
Non-inventory item	List	yes	yes	yes	yes	no
Non-wage payroll item	List	yes	no	no	no	no
Other charge item	List	yes	yes	yes	yes	no
Other name	List	yes	yes	yes	yes	no
Payment item	List	yes	yes	yes	yes	no
Payment method	List	yes	yes	no	yes	no
Preferences	Special	yes	no	no	no	no
Sales rep	List	yes	yes	yes	yes	no
Tax code	List	yes	yes	no	yes	no
Service item	List	yes	yes	yes	yes	No
Ship method	List	yes	yes	no	yes	No
Standard terms	List	yes	yes	no	yes	No
Subtotal item	List	yes	yes	yes	yes	No

Template	List	yes	no	no	no	No
Terms	List	yes	no	no	no	No
To do	List	yes	yes	no	yes	No
Vendor	List	yes	yes	yes	yes	No
Vendor type	List	yes	yes	no	yes	No
Wage payroll item	List	yes	yes	no	no	No
Bill	Txn	yes	yes	yes	yes	Yes
Bill payment Cheque	Txn	yes	yes	no	yes	Yes
Bill payment credit card	Txn	yes	yes	no	yes	Yes
Bill to pay	Txn	yes	no	no	no	No
Charge	Txn	yes	yes	yes	yes	Yes
Cheque	Txn	yes	yes	no	yes	Yes
Credit card charge	Txn	yes	yes	no	yes	Yes
Credit card credit	Txn	yes	yes	no	yes	Yes
Credit memo	Txn	yes	yes	Yes	yes	Yes
Deposit	Txn	yes	yes	no	yes	Yes
Estimate	Txn	yes	yes	yes	yes	No
Inventory adjustment	Txn	yes	yes	no	yes	no
Invoice	Txn	yes	yes	Yes	yes	Yes
Item Receipt	Txn	yes	no	No	yes	yes
Journal entry	Txn	yes	yes	no	yes	Yes
Purchase order	Txn	yes	yes	yes	yes	No
Receive payment	Txn	yes	yes	no	yes	no
Receive payment to deposit	Txn	yes	no	no	no	No
Sales order	Txn	yes	yes	yes	yes	No
Sales receipt	Txn	yes	yes	no	yes	Yes
Time tracking	Txn	yes	yes	no	yes	No
Vendor credit	Txn	yes	yes	no	yes	Yes
ListDisplay	UI Integration	no	yes	yes	no	No
TxnDisplay	UI Integration	no	yes	yes	no	No
AgingReport	Report	yes	no	no	no	No
BudgetSummary-Report	Report	yes	no	no	no	No
CustomDetailReport	Report	yes	no	no	no	No
CustomSummary-Report	Report	yes	no	no	no	No
GeneralDetailReport	Report	yes	no	no	no	No
GeneraSumaryReport	Report	yes	no	no	no	No
JobReport	Report	yes	no	no	no	No
PayrollDetailReport	Report	yes	no	no	no	No
PayrollSummary Report	Report	yes	no	no	no	No
TimeReport	Report	yes	no	no	no	No
DataExtDef	Special	yes	yes	yes	yes	No
DataExt	Special	no	yes	yes	yes	No
DataEventRecovery-Info	Event Notif.	yes	no	no	yes	No
DataEventSubscription	Event Notif.	yes	yes	no	no	No
UIEventSubscription	Event Notif.	yes	yes	no	no	No
UIExtensionSubscription	Event Notif.	yes	yes	no	no	No
Subscription	Event Notif.	no	no	no	yes	No

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 characterise 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

This chapter includes sample Visual Basic code that illustrates the APIs you can use for creating and interpreting QuickBooks messages. The module includes a routines that adds an invoice to QuickBooks. This uses the Microsoft XML DOM parser and raw qbXML to build the add request and parse the response.

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 booSessionBegun 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
booSessionBegun = 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.OpenConnection "", "Add Invoice Sample"
'Use the currently open company file
strTicket = qbXMLCOM.BeginSession("", QBXMLRP2Lib.qbFileOpenDoNotCare)
booSessionBegun = 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
booSessionBegun = 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 I As Long
I = InvoiceNodeList.length - 1
Do While _
InvoiceNodeList.Item(I).nodeName <> "InvoiceLineRet" And _
InvoiceNodeList.Item(I).nodeName <> "InvoiceLineGroupRet"
I = I - 1
Loop
Do While _
InvoiceNodeList.Item(I).nodeName = "InvoiceLineRet" Or _
InvoiceNodeList.Item(I).nodeName = "InvoiceLineGroupRet"
I = I - 1
Loop
I = I + 1
Do While I < InvoiceNodeList.length
If InvoiceNodeList.Item(I).nodeName = "InvoiceLineRet" Then
frmInvoiceDisplay.txtInvoiceLines.Text = _
frmInvoiceDisplay.txtInvoiceLines.Text & _
InvoiceNodeList.Item(I).selectSingleNode("ItemRef").selectSingleNode(
"FullName").Text & vbCrLf I = I + 1
ElseIf InvoiceNodeList.Item(I).nodeName = "InvoiceLineGroupRet" Then
frmInvoiceDisplay.txtInvoiceLines.Text = _
frmInvoiceDisplay.txtInvoiceLines.Text & _
InvoiceNodeList.Item(I).selectSingleNode("ItemGroupRef").selectSingleNode(
"FullName").Text & vbCrLf I = I + 1
Else
I = 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 booSessionBegun Then
qbXMLCOM.EndSession strTicket
qbXMLCOM.CloseConnection
End If
End
End If
End Sub
```

INDEX

A

Add operation 14
administrator 2, 10, 11
aggregates, defined 3, 15, 21, 22
authentication 7, 10
Authenticode™ 10
authorisation 10

B

BeginSession method 10, 25

C

Canadian Edition 3
Certificate 7, 10
certificate authority 10
checking the version of QuickBooks 5
client/server model 8
CloseConnection method 10, 27, 29
codes, status 8
communicating with QuickBooks 8
company data file 9
comparison of SDK API 4
Component Object Model (COM) 2, 6
Concepts Manual 6
connection between QuickBooks and your application 9

D

data events 21
Delete requests 14
Developer's Guide 7
Developer's Guide for QuickBooks 3
digital signature 10
Roadmap 6, 7

E

EndSession method 10, 27, 29
event notification 4, 6, 22
Events 9

F

FAQs (Frequently Asked Questions 5
Filters 17

G

General Detail reports 18

I

Intuit Developer Network (IDN) 7

K

Kristy Abercrombie 26

L

ListDelRq 14
Lists 12

M

message aggregate 16
message sets 8, 23
Messages 14, 19
naming conventions for 14
Request 4, 5, 6, 8, 10, 14, 16
Response 14, 16
Status 11, 15, 16

O

object references 15
objects 8, 12, 13, 14, 15, 17, 22, 26
Onscreen Reference 7, 14, 15
Operations 14, 22

P

personal data 11
programming standards 2
public key 10, 21
purpose of this document 1

Q

communicating with 10
components of 1, 3
methods 9, 10
QBFC 3, 4, 15, 22, 25
qbXML prolog 5
qbXML Request Processor 3, 4, 6, 8, 10
qbXML Request Processor API 3, 4, 6, 8, 10
qbXML specification 2, 3, 4, 5, 6, 15, 22
QBXMLRP2 ..... 4, 5, 6
Queries 17
QuickBooks 2003 ..... 4, 5, 6
QuickBooks Foundation Class 3, 22
QuickBooks Premier 1
QuickBooks Pro 1
QuickBooks SDK 1, 2, 3, 6, 11, 12, 19
QuickBooks user interface 4, 6, 9, 12, 17, 19, 24
sample code 4, 25
versions of 3, 4, 5, 6

R

Remote Data Sharing 8, 23
Remote Data Sharing Client 8
report data 17
Reports 17, 18, 23
request message 5, 8, 14, 15, 16, 23
response message 2, 3, 8, 14, 15, 22, 23
Runtime 3

S

qbXML 2, 3, 4, 5, 6, 8, 10, 15, 16, 17, 21, 22, 24, 25, 28
SDK manuals 3
SDKTest utility 3
session handle 9, 24

severity levels 11
small-business owner 1, 2, 10
Specification 5
Standard Profit and Loss Report 18
status codes 8

T

time tracking 1
Transactions 13
TxnDelRq 14
TxnVoidRq 14
types of 9, 11, 12, 13, 15, 24

U

user interface events 9

V

Validator utility 3
Void requests 14

X

XML 2, 3, 6, 15, 22, 24, 25, 27