QuickBooks® Merchant Service SDK

Developer's Guide for QuickBooks Merchant Service

Version 3.0

(April 2008)

QBMS SDK version 3.0, released April 2008. (c) 2008 Intuit Inc. All rights reserved.

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

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

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

For more information about the QuickBooks SDK and the SDK documentation, visit the i*ntuit developer web site*.

About This Guide

Chapter 1: Introduction

Chapter 2: Fraud Prevention Features

Chapter 3: Running Credit Card Transactions

(c) 2008 Intuit Inc. All rights reserved.

Chapter 4: Supporting QuickBooks Reconcile

Chapter 5: Signing Up for a PTC Test QBMS Merchant Account

Chapter 6: Testing Credit Card Transactions

Chapter 7: Error Handling

Chapter 8: Accessing QBMS from Desktop Applications

(c) 2008 Intuit Inc. All rights reserved.

(c) 2008 Intuit Inc. All rights reserved.

Appendix D: Supported Root Certificate Authorities

(c) 2008 Intuit Inc. All rights reserved.

About This Guide

This *Developer's Guide* describes the QuickBooks Merchant Service SDK (which we'll shorten to QBMS SDK or simply SDK in this guide). The purpose of this guide is to provide the details you need to know in order to successfully do credit card transactions from your application via qbmsXML messages.

Who Should Read This Guide

This guide is for developers who are creating desktop or server applications that integrate with QBMS and optionally with QuickBooks.

In order to create a desktop application, you should know a little about XML and how to assemble and post XML documents to a web URL (and handle responses) in your programming language of choice. You should know HTTPS since you need to post requests via HTTPS to QBMS. If you want to integrate with QuickBooks and/or QuickBooks Point of Sale as well, you'll need to know those products and their SDKs.

The knowledge requirements for hosted web applications are similar, but in addition you'll need to know how to get and present server and client certificates because QBMS requires a server certificate from a web app when QBMS makes callbacks to it, and a client certificate when the web app POSTs requests to the QBMS server.

Before You Begin

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

Terminology

In this guide, the term "desktop application" refers to applications using the desktop security model in which the application only needs a connection ticket to access a QBMS account. The term "hosted web application" refers to the hosted application security model, where the application needs a server certificate (for QBMS callbacks), a client certificate (for POSTing transactions to QBMS), and a connection ticket.

What's New in This Guide?

This version of the guide adds descriptions for the following new features available in QBMS qbmsXML spec 3.0:

(c) 2008 Intuit Inc. All rights reserved.

- Expansion of the existing CustomerCreditCard transaction requests to support restaurant authorizations.
- Expansion of the existing CustomerCreditCard transaction requests to support restaurant and lodging charges.
- A new request, CustomerCreditCardTxnIncrementalAuth, to handle an extension of a stay at a lodging.
- Support for merchant initiated batch close through a new request called MerchantBatchCloseRq and a new BatchID field in certain existing credit card transaction requests.

Components of the QBMS SDK

The QBMS SDK components provides everything you need to handle credit card transactions in your application via QuickBooks Merchant Services (QBMS) and also optionally save transaction data into QuickBooks.

The QBMS SDK components consist of:

- The test and production credit card transaction capabilities provided by QBMS.
- The qbmsXML specification that serves as an entry point into the QBMS credit card transaction functionality.
- A set of software tools, sample programs, and documentation to help you integrate credit card transaction capabilities into your application.
- Optionally, if you want to subsequently save credit card transaction data into QuickBooks, you can make use of qbXML functionality that supports the saving of QBMS credit card transaction data into QuickBooks.

The ability to save credit card data into QuickBooks and use its Reconciliation feature for card fees and check for funding status is an additional and useful feature. However, your application is not required to communicate with QuickBooks or use QuickBooks.

NOTE

There are no client-side runtime components. All applications, whether desktop applications or hosted web applications, communicate directly through HTTPS communication with the remote QBMS servers.

Technical Requirements

End users of applications integrated with QBMS must have a valid QuickBooks Merchant Service account. They can obtain an account online from *QBMS* or by phone.

(c) 2008 Intuit Inc. All rights reserved.

NOTE

A web link will be provided for third-party developer so they can automatically direct their customer/merchants to QBMS web sites to perform an online signup.

End users of applications that integrate QBMS transaction data with QuickBooks must also have a version of QuickBooks that supports the qbXML specification 4.1 and greater. Only QuickBooks versions 2005 R5, QuickBooks Enterprise Solutions 5.0 R4 and greater provide this support. (Support for Refund transactions in QuickBooks is only available beginning with QuickBooks 2006 and SDK version 5.0)

However, because QuickBooks 2008 supports the latest Payment Application Data Security Standard (PA DSS) requirements, we strongly recommend the use of QuickBooks 2008 and later, and qbmsXML spec 2.0 and later.

(c) 2008 Intuit Inc. All rights reserved.

(c) 2008 Intuit Inc. All rights reserved.

INTRODUCTION

This chapter contains a basic overview of the QBMS SDK.

Where to Find More Information

To find more information about the QBMS SDK, refer to the *QBMS Integration Center* website. In particular, you'll want to consult the *SDK knowledgebase*.

What is the QBMS SDK?

The QBMS SDK is an XML-based API that your application can use for credit card transactions using the QuickBooks Merchant Service (QBMS). The QBMS SDK supports card present, card not present, card swipe, void, and refund transactions as follows:

- Charge the customer's credit card (make a sale).
- Get an authorization for a transaction to be captured at a later time.
- Charge a transaction previously authorized over phone (voice auth).
- Capture a transaction previously authorized. Currently, there must be one and only one Capture operation for each Authorization. (Multiple captures for one authorization are not supported.)
- Void a previous transaction, including authorizations, charges, or refunds. Notice that this will succeed only if the transaction has not yet been settled.
- Obtain a refund.
- Issue one request that automatically results in a void or a refund based on transaction times and using only the original QBMS transaction ID rather than a credit card number.
- Invoke a merchant account query to determine certain things about the current merchant account, such as the credit card types the merchant accepts.
- In the credit card transactions, provide additional support for the restaurant and lodging industry.

Table 1-1 on page 12 shows the functionality provided.

Intuit Confidential (c) 2008 Intuit Inc. All rights reserved.

Table 1-1 QBMS Functionality At a Glance

Feature	Comment
Core Functionality	Ability to process credit cards through QBMS, either swiped or card not present.
Supported transactions	Authorization Incremental Authorizations (for extending a guest visit at a lodging) Capture (previous authorization) Credit Card Charge/Sale Merchant account query Refund Voice authorization Void VoidOrRefund (uses transaction ID, not credit card number, automatically determines whether to issue a void or a refund)
Key supported transaction features	Card security codes (such as CVC2, CVV2, CID) Card swipe (via the Track2Data field in transactions) AVS (address verification)
Credit Cards Supported	Visa MasterCard American Express Discover Diner's Club JCB
Supported Integrations with QuickBooks	Transaction data from QBMS can be saved in QuickBooks via the QB SDK SalesReceiptAdd, ReceivePaymentAdd, and ARRefundCreditCardAdd requests. Once this data is saved into QuickBooks, credit card fees can be accounted for using the Reconciliation feature and Get Funding Status. Note: We strongly recommend the use of qbmsXML 2.0 along with qbXML 6.0 and QB 2008 or greater, because this combination supports the latest PA DSS requirements. Note: the earliest qbXML spec and QuickBooks versions that support QBMS data is qbXML 4.1/QuickBooks 2005 R5, QBES 5.0 R4.

How Does QBMS Work with QuickBooks?

You can use the QBMS SDK only to do transactions via QBMS. But that would not take full advantage of its benefits, namely, the ability to save QBMS transaction request data into QuickBooks. Saving QBMS data into QuickBooks enables the merchant to use the QuickBooks Reconcile feature to account for credit card transaction fees and to get funding status.

If you intend to save QBMS transaction data into QuickBooks, then you'll be saving certain parts of the transaction request and the transaction response for inclusion in certain QB SDK transaction requests. The QB SD

K SalesReceiptRq, ReceivePaymentRq, and ARRefundCreditCardRq requests all accept QBMS transaction data. For more information, see Chapter 4, "Supporting QuickBooks Reconcile."

Intuit Confidential (c) 2008 Intuit Inc. All rights reserved.

QBMS and QuickBooks Online

Desktop and hosted applications can integrate with QuickBooks Online (QBO). However, QBO does not currently support the Reconciliation feature.

What Do I Need to Do to Integrate with QBMS?

The following checklist describes everything that you must do:

1. Decide on the security model that your application will use, desktop or hosted). For a description of these models, see "What is a Security Model and How Do I Choose One?.")
2. Register your application with the developer application gateway, at *appreg.intuit.com*. For more details, see "Registering Your Application."
3. If you are using the hosted security model, your application needs a server certificate and a client certificate to present to QBMS, so you need to obtain these. For the server certificate, Appendix D provides a list of root Certificate Authorities that are known to work with QBMS. The client certificate you must obtain from IDN by issuing a certificate signing request (CSR) to IDN as described at *appreg.intuit.com.*
4. Sign up for a QBMS account for testing purposes. For details, see Chapter 5, "Signing Up for a PTC Test QBMS Merchant Account."
5. Implement communication with QBMS (see "Communicating with QBMS").
    a. If you choose the desktop security model, you'll need to implement communication with QBMS following the material in Chapter 8, "Accessing QBMS from Desktop Applications."
    b. If you choose the hosted security model, you'll need to implement communication with QBMS following the material in Chapter 9, "Accessing QBMS From Hosted Web Applications."
6. Test the transactions your application will be performing. For details, see Chapter 6, "Testing Credit Card Transactions."

What is a Security Model and How Do I Choose One?

For a quick fact sheet on security models, check out the security model page at the *QBMS Integration Center.*

In order to choose the right security model for your implementation, you'll need to know a bit of background information about the desktop model and the hosted security models.

Intuit Confidential (c) 2008 Intuit Inc. All rights reserved.

The Desktop Security Model

In the desktop security model, permission to access a QBMS account to carry out transactions is granted by the account owner by means of a *connection ticket*. This ticket is created by QBMS for the QBMS account owner, then is copied by the QBMS account owner and pasted into your application. Your application encrypts it and stores it for subsequent use in QBMS communication. This ticket is valid until the account owner cancels it.

In this model, *any application that has that connection ticket* can access the QBMS account for transactions, which is why your application must protect it by encrypting it. If you are implementing a desktop application that is not accessible from the Internet, this security model is sufficient.

In this model, additional security is provided if the QBMS account owner desires it. The account owner can create the connection ticket with session authentication, so the account owner has to log on to the owner's QBMS account at the start of every session. So even if the connection ticket were somehow stolen, no transactions could occur unless the QBMS account owner first logged on.

Using Desktop Security with Applications Accessible via Internet

If you are implementing an application that *is* accessible from the Internet, you can still use the desktop security model if you take the following precautions:

1. Does your application support credit card refund or void transactions? If it does
    a. For refund or void transactions, use only connection tickets that have session authentication. This will require the QBMS account owner to logon at the start of every session.
    b. Alternatively, consider the hosted security model.

The Hosted Security Model

The hosted security model is the one used by hosted web applications. Like the desktop security model, permission to access a QBMS account to carry out transactions is granted by the account owner by means of the connection ticket. However, unlike the desktop model, there is no copy and paste of the connection ticket: the ticket is simply POSTed from QBMS to the hosted application. Also, in the hosted security model, connection tickets cannot have session authentication.

An additional level of security is provided in the hosted model through the use of certificates. Hosted web applications must have a server certificate in order to handle QBMS callbacks and they must also have a client certificate to POST transaction requests to QBMS. (A list of root certificate authorities that are known to work with QBMS are provided in Appendix D.)

Intuit Confidential (c) 2008 Intuit Inc. All rights reserved.

The hosted security model can be used for most types of applications but is required for hosted applications that are accessible over the Internet and that support multiple merchants. Also, you might want to consider this model if your application is an Internet-accessible application that supports refund and void transactions.

Registering Your Application

For the best, and up-to-date information on application registration, see the *Register Your Application* page at the QBMS Integration Center.

All applications are required to register with IDN before they can access QBMS. If you don't register, the Intuit gateways won't let your application into the QBMS data centers.

To register, visit *appreg.intuit.com*. (Additional registration instructions are located at the *IDN developer website*.)

Keep in mind there are two separate environments that you need to register for:

- Register your application with IDN to use the PTC test environment, to test your application.
- Register your application with IDN to use the QBMS production environment when you're ready to test live or go live.

IMPORTANT

After registering for PTC, you will be given an AppID that you use only to communicate with PTC. After registering for production, you will be given a *different* AppID that you use only to communicate with production QBMS. The AppID for the PTC test environment will not work with QBMS production and vice versa!

Communicating with QBMS

Figure 1-1 provides an overview of the things you need to do to communicate with QBMS. Some of them are one time tasks, such as getting certificates (hosted web applications) and registering your application. Some tasks, like getting a connection ticket, you need to get do once, but perhaps more than once if the merchant cancels the ticket. Other tasks, such as presenting certificates (hosted applications) and POSTing requests, are ongoing tasks that your application does automatically.

Intuit Confidential (c) 2008 Intuit Inc. All rights reserved.

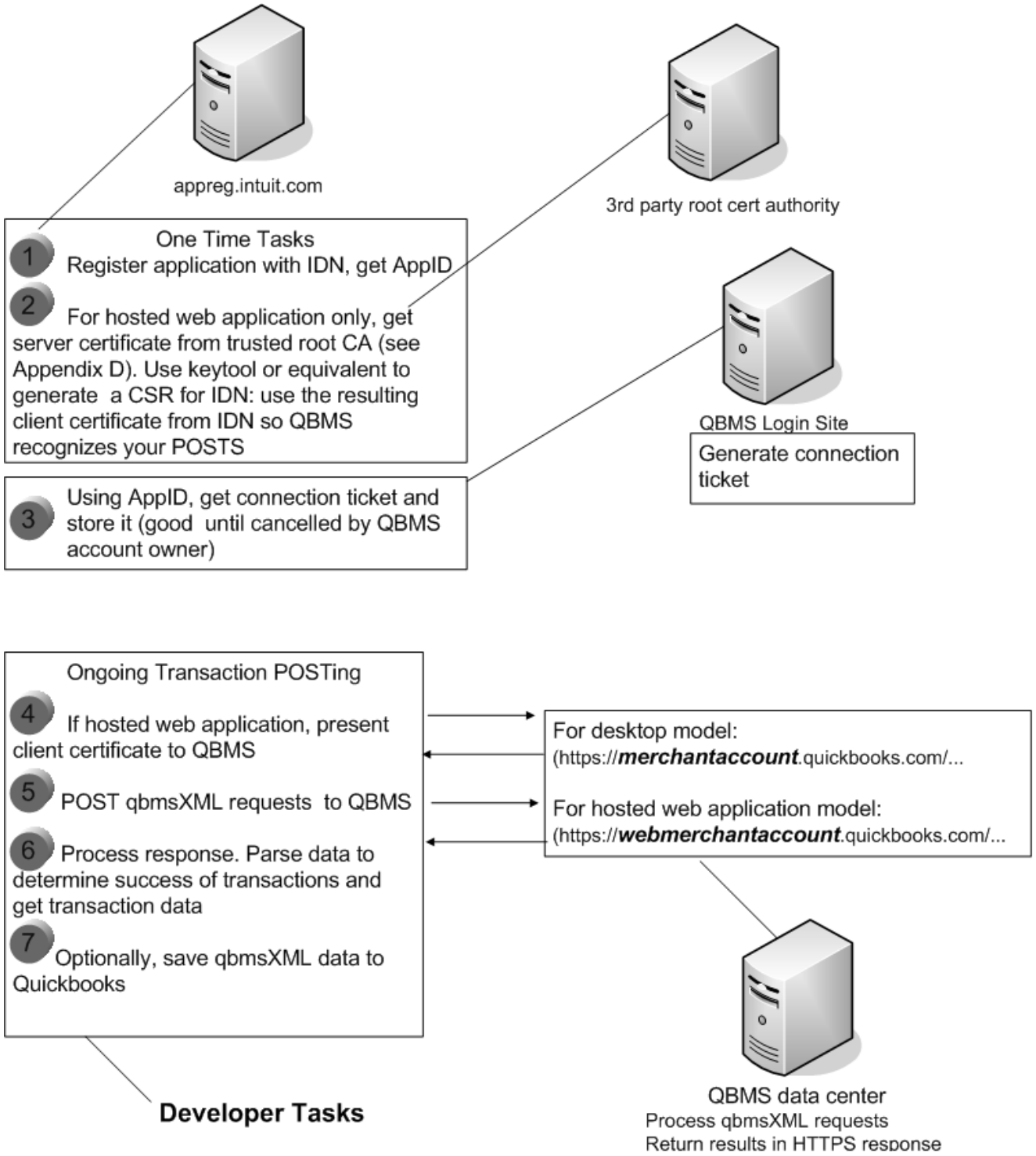

Figure 1-1 What needs to happen to communicate with QBMS

In Figure 1-1, notice that these tasks are the same for both production QBMS and for the PTC test environment: the only difference will be the URLs that are used, and the AppIDs

Intuit Confidential (c) 2008 Intuit Inc. All rights reserved.

will be different. (However, you can use the same valid server certificate and client certificate.) For details on the URLs used for product and PTC, see Chapter 8, “Accessing QBMS from Desktop Applications,” and Chapter 9, “Accessing QBMS From Hosted Web Applications.”

More About the Connection Ticket Task

As shown in Figure 1-1, your application needs to be granted permission to transact on behalf of a given merchant. This permission is represented by a connection ticket which is provided to your application as a result of the merchant executing a connection wizard hosted by QuickBooks Merchant Services. Upon completion of that wizard the connection ticket is POSTed back to a hosted web application or cut and pasted by the merchant into a desktop application. Your application need only open a browser to a specific URL for the merchant to complete the wizard. How to do this for desktop applications is shown in Chapter 8, “Accessing QBMS from Desktop Applications,” and how to do it for hosted applications is shown in Chapter 9, “Accessing QBMS From Hosted Web Applications.”

More About the Certificates Task

When you POST qbmsXML requests to our servers over HTTPS from a hosted web application, it must present a client certificate signed by Intuit. This is described in Chapter 9, “Accessing QBMS From Hosted Web Applications.”

Parsing the qbmsXML Response

After you POST, you’ll need to parse the response qbmsXML to determine if the transaction was successful and to ensure that the fraud-prevention checks (i.e. zip code and/or Card Security Code verification) meet the standards set by your application and/or the merchant's preferences.

In addition, for QBMS merchants who use QuickBooks, your application should:

a. Parse the response qbmsXML to obtain the credit card transaction result data
b. Send the credit card transaction result data to QuickBooks as part of a Sales Receipt, Payment Receipt, or Credit Card Refund to support QuickBooks' ability to check funding status and reconcile credit card transactions, including transaction fees, etc

What Am I Legally Required to do to Protect Financial Data?

Applications that access cardholder information using the SDK are required to follow the payment application data security standard (PA DSS) standard established by the payment card industry, which specifies how cardholder data must be protected. For details on this requirement, please refer to the Payment Card Industry (PCI) Security Standards Council website *https://www.pcisecuritystandards.org/*.

Intuit Confidential (c) 2008 Intuit Inc. All rights reserved.

However, there are a few items we'd especially like to draw attention to. Notice that you cannot store card security code (CVC2, CVV2, etc) data, and you cannot store Track2 data. The QBMS transaction data brought into QuickBooks must mask the credit card number: it should all be lowercase x except for the last four digits, with no dashes.

What Do My Merchant/Customers Need?

Before your merchant/customers can use your application, they must have a valid QuickBooks Merchant Service account. They can obtain an account online from *QBMS* or by phone.

NOTE

A web link will be provided for third-party developer so they can automatically direct their customers/merchants to QBMS web sites to perform an online signup.

The Merchant's QBMS Account Must be Set Up for eCommerce

To accept SDK-based transactions, when the merchant applies for a QBMS account, the merchant must check the "eCommerce or compatible third-party software" box on the application form. (If the merchant already has a QBMS account but is not yet set up for eCommerce, the merchant can upgrade the account.)

Also, when a merchant applies for a QBMS account, by default the account is configured to accept Visa, Discover, and MasterCard. To accept American Express the merchant must explicitly request this on their online (or phone-based) application. If the merchant has an existing account with American Express, QBMS Customer Service can link those accounts with their QBMS account.

Supported QuickBooks and qbXML Versions

If your application integrates QBMS transaction data with QuickBooks, the merchant must also have a version of QuickBooks that supports the QBMS SDK. We strongly recommend the use of qbXML specification 6.0 and greater and QuickBooks 2008 and Enterprise version 8 or greater, because these support the latest PCI/PA DSS standards required by card payment processors.

See the QBMS release notes for a full list of supported versions.

Intuit Confidential (c) 2008 Intuit Inc. All rights reserved.

Accessing Remote QuickBooks from QBMS Web Applications

If you want your hosted web application to access a QuickBooks company at a remote location, say at a customer/merchant's system, you can do so using the QuickBooks Web Connector. To see how to do this, please refer to the *QuickBooks Web Connector Programmer's Guide* included with the QB SDK.

Integrating a QBMS Application with QuickBooks Point of Sale

It is possible integrate an application both with QB POS and with QBMS. One typical way that this is used is that in QBPOS, merchants use two key transactions for sales - Sales Orders (most commonly used for online transactions) that are fulfilled later in the store by the retailer, and Sales Receipts (immediate fulfillment). In either case, we recommend use of QBMS for authorizing & capturing the credit card funds after fulfillment.

Where to Go for the Latest QBMS/qbmsXML Information

QBMS may provide or change features between releases of the SDK package. To find out about these changes, which are usually enhancements and improvements, go to the *IDN Forums* at the IDN website.

Intuit Confidential (c) 2008 Intuit Inc. All rights reserved.

Intuit Confidential (c) 2008 Intuit Inc. All rights reserved.

CHAPTER 2
Fraud Prevention Features

QuickBooks Merchant Service has fraud prevention features to allow merchants to set preferences for address verification (AVS) and for card security code checks. See “Fraud Prevention Preference Settings” for the default settings and the other settings available for the merchant.

To support the fraud prevention features, QBMS provides a web-based tool called the *Merchant Service Center* (MSC). This tool allows QuickBooks Merchant Service merchant/customers to

- Run reports & queries on their credit card activity
- Manage their fraud prevention settings.

There is also a *PTC test version* of the Merchant Service Center.

How Do Developers Use the Fraud Prevention Features?

To make use of the fraud prevention features and enable your merchants to make use of them, you must inform your users about Merchant Service Center and give them access to this tool from within your application.

If you don’t want to implement AVS & card security code fraud checks in your application, but want to provide the facility to your customer/merchants, simply point them to the *merchant service center* where they can configure fraud prevention settings themselves.

If you do implement AVS & card security fraud checks in your application, you may want to disable them and let the merchant use the Merchant Service Center tool directly.

A third alternative is to set up your application so that it does its own AVS/CSC verification. In this case, QBMS won’t decline a transaction if the AVS check fails, since your application is performing this check. (Of course, even in this scenario, if the merchant sets their fraud settings via the Merchant Service Center to decline such transactions, their fraud settings will override your application’s handling of this and the transaction will be declined.) If you want your application to do its own AVS/CSC checking, you must set up your application with QBMS in advance to do this. For details, see *Registering Applications* at the IDN website.

Fraud Prevention Preference Settings

The following tables list the settings available for the AVS and CVS verification.

(c) 2008 Intuit Inc. All rights reserved.

Table 2-1 Address Verification Service (AVS) preference settings

AVS Check Response	Options	Options Default
If neither Street Address nor Zip Code match	Accept/Reject Transaction	Reject
If Zip Code matches but Street Address does not match	Accept/Reject Transaction	Accept
If Street Address matches but Zip does not match	Accept/Reject Transaction	Accept
If Street Address and Zip Code are not available. **Note:** if the application does not send the AVS data in the XML, it is considered under the same case as AVS authorization system unavailable.	Accept/Reject Transaction	Accept

IMPORTANT

If a merchant wants to accept all transactions regardless of AVS check, they should mark all the above as Accept.

Table 2-2 Card Security Code preference settings

Card Security Check Response	Options	Options Default
If card security code does not match	Accept/Reject Transaction	reject
If card security code is not available Note: If the application does not send the card security code data in the XML, it is considered the same case as card security code checking not supported or processed.	Accept/Reject Transaction	Accept

Notes on Using AVS Features

You should be aware that QBMS successfully completes a transaction even if an AVS check fails or is unavailable. It is your responsibility to handle AVS failures in your code. Such orders should be flagged and merchant should be able to identify them and take corrective actions such as voiding the transaction.

Notice also that international addresses will result in an AVS check failure in QBMS.

(c) 2008 Intuit Inc. All rights reserved.

CHAPTER 3
Running Credit Card Transactions

This chapter describes the credit card transactions you can send to QBMS. It overviews some the basic things you'll need to know about sending transactions in general, and then briefly mentions each transaction type in detail.

What is the QBMS API?

The QBMS API has no code libraries and no runtime. It consists of an XML spec called qbmsXML. You write your transaction requests following the qbmsXML spec for each type of transaction. Listing 3-1 shows what one of these look like for a credit card authorization:

_____Listing 3-1 What qbmsXML looks like: sample authorization request

```
<?xml version="1.0" ?>
<?qbmsxml version="2.0"?>
<QBMSXML>
   <QBMSXMLMsgsRq>
      <CustomerCreditCardAuthRq>
         <TransRequestID>E09C86CF-9D6E-4EF2-BCBE-4D66B6B0F754<TransRequestID>
         <CreditCardNumber>4111111111111111</CreditCardNumber>
         <ExpirationMonth>12</ExpirationMonth>
         <ExpirationYear>2008</ExpirationYear>
         <IsECommerce>true</IsECommerce>
         <Amount>203.00</Amount>
         <CreditCardAddress>23 Garcia Ave</CreditCardAddress>
         <CreditCardPostalCode>94043</CreditCardPostalCode>
      </CustomerCreditCardAuthRq>
   </QBMSXMLMsgsRq>
</QBMSXML>
```

Don't worry about the details at this point. Just notice that this is all there is to the coding part, at least in the building of the transaction request. You do need to handle certain errors returned in the response, which is covered in the chapter on error handling.

This is what a typical response to a request looks like:

(c) 2008 Intuit Inc. All rights reserved.

```
  <QBMSXMLMsgsRs>
   <CustomerCreditCardAuthRs requestID="23909" statusCode="0" statusSeverity="INFO" statusMessage="Status OK"
    <CreditCardTransID>ME3365118336</CreditCardTransID>
    <AuthorizationCode>336511</AuthorizationCode>
    <AVSStreet>Pass</AVSStreet>
    <AVSZip>Pass</AVSZip>
    <CardSecurityCodeMatch>NotAvailable</CardSecurityCodeMatch>
    <ClientTransID>q0002ec9</ClientTransID>
   </CustomerCreditCardAuthRs>
  </QBMSXMLMsgsRs>
</QBMSXML>
```

Figure 3-1 Sample transaction response

This sample request and response represent the transaction request/response; to keep things simple, we've omitted the SignonMsgsRq aggregate, which we've already mentioned.

We'll cover this in a lot more detail later on. For right now, you just need to be aware of this aspect of the request-level work you'll be doing when you integrate your application with QuickBooks Merchant Service.

Where Can I Find the Full Syntax Details on the QBMS API?

Included with the SDK is an onscreen reference, which we call the OSR for short. The OSR is online at *http://developer.intuit.com/qbsdk-current/newOSR/index.html* . Various XML specs are documented in the OSR, so make sure you choose qbmsXML.

Each supported credit card transaction is listed in the OSR, along with explanations for each of the fields, and other information such as whether a field is mandatory, or can be used if some other field is used (ORs, ANDs, and so forth).

Sending Multiple Transaction Requests in a Single POST

You can batch up a maximum of 20 transaction requests in a single post to QBMS. When you do this, be careful how you handle the responses. They might not be in the same order as the requests, so in this case you should use the requestID attribute in the request and check the requestID attribute in the response to match these up. This same consideration applies to the QB SDK as well, although QB SDK frequently returns the responses in the same order. That cannot be expected from the QBMS SDK.

Notes on Running Transactions

The logic you need to follow when running transactions is fairly straightforward. To make a credit card charge for example, you need to build the qbmsXML request CustomerCreditCardChargeRq, as described in the OSR, filling in all required customer information. Then you must post this request to QBMS as described in the chapters on

(c) 2008 Intuit Inc. All rights reserved.

accessing QBMS from desktop applications and from hosted web applications. See Chapter 8, “Accessing QBMS from Desktop Applications,” and Chapter 9, “Accessing QBMS From Hosted Web Applications.”

You must check the response for success and optionally save the transaction data, optionally into QuickBooks via the ReceivePaymentAddRq, SalesReceiptAddRq, or ARRefundCreditCardAddRq requests in the QB SDK.

To get an authorization for a future transaction, for example when taking a customer order but prior to fulfilling it, you must build and post the CustomerCreditCardAuthRq request, check the response for success, and save the response data, particularly the transaction ID, as you will need the transactionID in order to perform a capture, which charges the credit card that has previously been authorized for the charge.

.If authorization is denied you’ll need to get a voice authorization. (See “Credit Card Voice Authorizations.”)

To capture a previously authorized transaction request, you need to build and post a CustomerCreditCardCapture request, using the transactionID obtained from the previous CustomerCreditCardAuthRq request.

To void a transaction you need to build and post a CustomerCreditCardTxnVoidRq request, using the transaction ID of the transaction to be voided. This request is successful only if the transaction has not yet been settled by the card issuer.

If the transaction has already been settled, you can build and post a CustomerCreditCardRefundRq request to refund an amount to the customer’s credit card account.

Card Swipe and Card Present Transactions

Card swipe transactions must use the Track2Data field in the transaction request rather than the IsCardPresent boolean. Only card swipe transactions get the discount rate, not card present transactions. Card present is used for transactions where the card is presented to the merchant but a card swipe transaction couldn’t be performed for some reason.

Notice that in the case of a voice auth transaction where the card was swiped (and therefore the transaction contains Track2Data) the transaction won’t qualify for the discount rate.

For a sample application supporting card swipe (Track2Data), see the RequestGenerator sample in the SDK sample subdirectory \samples.

Format of Track2Data

Track2 data is obtained from a card swipe, containing card number, card expiration date, and other optional data in an expected format, which is shown below.
Important! You must not store Track2 data. Doing so violates the requirements of the card processors as expressed in the Payment Application Data Security Standard (PA DSS).

(c) 2008 Intuit Inc. All rights reserved.

The Track2 data must be a minimum of 23 characters, 39 max. This is the format:

- The first character must be the start sentinel character ;
- The credit card number follows and is separated by the separator character =
- The card expiration date follows and any other data, with the termination character of ?

Example:

```
;372449635312118=1202101123456789?
```

where the first group of characters is the card number, and 1202 is the card expiration date in the format YYMM, which means our sample 1202 means February 2012.

(c) 2008 Intuit Inc. All rights reserved.

Credit Card Authorizations

CustomerCreditCardAuth is used for a transaction in which the merchant needs authorization of a charge, but does not wish to actually make the charge at this point in time. For example, if a customer orders merchandise to be shipped, you could issue this request at the time of the order to make sure the merchandise will be paid for by the card issuer. Then at the time of actual merchandise shipment, you perform the actual charge using the request CustomerCreditCardCaptureRq.

It is very important to save the CreditCardTransID from the response to this request, because this is required for the subsequent CustomerCreditCardCapture request. Notice that currently Auth transaction data cannot be stored in QuickBooks.

NOTE

The authorization is valid only for a fixed amount of time, which may vary by card issuer, but which is usually several days. QBMS imposes its own maximum of 30 days after the date of the original authorization, but most issuers are expected to have a validity period significantly less than this.

(c) 2008 Intuit Inc. All rights reserved.

Credit Card Capture

This request can be made only after a previous and successful CustomerCreditCardAuth request, where the card issuer has authorized a charge to be made against the specified credit card in the future. The CreditCardTransID from that prior transaction must be used in this subsequent and related transaction. This request actually causes that authorized charge to be incurred against the customer's credit card.

Notice that you cannot have multiple capture requests against a single CustomerCreditCardAuth request. Each CustomerCreditCardAuth request must have one and only one capture request.

The elements in the response are supplied in the CreditCardTxnResultInfo aggregate when you save the transaction to QuickBooks: most of these are used by QuickBooks internally for the Reconcile feature and the Get Funding Status feature. See the OSR for field descriptions.

(c) 2008 Intuit Inc. All rights reserved.

Credit Card Charge

This request, if successful, causes a charge to be incurred by the specified credit card. Notice that the authorization for the charge is obtained when the card issuer receives this request. The resulting authorization code is returned in the response to this request.

Notice that voice authorizations cannot be handled by this request. For voice authorizations, use the CustomerCreditCardVoiceAuth request.

The elements in the response are supplied in the CreditCardTxnResultInfo aggregate when you save the transaction to QuickBooks: most of these are used by QuickBooks internally for the Reconcile feature and the Get Funding Status feature. See the OSR for field descriptions

(c) 2008 Intuit Inc. All rights reserved.

Credit Card Refunds

Starting with qbmsXML spec 2.1, there are two ways to get a refund:

- CustomerCreditCardTxnVoidOrRefundRq, which is the preferred way for getting a refund. QBMS figures out whether to request a void or a refund based on factors such as the time of the original transaction and the time of the CustomerCreditCardTxnVoidOrRefund transaction. The void or refund is tied to an actual sale or capture transaction.
- CustomerCreditCardRefundRq, which you would use to explicitly request a refund that was not tied to any sale or capture transaction.

NOTE

A refund transaction may involve some processing fees from the card issuer, depending on the bank or credit card processor that is used. In comparison to refund transactions (for those cases where fees are charged for a transaction void), the fees for a void are normally lower. However, a refund restores a card's credit limit faster than a void does (approximately 24 hours versus a few days).

CustomerCreditCardTxnVoidOrRefund

The CustomerCreditCardVoidOrRefund request uses the CreditCardTransID of the original transaction to request either a void or a refund of that particular transaction. QBMS figures out whether to request a void or a refund based on various factors such as the current time and the time of the original transaction.

You can optionally supply a refund amount. If the request is processed as a refund, the amount is checked to make sure it doesn't exceed the amount of the original transaction: the amount doesn't have to be equal to the original amount--you can do partial refunds.

NOTE

CustomerCreditCardVoidOrRefund provides an alternative to the CustomerCreditCardRefund request. It does not replace it: you can use whichever of these best meets your needs.

You can only use the CustomerCreditCardVoidOrRefund for a previous successful sale transaction or delayed capture transaction that has not already been voided. (Using this request on a transaction voided earlier will generate an error.)

You can optionally set the ForceRefund field in the CustomerCreditCardVoidOrRefund to true to make sure a refund is given rather than a void. Forcing a refund must be done only when necessary, for example, if there is a need to restore a customer's credit card limit quickly. (Notice that you cannot set ForceRefund to false, as this will generate an error.)

(c) 2008 Intuit Inc. All rights reserved.

If the response to a CustomerCreditCardVoidOrRefund request contains the VoidOrRefundTxnType "Refund", then you can save the transaction in QuickBooks using the QB SDK request ARRefundCreditCardAdd. To save the transaction you need to supply the last 4 digits of the credit card number, and expiration month and year of the card used in the original sales or delayed capture transaction.

If the response to a CustomerCreditCardVoidOrRefund request contains the VoidOrRefundTxnType "Void", then you do not save any data into QuickBooks.

CustomerCreditCardRefund

You use CustomerCreditCardRefundRq to explicitly request a refund, supplying the credit card number to be refunded. Notice that there is no reference to any CreditCardTransID in the CustomerCreditCardRefund request! This means that there will be no checking by the card processor to determine whether the refund should be issued. It is the responsibility of the merchant and of the integrated application to keep track of which customers should be refunded and the amount of the refund.

To refund an amount to a customer’s credit card using CustomerCreditCardRefundRq, you must

1. Build a CustomerCreditCardRefund request, supplying the credit card information and the amount to be credited.
2. Post the request to QBMS.
3. Check for success
4. Save the transaction data.

Using the response data obtained from the CustomerCreditCardRefund request, you can optionally save the refund transaction data into QuickBooks via the request ARRefundCreditCardAddRq.

Credit Card Voids

The CustomerCreditCardTxnVoid request cancels a credit card charge request that was successfully made, but which has not yet been settled (that is, funded by the card issuer to the merchant's bank account). Refund transactions can also be voided.

Normally, transactions are funded within 2 to 3 business days. A transaction that is voided in this way incurs no transaction fees for authorizations, but fees may still be assessed for charges. You must supply the valid CreditCardTransID that was returned in a previously successful CustomerCreditCardCharge, CustomerCreditCardAuth, or CustomerCreditCardCapture request.

NOTE

Even where fees may apply to a transaction void, the fees for a void are normally lower than those for refunds.

(c) 2008 Intuit Inc. All rights reserved.

This request can also be used to void an authorization or the subsequent capture transaction of an authorization if used prior to any settlement. This prevents settlement of the transaction but notice that it does not release the authorization itself (which is a hold on funds). The hold on funds typically lasts for about a week. If you want to remove the hold before it expires, you should be aware that the hold on funds cannot be removed via payment gateways but instead must be done manually by contacting the issuing bank.

Notice that data from this transaction is not saved into QuickBooks.

Credit Card Voice Authorizations

You may need to obtain a voice authorization during a credit card charge operation or during a credit card authorization operation. For example, an attempted charge or authorization request may be denied by the card issuer and a voice authorization is then subsequently attempted.

To charge the customer's card using a voice authorization, the merchant must call the card issuer and get the voice authorization code. Your application must then build and post a CustomerCreditCardVoiceAuthRq request with the AuthorizationCode field filled in with the voice authorization code obtained from the card issuer.

Using Track2Data in Voice Auth

When Track2Data is used in a voice auth request, QBMS does NOT pass the track2Data to processing gateways. Instead it uses the track2Data only to parse the credit card number that is sent to processing gateways. In other words, this is NOT a transaction that qualifies for the discount rate.

(c) 2008 Intuit Inc. All rights reserved.

Merchant Account Queries

The MerchantAccountQuery is used to query information about the current merchant account . The query returns with the credit card types (Visa, MasterCard, Discover, AmericanExpress, JCB, DinersClub) that the merchant account accepts. (That is, there will be a separate CreditCardType element in the response for each supported card.) If the merchant account has a set convenience fee value, the ConvenienceFees element will also be returned in the response.

NOTE

Convenience fee based accounts are accounts that charge customers a fixed fee per transaction regardless of the size of the transaction, for the convenience of using a credit card.

If the merchant account cannot be identified or is not subscribed to QBMS, the status code 10202 and the status message "An error occurred during account validation" are returned.

Lodging Transactions

Card associations and networks require lodging merchants to send a set of fields along with authorization and settlement requests. These fields identify these transactions as originating from lodging merchants and possibly qualifying for special pricing in some cases.

To support the lodging merchants, a new lodging aggregate has been added to all authorization, refunds, void, sale and capture requests beginning with qbmsXML 3.0. In addition, a new transaction type called incremental authorization added in qbmsXML 3.0 is targeted specifically for lodging merchants. This incremental auth request allows the merchants to add to an existing authorization when a guest extends his/her stay.

For more details, see the *Lodging implementation guide* at the QBMS Integration Center website.

Restaurant Transactions

Similar to the Lodging aggregate, the Restaurant aggregate is also added in qbmsXML 3.0 to provide certain fields required by restaurant merchants.

For more details, see the *Restaurant implementation guide* at the QBMS Integration Center website.

(c) 2008 Intuit Inc. All rights reserved.

Merchant-Initiated Batch Close Transactions

Made available in QBMS SDK 3.0, the QBMS batch close feature allows merchants to close *one or more* open batches during the business day when the merchant wants to close the batches. The BatchID field is available in the various transaction requests if you specify the use of qbmsXML spec level 3.0 or greater when you make your transaction requests.

For complete information on using the batch close feature in QBMS, see the *Batch Close Center implementation guide* at the QBMS Integration website.

(c) 2008 Intuit Inc. All rights reserved.

CHAPTER 4
SUPPORTING QUICKBOOKS RECONCILE

This chapter describes the Reconcile and Get Funding Status feature of QuickBooks and QBMS.

What is the Reconcile Feature and Why is it Needed?

The QuickBooks with QBMS Reconcile feature is designed to solve the problem of keeping the merchant's QuickBooks company data in sync with the merchant's actual bank account data. There are two areas where these can get out of sync:

- Credit card funding of transactions occurring in the real bank account are not reflected in QuickBooks until the merchant performs a Deposit to record the deposit of the transaction funds in the QuickBooks bank account.
- Credit card transaction fees taken out of the merchant's real bank account are not reflected in the QuickBooks company until the merchant updates QuickBooks with those assessed fees. These fees are charged on a daily as well as monthly basis and consist of fees such as transaction fees, discount fees, downgrades, monthly fees and chargebacks.

To help sync up in the first area (Get Funding Status), QuickBooks with QBMS provides Funding Status information for each batch of deposits inside the QuickBooks Undeposited Funds account. The Get Funding Status button gets the current status of the selected transactions from QBMS. If the transactions are funded at the merchant's bank, the merchant knows that the transactions need to be moved from QuickBooks' Undeposited Funds into the appropriate bank account set up to receive those funds.

To help sync up the QuickBooks company in the second area (transaction fees), QuickBooks with QBMS enables the merchant to download the fees for VISA, Discover, and Mastercard that were posted to the bank account. This permits accurate matching of the fees with the actual bank debits. For American Express, since the funds are deposited net of fees, QB lets the merchant compute the fees through a fees calculator while depositing the funded batch of transactions in QuickBooks.

How Do the QBMS and QB SDKs Support the Reconcile Feature?

In the scenarios described above, the credit card transaction synching activities that need to take place result from credit card transactions that are carried out within QuickBooks with QBMS. Because the transactions are performed inside QuickBooks, the transaction data is already in QuickBooks and all that must be done is the synching-up in QuickBooks using the Reconcile feature.

(c) 2008 Intuit Inc. All rights reserved.

But what happens when you have a third party application (integrated with the QBMS SDK) performing the credit card transactions? How does that transaction data get put into QuickBooks? That transaction data must be saved by the third party application into QuickBooks via the QB SDK requests ReceivePaymentAdd, SalesReceiptAdd, or ARRefundCreditCardAdd. (The response data from the QBMS transaction requests contain data that must be included in those requests.)

IMPORTANT

The aggregate containing the supplied QBMS transaction data must mask the credit card number with lower case x and no dashes. For example, xxxxxxxxxxxx1234.

Notice that once the transaction data is automatically saved into QuickBooks, the merchant must still manually perform the sync operations (making QB deposits when the real Bank account is funded, loading any transaction fees into QuickBooks). Those activities are NOT automated by the QBMS SDK or by the QB SDK.

IMPORTANT

You must be careful when bringing data from QBMS credit card transactions into a QuickBooks company via the SalesReceiptAdd, ReceivePaymentAdd, or ARRefunCreditCardAdd. The QB SDK has no way of knowing which company file should get that information, so if you are logged into the wrong company file, the wrong company file will get that data. One way to implement a check on whether the company is the expected one is to use a data extension on the QuickBooks company.

Saving the Transaction Data Into QuickBooks

The ReceivePaymentAdd, SalesReceiptAdd, and ARRefundCreditCardAdd requests accept credit card data originating from QBMS transactions, if the current QuickBooks company is set up to use QBMS and has a valid QBMS account.

Make Sure the CustomerRef and PaymentRef Match the Transaction

IF you build a ReceivePaymentAdd, SalesReceiptAdd, or ARRefundCreditCardAdd request that contains QBMS transaction data, you are responsible for making sure the credit card transaction is mapped to the proper CustomerRef. Also, you must make sure you specify the proper credit card type in the PaymentMethod Ref. This information is NOT included in the credit card aggregate data.

If you don’t specify the credit card type in the PaymentMethodRef, the ReceivePaymentAdd, SalesReceiptAdd, or ARRefundCreditCardAdd request will fail.

(c) 2008 Intuit Inc. All rights reserved.

Where to Find the Transaction Data You Need

If you are including QBMS credit card data, both the ReceivePaymentAdd and SalesReceiptAdd requests require the same parent aggregate CreditCardTxnInfo and the same child aggregates CreditCardTxnInputInfo and CreditCardTxnResultInfo, as shown in the OSR. All of the data in these aggregates must be obtained from the qbmsXML credit card requests for CreditCardTxnInputInfo and responses for CreditCardTxnResultInfo.

However, there are two items in the CreditCardTxnResultInfo that might be slightly tricky. The ResultCode and the ResultMessage are the StatusCode and StatusMessage returned as attributes in the qbmsXML responses.

Sample qbXML

The following sample XML shows a SalesReceiptAdd request containing QBMS credit card data. The transaction is for a customer named John Hamilton. Notice the payment ref is set to Credit Card: if you don't have this, the request will fail. Notice that the aggregate CreditCardTxnResultInfo contains a ResultCode of 0, which means the original qbmsXML transaction request was successful.

Notice the processing instruction qbxml version=”6.0”. We specify that because the request is using qbmsXML 2.0 features, which you should do if qbXML 6.0 and qbmsXML 2.0 are available.

```
<?xml version="1.0" ?>
<?qbxml version="6.0"?>
<QBXML>
   <QBXMLMsgsRq onError="stopOnError">
      <SalesReceiptAddRq requestID = "101">
         <SalesReceiptAdd>
            <CustomerRef>
               <FullName>John Hamilton</FullName>
            </CustomerRef>
            <TxnDate>2005-02-23</TxnDate>
            <RefNumber>2345</RefNumber>
            <PaymentMethodRef>
               <FullName>Visa</FullName>
            </PaymentMethodRef>
            <Memo>QBMS SDK Test 2345</Memo>
        <CreditCardTxnInfo>
            <CreditCardTxnInputInfo>
               <CreditCardNumber>xxxxxxxxxxxx4444</CreditCardNumber>
               <ExpirationMonth>12</ExpirationMonth>
               <ExpirationYear>2010</ExpirationYear>
               <NameOnCard>John Hamilton</NameOnCard>
               <CreditCardAddress>2750 Coast Avenue</CreditCardAddress>
               <CreditCardPostalCode>94043</CreditCardPostalCode>
               <CommercialCardCode>Doe123</CommercialCardCode>
```


(c) 2008 Intuit Inc. All rights reserved.

```
            <TransactionMode>CardNotPresent</TransactionMode>
         </CreditCardTxnInputInfo>
         <CreditCardTxnResultInfo>
            <ResultCode>0</ResultCode>
            <ResultMessage>STATUS OK</ResultMessage>
            <CreditCardTransID>V64A76208243</CreditCardTransID>
            <MerchantAccountNumber>4269281420247209</MerchantAccountNumber>
            <AuthorizationCode>185PNI</AuthorizationCode>
            <AVSStreet>Pass</AVSStreet>
            <AVSZip>Fail</AVSZip>
            <CardSecurityCodeMatch>Pass</CardSecurityCodeMatch>
            <ReconBatchID>420050223 MC 2005-02-23 QBMS 15.0 pre-beta</ReconBatchID>
            <PaymentGroupingCode>4</PaymentGroupingCode>
            <PaymentStatus>Completed</PaymentStatus>
            <TxnAuthorizationTime>2005-02-23T20:57:13</TxnAuthorizationTime>
            <TxnAuthorizationStamp>1109192233</TxnAuthorizationStamp>
            <ClientTransID>q0002ee5</ClientTransID>
         </CreditCardTxnResultInfo>
         </CreditCardTxnInfo>
         <SalesReceiptLineAdd>
            <ItemRef>
               <FullName>Fee</FullName>
            </ItemRef>
            <Rate>100.00</Rate>
         </SalesReceiptLineAdd>
         </SalesReceiptAdd>
         </SalesReceiptAddRq>
   </QBXMLMsgsRq>
</QBXML>
```


(c) 2008 Intuit Inc. All rights reserved.

CHAPTER 5
SIGNING UP FOR A PTC TEST QBMS MERCHANT ACCOUNT

There are three types of QBMS account that you must be aware of:

- The PTC test account described in this chapter. This is a test account in our PTC test environment that is available for no charge with very few restrictions (e.g., no performance testing is allowed). You should use this account for your development and testing. The transactions in this environment use test credit card numbers and run against a QBMS emulator. To apply and setup, follow the instructions in this chapter.
- Alternatively, you can obtain a restricted but real QBMS account that can be used to run valid credit cards and test end to end capability in production. QBMS waives monthly, annual and setup fees however, the transaction fees will apply. Use this account to test your application in production prior to deployment. Please go to the *QBMS Integration Center website* for more information and to apply.
- A real production QBMS account that is used to run normal business transactions with real credit cards. Normal account fees apply. Use this account to run your normal business transactions. You can also use this account to verify your application in production prior to deployment. To apply, go to the *QuickBooks Merchant Service* website.

NOTE

When you (or your merchants) apply for a real, full production QBMS account, the account accepts Visa, Discover, and Mastercard by default. To make the account also accept American Express, you must explicitly request this on their online (or phone-based) application. Also, if you have an existing account with American Express, QBMS Customer Service can link that accounts with the QBMS account as well. Finally, in order to get SDK-based transactions accepted by the QBMS account, you (or your merchants) must check the "eCommerce or compatible third-party software" box on the application form.

Signing Up For a PTC Test Account

To set up your QBMS test account if you are not integrating with QuickBooks,

1. Go to the QBMS Developer Program website, and in the Integration Center page click on Get a Test Account to bring up the Test Account Page.
2. Follow the prompts to obtain your test account. These are self explanatory. However, you'll need to know a few things that we'll cover here:

(c) 2008 Intuit Inc. All rights reserved.

When prompted to supply a valid email address, make sure you supply a valid address that has not been used before in any of QB Online or QBMS test or production environments. (Ignore any fee information for the test environment: there are no fees in the test environment for test transactions.) That email address will be your login to the test account when you need to login for connection tickets or session tickets.

Accessing Your Test Account with QuickBooks (Optional)

IMPORTANT

You cannot use one of the sample QuickBooks company files. You should NOT use your production QuickBooks company file either because the same company file may not be usable in both the PTC test and QBMS production environments. Instead, create and use a new company file for your testing.

If you want to integrate QuickBooks with your QBMS application, to reflect QBMS transaction data back into your QuickBooks company, you'll need to set up your company file to access your QBMS PTC test account.

To set up QuickBooks to use your QBMS test account,

1. **Make sure QuickBooks is not running** and make sure you are connected to the Internet.
2. From the Windows Start menu, select Programs->IDN SDK->QBMS SDK->Tools->QBMS Use IdnBeta. This will cause QuickBooks to use the PTC test environment for the credit card processing signup and for all subsequent credit card transactions.
3. Start QuickBooks and create a new test company and assign as its email address the same email address you used when you obtained your test account. (Company->Company Information->E-mail).
4. From the Customers menu, select Receive Payments. The first time you do this in a new company file you are asked whether your business accepts credit cards. Respond by selecting "Yes." This will cause a QBMS modal dialog to be displayed showing a couple of choices.
5. In the QBMS dialog that is displayed, select the choice, "Activate an existing QBMS account" and click OK.
6. Click OK when prompted to launch the web browser.
7. The QBMS login page is displayed, prefilled with your company's email address. This must be the same address used to create your test QBMS account. Supply your QBMS test account password, click Log In, and wait a moment or two.

   If you have been using the QBMS test account already with one test company, you'll be prompted to transfer the test account or create a new test account. Choose whichever one of these options you want and proceed.
8. At this point, you should get the prompt indicating that your QBMS test account has been activated for this company file. On the form is a button labeled "Receive

(c) 2008 Intuit Inc. All rights reserved.

Payment." You can either proceed with the RecievePayment transaction or close the QBMS form, since your QB company file is now activated for QBMS in the PTC test environment.

Restoring QuickBooks to Point to the Live QBMS Environment

After you are finished developing and thoroughly testing your application, you need to test it in the production environment, which means you must change QuickBooks to point to the production QBMS environment, and you must then use your real company file to apply for a real merchant account from QBMS.

To change to the production environment, from the Windows Start menu, select Programs->Intuit SDKs->QBMS SDK X.X->Tools->QBMS Use Production.

As a result of this change, all QBMS signup links will send you to the live QBMS signup location. All subsequent credit card processing within QuickBooks is performed in the live environment and must use real credit card numbers and will incur transaction charges.

IMPORTANT

Each applicaton must be registered separately for the PTC test environment and for the live production environment. The appID value issued for the PTC environment will not work in the production environment, and vice versa.

(c) 2008 Intuit Inc. All rights reserved.

(c) 2008 Intuit Inc. All rights reserved.

CHAPTER 6
TESTING CREDIT CARD TRANSACTIONS

This chapter provides information on the kinds of testing you may need to perform on your application. There are two basic kinds of testing:

- Credit card transaction testing in the QBMS emulation environment
- Testing with QuickBooks

Testing Credit Card Transactions

When your application is using the QBMS test environment (.ptc), all of the transation requests are sent to the QBMS emulation backend. This emulator "processes" the requests and returns the appropriate responses. The word "process" is in quotes because there is no real processing going on. The card number is checked against the valid card numbers listed below (Table 6-1 on page 43), and the expiration date is checked against the current date.

Table 6-1 Valid Test Credit Card numbers

Card Type	Test Number	Number of Characters
Master Card	5105105105105100	(16)Characters
Master Card	5555555555554444	(16)Characters
VISA	4222222222222	(13)Characters
VISA	4111111111111111	(16)Characters
VISA	4012888888881881	(16)Characters
American Express	378282246310005	(15)Characters
American Express	371449635398431	(15)Characters
Amex Corporate	378734493671000	(15)Characters
Diners Club	38520000023237	(14)Characters
Diners Club	30569309025904	(14)Characters
Discover	6011111111111117	(16)Characters
Discover	6011000990139424	(16)Characters

IMPORTANT

Currently, there is no way to test voice authorization requests in the QBMS test environment.

However, the AVS values and CVS values are not checked, nor account limits, nor status, and so forth. To cause the appropriate errors to be returned in response XML, however, you can supply a configID value within the <NameOnCard> tag that will cause the error you want returned:

(c) 2008 Intuit Inc. All rights reserved.

```
<NameOnCard>configid=value </NameOnCard>
```

Simply replace “value in the above line with one of the ConfigID values listeds in Table 6-2 on page 44.

Table 6-2 ConfigID values and the errors they generate:

Error to be Returned	ConfigID value to insert	Error Emulated
10200	10200_comm	An error occurred while communicating with the credit card processing gateway.
10201	10201_login	An error occurred during login to the processing gateway.
10301	10301_ccinvalid	This credit card account number is invalid.
10400	10400_insufffunds	This account does not have sufficient funds to process this transaction.
10401	10401_decline	The request to process this transaction has been declined.
10403	10403_acctinvalid	The merchant account information submitted is not recognized.
10404	10404_referral	This transaction has been declined, but can be approved by obtaining a Voice Authorization code from the card issuer.
10405	10405_void	An error occurred while attempting to void this transaction.
10406	10406_capture	An error occurred while processing the capture transaction.
10500	10500_general	A general error occurred at the credit card processing gateway.
10000	10000_avscvdfail	Status OK, AVS Street and Zip fail, card security code fail
10000	Is default: supply no configID value.	Status OK, AVS Street and Zip pass, card security code pass (AVS and CSC fields supplied in the request)
10000	Is default: supply no configID value	Status OK, AVS Street and Zip unavailable, card security code unavailable (used only required fields in the xml request)

(c) 2008 Intuit Inc. All rights reserved.

For example, in the following request, a credit card charge transaction is requested, with the NameOnCard tag set to generate an insufficient funds error:

```
<QBMSXML>
<QBMSXMLMsgsRq>
   <CustomerCreditCardChargeRq>
      <TransRequestID>02B123451</TransRequestID>
      <CreditCardNumber>5555555555554444</CreditCardNumber>
      <ExpirationMonth>12</ExpirationMonth>
      <ExpirationYear>2008</ExpirationYear>
      <Amount>130.00</Amount>
      <NameOnCard>configid=10400_insufffunds</NameOnCard>
   </CustomerCreditCardChargeRq>
</QBMSXMLMsgsRq>
</QBMSXML>
```

Testing Track2 Data

Certain QBMS transactions support the use of Track2 data. The PTC test environment supports testing of this feature. There is a sample called RequestGenerator included with the SDK, in the \samples subdirectory, that you can use to do this.

If you don't want to use that sample, you can also supply the following value for the <Track2Data> element in your requests:

```
;372449635312118=1202101123456789?
```

Testing CustomerCreditCardTxnVoid

The PTC emulator considers all transactions settled. Void transactions cannot succeed on settled transactions: the error 10405 would be returned in this case. So You can use the PTC test environment to test this for the failure case. That is, you can issue this request and you should get error 10405 back in the response.

Testing With QuickBooks

In the test environment, certain QuickBooks/QBMS features can be tested, for example, Funding Status. However, the Reconciliation feature requires real data from financial institutions, and so cannot be tested.

What is it that you need to test? If you write QBMS credit card transaction data into QuickBooks using the QB SDK requests ARRefundCreditCardAdd, ReceivePaymentAdd, or SalesReceiptAdd, you can visually check the results by examining the Make Deposits form, which is accessed from the QuickBooks main form by selecting Banking->Make Deposits.

(c) 2008 Intuit Inc. All rights reserved.

This should bring up the Payments to Deposit form which lists the payments and also refunds. You can examine this form to determine whether your data is showing up correctly in QuickBooks. You can also select a payment and click the Get Funding Status button, which will, in the test environment, get the funding status from the emulator.

Testing and Diagnosing Web Apps

During development, if you are running into problems getting your web app to work with QBMS servers you would POST to the *test environment* diagnostic tool at this URL:

```
https://webmerchantaccount.ptc.quickbooks.com/j/diag/http
```

When you deploy your application, you need to provide your users with access to this debugging capability so it can be available at runtime in your application. This can help you with problems stemming from certificate expiration and the revocation process. You should post to the *production environment* diagnostics site below,so your users can supply you with needed information.

```
https://webmerchantaccount.quickbooks.com/j/diag/http
```

To interpret and use what you see at the diagnostic site, please refer to this AlphaGeek article:

Troubleshooting Certificates for QuickBooks Merchant Service

IMPORTANT

Due to certificate expiration and the revocation process the above debugging capability needs to be available at runtime.

(c) 2008 Intuit Inc. All rights reserved.

CHAPTER 7
ERROR HANDLING

This chapter covers error handling for both desktop and hosted web applications.

Types of Errors Your Application Must Handle

Your application must handle three types of errors:

- Standard HTTPS errors resulting from your application's attempt to use network resources.
- QBMS connection-related errors (returned in the StatusCode field of SignonDesktopRs or SignonTicketRs) resulting from your application's attempt to communicate with it.
- QBMS or credit card processor errors resulting from attempted transactions that fail or are denied. (These are returned in the StatusCode field of the actual qbmsXML transaction request.)

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

NOTE

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

QBMS Connection-Related Errors

The following is a list of QBMS connection-related errors that can be returned in the StatusCode field of SignonDesktopRs, SignonTicketRs, or SignonAppCertRs.

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

2010
Unauthorized

2020
Session Authentication required

2030
Unsupported signon version

2040
Internal error

(c) 2008 Intuit Inc. All rights reserved.

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

QBMS and Card Processor Errors

For a list of QBMS and card processor errors that could be returned in the StatusCode field of the various qbmsXML transaction requests, see Appendix A, “Status Codes Returned in Responses.”

QBMS Error Recovery

Beginning with qbmsXML 2.0, a new required field is included with all QBMS transaction requests: TransRequestID. (We recommend using UUID/GUID-based values.) This mandatory element should not be confused with the RequestID attribute, which is optional and is not used in QBMS error recovery.

TransRequestID is an application-supplied value that identifies the transaction to QBMS. The purpose of this is to prevent duplicate transactions, as might occur in a network outage where the transaction is actually successful, but a network problem occurred before the sender could be notified of success.

The sender can safely resend the transaction and QBMS will recognize the resend as a duplicate transaction because of the TransRequestID. QBMS will return the proper response data, but no additional processing charge will be assessed.

The TransRequestID need only be unique for the merchant sending the request for a period of 15 minutes. After this time has elapsed, the TransRequestID can be reused. Even if other merchants are using the same application, the ID need only be unique to the one merchant, it need not be unique across all the merchants.

.

(c) 2008 Intuit Inc. All rights reserved.

CHAPTER 8
ACCESSING QBMS FROM DESKTOP APPLICATIONS

This chapter describes how to access QBMS for credit card transactions from a desktop application. It covers connection/session ticket management, building the message signon blocks, handle POSTing to the QBMS data center, and so forth.

Before You Start

You must sign up with QBMS to obtain a PTC test account before you can do any of the things we describe in this chapter in the PTC environment. You must obtain a real QBMS account if you want to any of the things we describe in this chapter in the production QBMS environment.

Finally, your customer/merchant needs to sign up and obtain a real QBMS account before they can run your production application!

Signing up for a PTC account or a real QBMS account is covered in Chapter 5, “Signing Up for a PTC Test QBMS Merchant Account.”

Registration with Intuit Gateways is Required for Access

All applications are required to register with IDN before they can access QBMS. If you don’t register, the Intuit gateways won’t let your application into the QBMS data centers.

To register, visit *appreg.intuit.com*. (Additional registration instructions are located at the *IDN developer website*.)

Keep in mind there are two separate environments that you need to register for:

- Register your application with IDN to use the PTC test environment, to test your application.
- Register your application with IDN to use the QBMS production environment when you’re ready to test live or go live.

IMPORTANT

After registering for PTC, you will be given an AppID that you use to communicate with PTC. After registering for production, you will be given a *different* AppID that you use to communicate with production QBMS. The AppID for the PTC test environment will not work with QBMS production and vice versa!

(c) 2008 Intuit Inc. All rights reserved.

Security Rules For Your Application

This section describes security rules that must be observed in your application code. If your application doesn't follow these rules it may lose access to QBMS.

The following security rules must be observed by your application:

- Your application may not automate any part of the QBMS user interface, including the application authorization process.
- If you integrate with QuickBooks, your application may not request and/or store the user's QuickBooks logon and password.
- You cannot share connection tickets or session tickets between different applications.
- If your application is a browser application, you must not allow your pages to be cached.
- Your application must encrypt the connection ticket before storing it and it must keep the session ticket in memory only.

Accessing QBMS: What You Need to Do

To access QBMS for credit card transactions, you need to do these three things:

- Support QBMS user authorization. This means you need to send your user to the QBMS login sites to get connection tickets and possibly session tickets.
- Using HTTPS, POST the SDK requests to the QBMS merchant account at the QBMS data center.
- Handle the possible HTTPS-related errors and ticket-related errors. (For more details on this, see Chapter 7, "Error Handling.")

The following sections provide more details.

Posting qbmsXML to QBMS (No Session Authentication)

"I just want to post a credit card charge to QBMS: how do I do that?" Well, let's take the simple case where you're using a connection ticket that doesn't require session authentication.

Figure 8-1 shows an overview of what happens in this scenario, including getting the connection ticket and session ticket.

(c) 2008 Intuit Inc. All rights reserved.

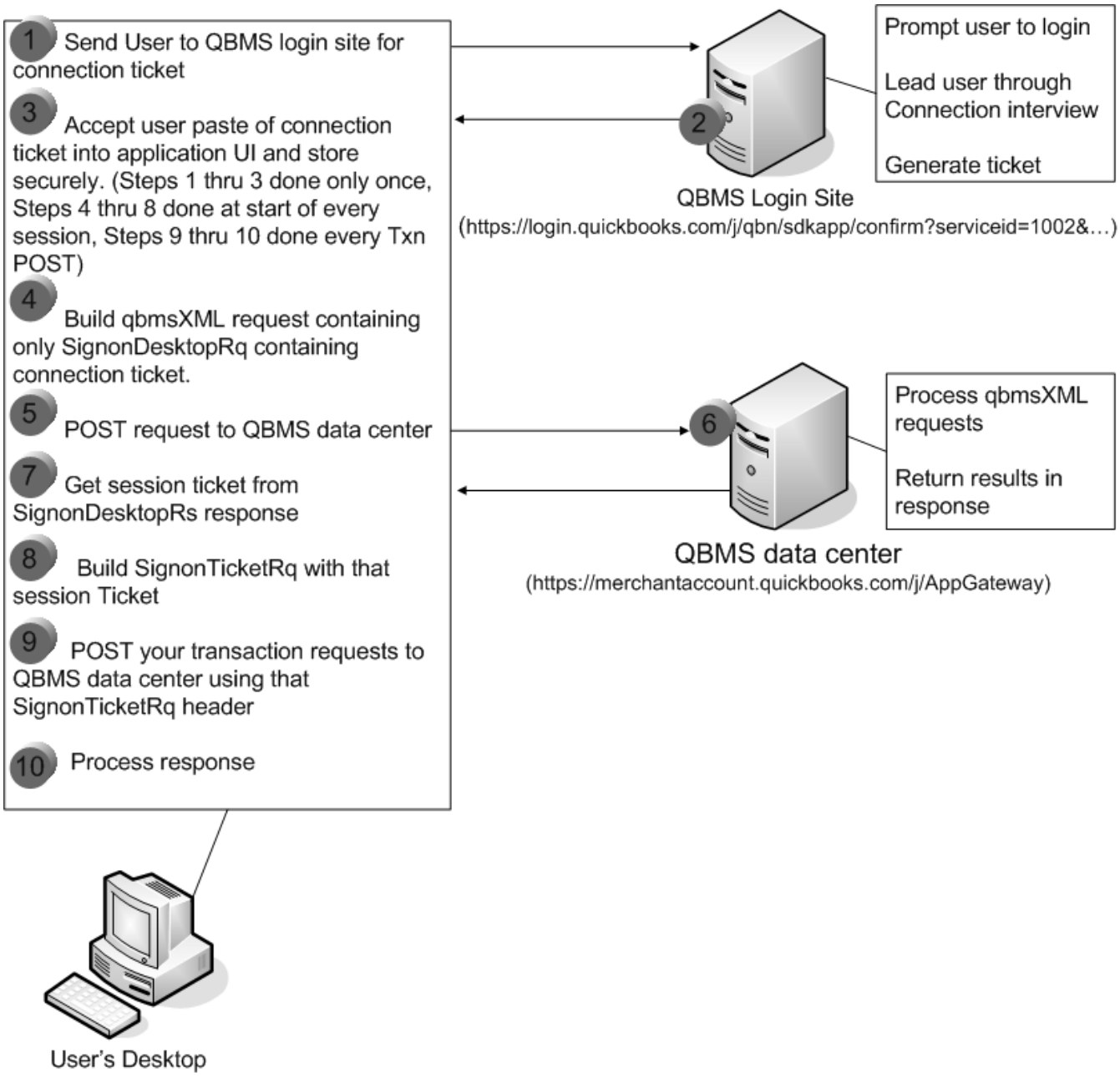

Figure 8-1 POSTing to QBMS, connection ticket only

In Figure 8-1, you only get the connection ticket once and then store it securely for subsequent use, conceivably forever, or until the merchant account owner revokes it. You also need to get the session ticket as shown in the figure only once for every transaction session.

The other things that are happening in this communication will be a bit more clear when we start looking at some samples, which we'll do now.

(c) 2008 Intuit Inc. All rights reserved.

What Your qbmsXML Containing SignonDesktopRq Looks Like

Once you get the connection ticket from the user pasting it into your application (we'll get to connection tickets soon, but for now, hold on), you need to send a qbmsXML string containing only the SignonDesktopRq within the SignonMsgsRq aggregate, as shown below:

Listing 8-1 Sending SignonDesktopRq to Get the Session Ticket

```
<?xml version="1.0" ?>
<?qbmsxml version="2.0"?>
<QBMSXML>
   <SignonMsgsRq>
      <SignonDesktopRq>
         <ClientDateTime>2006-09-20T15:49:26</ClientDateTime>
         <ApplicationLogin>TxnTester.intuit.com</ApplicationLogin>
         <ConnectionTicket>TGT-77-1029837654129087629350</ConnectionTicket>
      </SignonDesktopRq>
   </SignonMsgsRq>
</QBMSXML>
```

Notice we don't supply the optional information here, not even the AppID, because they aren't required once you have the connection ticket. (Starting with qbmsXML 2.0, the AppID, Language and AppVer are optional.)

Figure 8-2 is what the response looks like, if you're successful:

Figure 8-2 SignonDesktop response containing session ticket

We've circled the part we need from the response. You must take that session ticket and build a SignonTicketRq that you'll include with every transaction POSTed to the QBMS data center during the current session.(If you start a new session, you'll need to send the SignonDesktopRq again as shown in Listing 8-1, to get a new session ticket to put in your SignonTicketRq.)

(c) 2008 Intuit Inc. All rights reserved.

What Your Transaction Requests Look Like

Listing 8-2 below is what you would send to QBMS data center for transaction processing. Notice the SignonMsgsRq aggregate containing SignonTicketRq and the session ticket.

_____Listing 8-2 Fully formed Request String Ready to Post to QBMS

```
<?xml version="1.0" ?>
<?qbmsxml version="2.0"?>
<QBMSXML>
   <SignonMsgsRq>
      <SignonTicketRq>
         <ClientDateTime>2006-09-29T08:46:58</ClientDateTime>
         <SessionTicket>V1-32-10298376541290876293 5g:85095501</SessionTicket>
      </SignonTicketRq>
   </SignonMsgsRq>
   <QBMSXMLMsgsRq>
      <CustomerCreditCardAuthRq requestID="23909">
         <TransRequestID>E09C86CF</TransRequestID>
         <CreditCardNumber>4111111111111111</CreditCardNumber>
         <ExpirationMonth>12</ExpirationMonth>
         <ExpirationYear>2008</ExpirationYear>
         <IsECommerce>true</IsECommerce>
         <Amount>203.00</Amount>
         <CreditCardAddress>23 Garcia Ave</CreditCardAddress>
         <CreditCardPostalCode>94043</CreditCardPostalCode>
      </CustomerCreditCardAuthRq>
   </QBMSXMLMsgsRq>
</QBMSXML>
```

The various fields in the SignonDesktopRq and SignonTicketRq are explained in the appendixes. However, the only ones you would normally use are the Client date time and the ticket fields.

How Do You POST the qbmsXML to Production QBMS?

IMPORTANT

The URLs shown here are case sensitive. Be sure to use the case as shown in the examples.

Lets say we have the contents of Listing 8-2 in a string we called XmlString, just picking a name out of the hat. This is how you would post that request string, in VB using a Microsoft XML XMLTTPP40 object:

(c) 2008 Intuit Inc. All rights reserved.

_____ Listing 8-3 Posting Requests to QBMS

```
Dim RequestURL As String
'This is the URL you use to send requests to QBO
RequestURL = "https://merchantaccount.quickbooks.com/j/AppGateway"
Dim objXMLHttp As XMLHTTP40

Set objXMLHttp = New XMLHTTP40
objXMLHttp.open "POST", RequestURL, False
objXMLHttp.setRequestHeader "content-type", "application/x-qbmsxml"
'XmlString is the string shown above in Listing 8-2
objXMLHttp.send XmlString

'for grins, show the response in a message box
Dim resp As String
resp = objXMLHttp.responseText
MsgBox resp
```

That's all there is to it. Your own language of choice will vary the code particulars, but you'll always need to POST qbmsXML (formed like our sample!) to the the URL shown in the above sample.

Wait! How Do I POST to the PTC Test Environment?

IMPORTANT

The URLs shown here are case sensitive. Be sure to use the case as shown in the examples.

You use the same code as that shown for Listing 8-3, only replace the RequestURL line with this one:

```
RequestURL = "https://merchantaccount.ptc.quickbooks.com/j/AppGateway"
```

All we did was insert .ptc between merchantaccount and quickbooks.com. This will send your request to the emulator so you can get a test response.

Sending the User to Get a Connection Ticket

We've put off discussing this so you would have some context first of the overall process of getting tickets and sending transaction requests. Now we need to show how to get a connection ticket.

If you don't already have the connection ticket, you must send your user to the QBMS login site, where the user logs on to his or her QBMS account and follows the connection prompts to create a connection ticket for your application. The user then copies that ticket into your application via the Windows clipboard.

(c) 2008 Intuit Inc. All rights reserved.

Here's how to send the user to the login site to get a connection ticket in VB:

_____Listing 8-4 Sending User to Production QBMS Login site for Connection Ticket

```
Dim loginURL As String
loginURL = "https://merchantaccount.quickbooks.com/j/sdkconnection/connectionList?
                                                                    appid=56988448"
Dim IE1 As New InternetExplorer
IE1.Visible = True
IE1.Navigate (loginURL)
```

The only thing you change in the above code so it works with your application is the appid value *56988448*. That must be replaced by your own application ID obtained when you registered your application with IDN.

As a result of this code, the user will be sent to the QBMS login site to login to the user's QBMS merchant account. Then, if the user chooses to grant your application the connection ticket, the user will go through the connection interview at that site and copy the resulting connection ticket into your application, into the UI component you will have thoughtfully provided.

You'll encrypt the connection ticket and store it persistently, because that ticket is good til the user cancels it. (For sample encryption code, see "Using wincrypt to Store Connection Tickets.")

Getting a Connection Ticket from the PTC Enviroment

To get a PTC connection ticket, use the code shown in Listing 8-4, but change the loginURL line to:

```
loginURL = "https://merchantaccount.ptc.quickbooks.com/j/sdkconnection/connectionList?
                                                                    appid=56988448"
```

again, changing the appid value to whatever your own AppID is.

Detecting/Handling Invalid Connection Tickets

In Listing 8-4 we showed you how to send the user to get a connection ticket. You could put this code in a "subscribe" button click event handler if you want. But whether you use it that way or not, you must always also put that code in a check for an invalid connection ticket. Why?

The connection ticket may be removed at any time by the user, or be rendered invalid by QBMS itself for security reasons. You cannot assume that once you have the ticket, it will always work. If the connection ticket is invalid (or even missing!), the response to your qbmsXML request will contain a status code of 2000 in the StatusCode attribute within the SignonDesktopRs. So you need to check for this condition and if it occurs, send the user back to the QBMS login site to get a valid one.

(c) 2008 Intuit Inc. All rights reserved.

The following code shows you how to check for an invalid or missing connection ticket and then respond by sending the user to the QBMS login site for a connection ticket. We use a DOM document to make life easier in getting to the SignonDesktopRs field we want.

Listing 8-5 Checking for Invalid or Missing Connection Ticket

```
' This is the response to the POST where we send the SignonDesktopRq request to QBMS
resp = objXMLHttp.responseText

Dim doc As DOMDocument40
Set doc = New DOMDocument40
doc.async = False
doc.validateOnParse = False

' Load the response into DOM and traverse to the StatusCode field in SignonDesktopRs
' which is within SignonMsgsRs
doc.loadXML (resp)
Dim top As IXMLDOMNode
Set top = doc.documentElement
Dim responses As IXMLDOMNodeList

' All we care about is the SignonMsgsRs part
Set responses = top.selectNodes("SignonMsgsRs")
Dim rs As IXMLDOMElement
Set rs = responses.Item(0)
Dim dtResponses As IXMLDOMNodeList

' Then we need the response to the SignonDesktopRq
Set dtResponses = rs.selectNodes("SignonDesktopRs")
Dim dtRs As IXMLDOMElement
Set dtRs = dtResponses.Item(0)
Dim dtStatusCode As String

' Look at the status code
dtStatusCode = dtRs.getAttribute("statusCode")
dtStatusMsg = dtRs.getAttribute("statusMessage")

If "2000" = dtStatusCode Then
   MsgBox ("Connection ticket required: please get ticket and paste it into our app")
   Dim loginURL As String
   loginURL = "https://merchantaccount.quickbooks.com/j/sdkconnection/
                                                        connectionList?appid=56988448"
   Dim IE1 As New InternetExplorer
   IE1.Visible = True
   IE1.Navigate (loginURL)
End If
```

To make the above code work for PTC, replace the loginURL line in Listing 8-5 with this:

(c) 2008 Intuit Inc. All rights reserved.

```
loginURL ="https://merchantaccount.ptc.quickbooks.com/j/sdkconnection/connectionList?
                                                                    appid=56988448"
```

Again, replace the appid value in the loginURL with your own AppID.

Posting qbmsXML to QBMS with Session Authentication

In Listing 8-2 through Listing 8-4 we showed you the simple case, posting qbmsXML to the QBMS data center where the connection ticket used does not require session authentication. But in "real life," users will create some connection tickets with session authentication and some without it. Because there is no way your application can figure out in advance which kind of connection ticket it is dealing with, your application needs to handle both possibilities.

So how *do* you handle this situation? Figure 8-3 shows an overview of what needs to happen:

(c) 2008 Intuit Inc. All rights reserved.

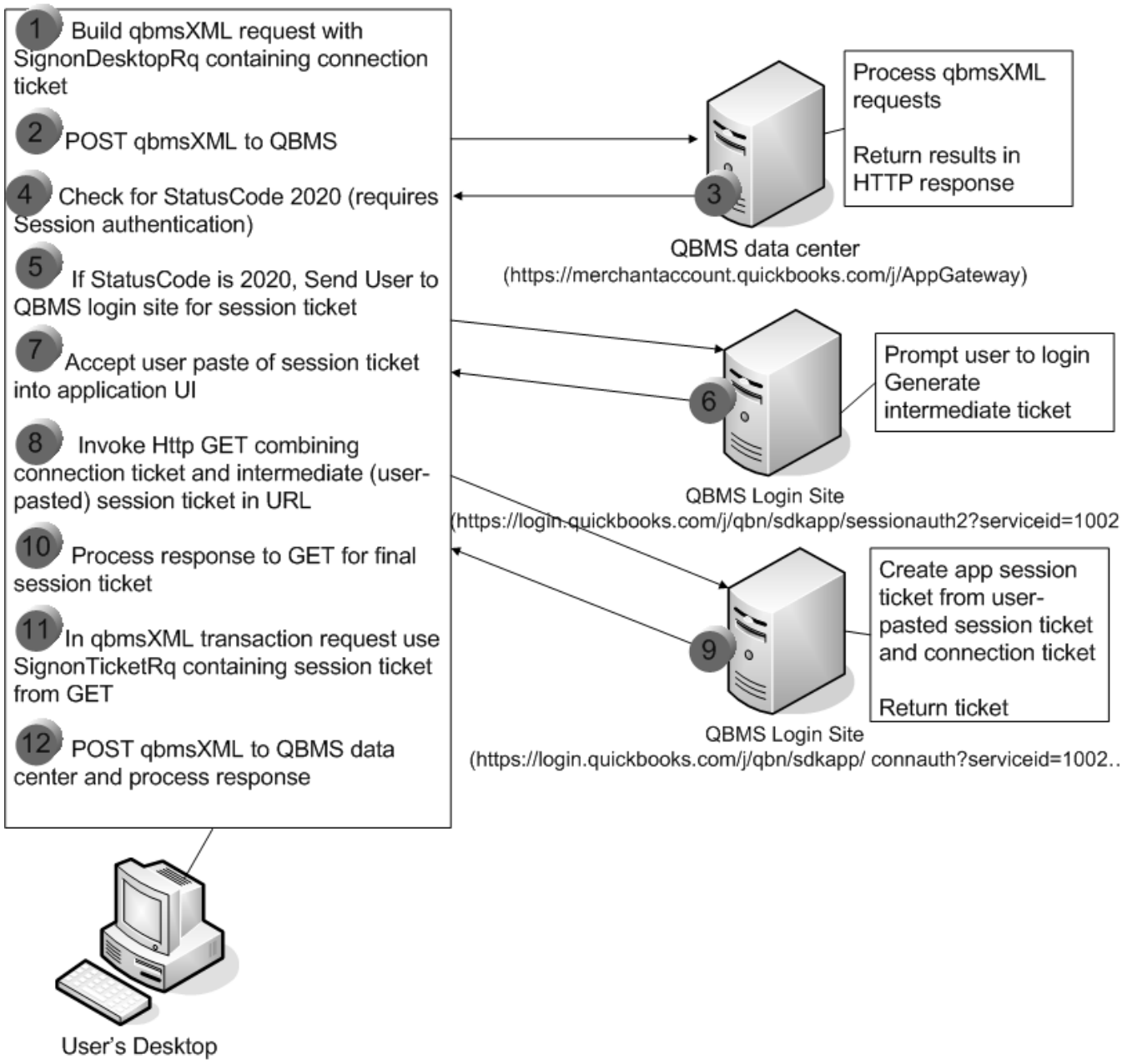

Figure 8-3 Handling connection tickets requiring session authentication

Here's what Figure 8-3 is showing you:

1. POST the qbmsXML to the data centers just like we showed you in Listing 8-2 through Listing 8-4.
2. Check the response to that POST: if the connection ticket requires session authentication, you'll get an error 2020 returned in the StatusCode field of SignonDesktopRs.

(c) 2008 Intuit Inc. All rights reserved.

3. Respond to the StatusCode 2020 by sending the user to the QBO login site, where the user logs into his or her QBMS account and obtains a session ticket to paste into your application's UI.
4. Now for the tricky part. The user-pasted session ticket is an intermediate ticket: you can't use it to access the QBMS merchant account just yet. The QBMS login site needs to combine it with the connection ticket for security reasons. You need to supply the connection ticket and the intermediate (user-pasted) session ticket in a URL that points to the QBMS login site again so the QBMS site can do what it needs to do without user intervention. (We'll show you how to build this URL in Listing 8-6.)
5. Invoke an HTTPS GET using that URL. The login site will construct your application session ticket from the connection ticket and the user-pasted session ticket. The actual application-ready session ticket is returned in the response to that GET. (Be careful: the first three characters in the response are for status: the ticket starts at the fourth character!)
6. In the qbmsXML transaction request you want to POST, insert a SignonTicketRq aggregate containing the application-ready session ticket from the GET, which you have obtained in such a laborious fashion.
7. POST the qbmsXML to the QBMS data centers.

Have you got all that? Here's some code to make things a bit more clear.

Sending User to Get Intermediate Session Ticket

In our sample, we are POSTing the qbmsXML shown in Listing 8-2. Our POST uses SignonDesktopRq containing a connection ticket. Because this connection ticket requires session authentication, the qbmsXML request is not successful, but instead error 2020 is returned in the status code for SignonDesktopRs.

Here's that activity and our handling of it, which is to message the user and send the user to the login site to log in and get a session ticket.

_____Listing 8-6 Handling Error 2020: Requires Session Authentication

```
' Post qbmsXML to QBMS
Dim RequestURL As String
RequestURL = "https://merchantaccount.quickbooks.com/j/AppGateway"

' XmlString is the qbXML shown in Listing 8-2
Dim MyHttpObject As XMLHTTP40
Set MyHttpObject = New XMLHTTP40
MyHttpObject.open "POST", RequestURL, False
MyHttpObject.setRequestHeader "content-type", "application/x-qbmsxml"
MyHttpObject.send XmlString
```


(c) 2008 Intuit Inc. All rights reserved.

```
' Get the response to the POST and load it into DOM doc
Dim resp As String
resp = MyHttpObject.responseText
Dim doc As DOMDocument40
Set doc = New DOMDocument40
doc.async = False
doc.validateOnParse = False
doc.loadXML (resp)

' Traverse DOM doc to the node containing SignonDesktopRs
' and get the StatusCode from it
Dim top As IXMLDOMNode
Set top = doc.documentElement
Dim responses As IXMLDOMNodeList
'All we care about is the SignonMsgsRs part
Set responses = top.selectNodes("SignonMsgsRs")

Dim ResponseItem As IXMLDOMElement
Set ResponseItem = responses.Item(0)
Dim dtResponses As IXMLDOMNodeList
Set dtResponses = ResponseItem.selectNodes("SignonDesktopRs")
Dim dtRs As IXMLDOMElement
Set dtRs = dtResponses.Item(0)
Dim dtStatusCode As String
dtStatusCode = dtRs.getAttribute("statusCode")

' If status code is 0, the qbmsXML request got through and we have a good session ticket,
' which means our connection ticket didn't require session authentication
' If status code is 2020, it didn't get through: need to do session authentication
' so send user to QBMS login site to get one
If "2020" = dtStatusCode Then
   MsgBox ("Login to QBMS and get the session ticket from QBMS")
   Dim SessAuthURL As String
   SessAuthURL = "https://login.quickbooks.com/j/qbn/sdkapp/sessionauth2?serviceid=
                                    1002&appid=56988448&service_flags=qbmssdk%3dtrue"
   Dim IE1 As New InternetExplorer
   IE1.Visible = True
   IE1.Navigate (SessAuthURL)
   End If
```

Again, if you're working in the PTC environment, replace the sessAuthURL line with:

```
SessAuthURL = "https://login.ptc.quickbooks.com/j/qbn/sdkapp/sessionauth2?serviceid=
                                    1002&appid=56988448&service_flags=qbmssdk%3dtrue"
```

And in both cases, make sure you replace the appid value shown in these samples with your own AppID given you by IDN as a result of registering your app.

(c) 2008 Intuit Inc. All rights reserved.

The end result of this code is the intermediate session ticket that you need to supply in a GET to the QBMS login site as we'll show you right now.

Transforming the Intermediate (User-Pasted) Session Ticket

If the user choses to grant the session ticket to your application, the result from the code in Listing 8-6 is a session ticket pasted into your application UI by your user. You need to submit this session ticket in a URL that you send to QBMS(via an HTTPS GET) for further transformation for security purposes.

This is shown in Listing 8-7:

_____Listing 8-7 Transforming the User-Pasted Session Ticket

```
' Read the user-pasted session ticket from our UI texbox control
Dim SessTicketFromUser As String
SessTicketFromUser = SessionTicket.Text

' Put connection ticket and user-pasted session ticket in the URL to be used
' in the GET invocation: GET invoked on the QBMS login site. Notice there is
' no user interaction in this.
Dim authURL As String
authURL = "https://login.quickbooks.com/j/qbn/sdkapp/
connauth?serviceid=1002&appid=56988448&conntkt=TGT-135-1029837654129087629353zg" +
"&sessiontkt=" + SessTicketFromUser

' Do the GET and get the response. The first three response chars are the status code:
' check this for success (000 is success) and get the ticket starting at the 4th char
Dim http As New XMLHTTP40
http.open "GET", authURL, False
http.send authURL
Dim resp As String
Dim status As String
resp = http.responseText

status = Mid(resp, 1, 3)
If (Not status = "000") Then
   MsgBox "Problem updating session ticket"
   Exit Sub
End If

' We now have the application-ready session ticket: this goes into
' the SignonTicketRq aggregate that we'll substitute for the SignonDesktopRq
' before we POST our qbmsXML
resp = Mid(resp, 4)
```


(c) 2008 Intuit Inc. All rights reserved.

Substitute your own AppID value for the one we show in the authURL line. Also, to make this work with PTC, replace that authURL line with this one:

```
authURL = "https://login.ptc.quickbooks.com/j/qbn/sdkapp/connauth?serviceid=1002&appid=
56988448&conntkt=TGT-135-1029837654129087629353zg" + "&sessiontkt=" + SessTicketFromUser
```

Using the Transformed Session Ticket in Your qbmsXML

Once QBMS returns the transformed session ticket from your GET, as shown in Listing 8-7, you must supply it in the SessionTicket element within a SignonTicketRq aggregate. The resulting POST-ready qbmsXML string will look exactly like that shown in Listing 8-8.

_____ Listing 8-8 Sample qbmsXML for transaction sent under session authentication

```
<?xml version="1.0" ?>
<?qbmsxml version="2.0"?>
<QBMSXML>
   <SignonMsgsRq>
      <SignonTicketRq>
         <ClientDateTime>2006-09-29T08:46:58</ClientDateTime>
         <SessionTicket>V1-32-102983765412908762935g:85095501</SessionTicket>
      </SignonTicketRq>
   </SignonMsgsRq>
   <QBMSXMLMsgsRq>
      <CustomerCreditCardAuthRq requestID="23909">
         <TransRequestID>E09C86CF</TransRequestID>
         <CreditCardNumber>4111111111111111</CreditCardNumber>
         <ExpirationMonth>12</ExpirationMonth>
         <ExpirationYear>2008</ExpirationYear>
         <IsECommerce>true</IsECommerce>
         <Amount>203.00</Amount>
         <CreditCardAddress>23 Garcia Ave</CreditCardAddress>
         <CreditCardPostalCode>94043</CreditCardPostalCode>
      </CustomerCreditCardAuthRq>
   </QBMSXMLMsgsRq>
</QBMSXML>
```

URLs Used to Access QBMS from Desktop Applications

IMPORTANT

The URLs shown here are case sensitive. Be sure to use the proper case as shown.

The URLs used to access QBMS are listed in the following table.

(c) 2008 Intuit Inc. All rights reserved.

Table 8-1 PTC Test Environment and Production URLs

Function of the URL	PTC URLs	Production URLs
Send user to get connection ticket	https://merchantaccount.ptc.quickbooks.com/j/sdkconnection/connectionList?appid=myAppID	https://merchantaccount.quickbooks.com/j/sdkconnection/connectionList?appid=myAppID
Send user to get intermediate Session ticket	https://login.ptc.quickbooks.com/j/qbn/sdkapp/sessionauth2?serviceid=1002&appid=MyAppID&service_flags=qbmssdk%3dtrue	https://login.quickbooks.com/j/qbn/sdkapp/sessionauth2?serviceid=1002&appid=MyAppID&service_flags=qbmssdk%3dtrue
URL for HTTPS "GET" to transform intermediate session ticket	https://login.ptc.quickbooks.com/j/qbn/sdkapp/connauth?serviceid=1002&appid=MyAppID&conntkt=MyConnTicket&sessiontkt=SessTicketFromUser where MyAppID is your AppId value, MyConnTicket is your connection ticket, and SessTicketFromUser is the user-pasted session ticket.	https://login.ptc.quickbooks.com/j/qbn/sdkapp/connauth?serviceid=1002&appid=MyAppID&conntkt=MyConnTicket&sessiontkt=SessTicketFromUser where MyAppID is your AppId value, MyConnTicket is your connection ticket, and SessTicketFromUser is the user-pasted session ticket.
Where to POST transaction requests	https://merchantaccount.ptc.quickbooks.com/j/AppGateway	https://merchantaccount.quickbooks.com/j/AppGateway

The SignonDesktop and SignonTicket Request Definitions

The aggregate definitions for SignonDesktopRq and SignonTicketRq, along with their responses are provided in the SDK subdirectory \doc, in the files qbmsxmlso20.xml.

For your convenience, we've listed and explained some of these in Appendix B, "Signon Requests and Responses XML."

Using wincrypt to Store Connection Tickets

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

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


unsigned char *GetRandomBytes( unsigned long *length )
{
```


(c) 2008 Intuit Inc. All rights reserved.

```
  HCRYPTPROV   hCryptProv;
  BYTE     pbData[1024]; // Size chosen arbitrarily.
  BYTE     *output;

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

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

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

  return output;
}

void main()
{

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

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


(c) 2008 Intuit Inc. All rights reserved.

```
  DATA_BLOB DataIn;
  DATA_BLOB DataOut;
  DATA_BLOB DataVerify;
  BYTE *pbDataInput =(BYTE *)"Hello world of data protection.";
  DWORD cbDataInput = strlen((char *)pbDataInput)+1;
  DataIn.pbData = pbDataInput;
  DataIn.cbData = cbDataInput;
  CRYPTPROTECT_PROMPTSTRUCT PromptStruct;
  LPWSTR pDescrOut = (LPWSTR)0xbaadf00d ; // NULL;


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

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

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

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

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

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

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

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


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

  if(CryptProtectData(
    &DataIn,
    L"We're protecting test data. ",  // A description sting.
    &random,              // Optional entropy.
    NULL,                                // Reserved.
```


(c) 2008 Intuit Inc. All rights reserved.

```
    &PromptStruct,                      // Pass a PromptStruct.
    0,
    &DataOut))
  {
    printf("The encryption phase worked. \n");
  }
  else
  {
    HandleError("Encryption error!");
  }
  //-----------------------------------------------------------------
  //   Begin unprotect phase.

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

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

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

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


(c) 2008 Intuit Inc. All rights reserved.

```
//  that does more extensive error reporting.

void HandleError(char *s)
{
    fprintf(stderr,"An error occurred in running the program. \n");
    fprintf(stderr,"%s\n",s);
    fprintf(stderr, "Error number %x.\n", GetLastError());
    fprintf(stderr, "Program terminating. \n");
    exit(1);
} // End of HandleError
```


(c) 2008 Intuit Inc. All rights reserved.

(c) 2008 Intuit Inc. All rights reserved.

CHAPTER 9
ACCESSING QBMS FROM HOSTED WEB APPLICATIONS

This chapter describes what you need to do to enable a hosted web application to communicate with QBMS.

Task Checklist

To enable your hosted web application to communicate with QBMS, you must do the following:

1. Obtain a PTC QBMS account for testing during development, and a production one for live testing.
2. Register your application with IDN (*appreg.intuit.com*).
3. Obtain a server certificate from a supported root certificate authority (see Appendix D).
4. Obtain a client certificate by generating a certificate signing request (CSR) and use the *appreg.intuit.com* site to get it signed by Intuit.
5. Understand and follow the QBMS security requirements for hosted web applications.
6. In your code, using SSL, implement the presentation of the client certificate to be used when POSTing to QBMS.
7. In your code, prompt your merchant/customer to grant a connection ticket authorizing your application to access the merchant's QBMS account. Respond by sending merchant to the QBMS login page to get a connection ticket. QBMS POSTs the connection ticket back to the application subscription URL you specified when you registered your application. Handle this POST at that URL and store the connection ticket securely.
8. In your code, implement session ticket handling code. That is, prior to sending QBMS transaction requests for a particular merchant, get a session ticket by sending a SignonMsgsRq containing a SignonAppCertRq with that merchant's connection ticket. You'll POST this to the QBMS data exchange URL to get the session ticket in the SignonAppCertRs response.
9. In your code, build the desired requests in a qbmsXML string. This string contains a SignonMsgsRq containing a SignonTicketRq with the session ticket from the SignonAppCertRs in the previous step, followed by the QBMS transaction requests. POST this whole XML string to the QBMS data exchange URL and process the response.
10. In your code, handle connection ticket cancellation notification from QBMS. (If the merchant cancels the connection ticket, QBMS sends a notification to the Application Cancel URL that you specified when you registered your application.)

We'll describe these tasks in more detail in this chapter.

(c) 2008 Intuit Inc. All rights reserved.

Obtaining a QBMS Account

You must obtain a PTC test account before you can POST transactions to the PTC environment. You must obtain a real QBMS account if you want to POST in the production QBMS environment for live testing.

Finally, your customer/merchant needs to sign up and obtain a real QBMS account before they can run your production application.

Signing up for a PTC account or a real QBMS account is covered in Chapter 5, “Signing Up for a PTC Test QBMS Merchant Account.”

Registration with Intuit Gateways is Required for Access

All applications are required to register with IDN before they can access QBMS. If you don’t register, the Intuit gateways won’t let your application into the QBMS data centers.

To register, visit *appreg.intuit.com*. (Additional registration instructions are located at the *IDN developer website*.)

Keep in mind there are two separate environments that you need to register for:

- Register your application with IDN to use the PTC test environment, to test your application.
- Register your application with IDN to use the QBMS production environment when you’re ready to test live or go live.

IMPORTANT

After registering for PTC, you will be given an AppID that you use to communicate with PTC. After registering for production, you will be given a *different* AppID that you use to communicate with production QBMS. The AppID for the PTC test environment will not work with QBMS production and vice versa!

Hosted Applications Need Certificates to Access Intuit Gateways

Your hosted web application needs a server certificate in order to receive callbacks from QBMS. Appendix D lists the root certificate authorities (CAs) that are known to work with QBMS.

Your hosted web application also needs a client certificate signed by Intuit in order to POST QBMS transaction requests to QBMS. (You need register your application with IDN before you can generate your certificate signing request.) If you haven’t done this before, or are unsure of the process, more instructions are provided at the IDN website at this location:

(c) 2008 Intuit Inc. All rights reserved.

http://developer.intuit.com/qbms/integration_center/?id=1294

Security Requirements

The following security rules must be observed by your application:

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

Failure to follow these requirements may result in your application losing access to QBMS.

How to Present the Client Certificate to QBMS

The Intuit-signed client certificate must be presented to QBMS at every POST to QBMS sites. You have to use SSL at your web server in conjunction with the client certificate.

An ASP.NET Example

Details on certificate management for ASP.NET implementations are available at the IDN website, in the *AlphaGeek* archives. Also, check out the sample program IDNRequestor at the SDK subdirectory \samples\qbms\c-sharp.

A Java Example: Presenting a Client Certificate in a Java Servlet

Suppose your server application is implemented via a Java servlet. You just add a few lines of code in your servlet's init method to specify the use of SSL, your keystore, and your keystore password, and the IDN certificate will be used automatically for all POSTs to QBMS.

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

The following code (from the sample servlet WebTxnTester in the SDK subdirectory \samples\qbms) shows the required lines in the servlet init method:

(c) 2008 Intuit Inc. All rights reserved.

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

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

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

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

How to Implement Connection Ticket Support

Implementing connection ticket support in hosted applications differs from desktop application implementations. In a hosted application, you do have to send the merchant to the QBMS connection list URL to create a connection ticket to grant your application authorization to access the merchant's account, just like you do for a desktop application.

However, there is no copy and paste of connection tickets into a hosted application. Instead, once created and assigned by the merchant, the connection ticket is automatically POSTed by QBMS to the subscription URL that you specified when you registered your application. You just need to handle the ticket POST at that URL.

Also, unlike desktop applications, there can be *no session authentication* for hosted applications, so you don't code for that possibility.

Implementing connection ticket support in hosted web applications consists of the following:

1. Enabling the merchant to start the authorization process from your main form in the web browser and responding to the merchant action by sending the merchant to the QBMS connection page to create a connection ticket. QBMS will POST the connection ticket back to your application.
2. Handling the connection ticket POST from QBMS at the application subscription URL you specified when you registered your application. You'll extract the connection ticket from the POST and store it securely for future use for that particular merchant. Your application will use it to automatically obtain a session ticket when beginning a session to send QBMS transactions to that merchant's QBMS account, as described in "Posting QBMS Transactions to the Data Exchange URL."

(c) 2008 Intuit Inc. All rights reserved.

Sending the Merchant to QBMS to Create a Connection Ticket

In your main form displayed in the user's web browser, provide a means for the merchant to start the authorization process, for example, a Subscribe button. Whatever the means, respond to the merchant's action by sending the merchant to the QBMS login page at the following URL:

for production:

```
https://login.quickbooks.com/j/qbn/sdkapp/confirm?appid=<myAppid>&serviceid=1002
                                                          &appdata=<myAppData>
```

for PTC test:

```
https://login.ptc.quickbooks.com/j/qbn/sdkapp/confirm?appid=<myAppid> &serviceid=1002
                                                          &appdata=<myAppData>
```

where

-https://login.quickbooks.com/j/qbn/sdkapp/confirm?
or
https://login.ptc.quickbooks.com/j/qbn/sdkapp/confirm?
is the location of the login page.

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

-serviceid
Must be set to the value 1002.

myAppData
is the unique (unique within the application) ID representing the merchant for whom the connection ticket is intended. The AppData gives you a way to identify which connection ticket belongs with which merchant, so you would normally store this along with the returned connection ticket.

After you send the merchant to the QBMS login site, the merchant is led through the authorization process. After the process is complete and a connection ticket is created and assigned, the user is notified of the success, and a connection ticket is returned to your Application Subscription callback URL.

Handling the Connection Ticket POST from QBMS

After your application gets the POST from QBMS, you need to extract the connection ticket from the POST.

(c) 2008 Intuit Inc. All rights reserved.

Example: Extracting the Connection Ticket

The following sample method (from the WebTxntester sample from the SDK samples\qbms subdirectories, is invoked from the servlet doPost method, which handles the POST back from QBMS.

In this snippet, the HTTP request (from the QBMS POST of the connection ticket from the merchant) is queried for the appdata and the connection ticket parameters, the connection ticket, and an intermediate session ticket.

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

      if (request.getContentType() != null) {
         conntkt = request.getParameter("conntkt");
         appdata = request.getParameter("appdata");
         appid = request.getParameter("appid");
         }
      }
```

The WebTxnTester sample from the SDK stores this data, if you want to see that aspect of it.

Getting a Session Ticket for Use in QBMS Transaction POSTs

A session ticket is valid for 60 minutes after its last use, or for an absolute maximum of 24 hours even if used within every 60 minute timeframe, then it expires.

Remember that you need to have a connection ticket before getting a session ticket. Also, remember that connection tickets aren't directly used in QBMS transactions from hosted applications, only session tickets are.

To see what we mean by this, take a look at a typical QBMS transaction POST from a hosted web application looks like the one in Listing 9-1.

(c) 2008 Intuit Inc. All rights reserved.

_____Listing 9-1 Fully formed QBMS transaction request ready to post to QBMS

```
<?xml version="1.0" ?>
<?qbmsxml version="2.0"?>
<QBMSXML>
   <SignonMsgsRq>
      <SignonTicketRq>
         <ClientDateTime>2006-09-29T08:46:58</ClientDateTime>
         <SessionTicket>V1-32-10298376541290876293 5g:85095501</SessionTicket>
      </SignonTicketRq>
   </SignonMsgsRq>
   <QBMSXMLMsgsRq>
      <CustomerCreditCardAuthRq requestID="23909">
         <TransRequestID>E09C86CF</TransRequestID>
         <CreditCardNumber>4111111111111111</CreditCardNumber>
         <ExpirationMonth>12</ExpirationMonth>
         <ExpirationYear>2008</ExpirationYear>
         <IsECommerce>true</IsECommerce>
         <Amount>203.00</Amount>
         <CreditCardAddress>23 Garcia Ave</CreditCardAddress>
         <CreditCardPostalCode>94043</CreditCardPostalCode>
      </CustomerCreditCardAuthRq>
   </QBMSXMLMsgsRq>
</QBMSXML>
```

Notice the SignonTicketRq, which contains a session ticket and not a connection ticket. What happened to the connection ticket? How did we get the session ticket? We get the session by doing this:

1. To the QBMS data exchange URL, POST a SignonMsgsRq request that contains an SignonAppCertRq request. Notice that you'll need to supply the connection ticket here.
2. The response from QBMS will contain a SignonAppCertRs that contains the session ticket. Retrieve and use this session ticket in all subsequent QBMS transaction POSTs for the current session for the merchant.

Listing 9-2 shows a sample QBMSXML string like the one you need to POST:

_____Listing 9-2 Sending SignonAppCertRq to Get the Session Ticket

```
<?xml version="1.0" ?>
<?qbmsxml version="2.0"?>
<QBMSXML>
   <SignonMsgsRq>
      <SignonAppCertRq>
         <ClientDateTime>2006-09-20T15:49:26</ClientDateTime>
         <ApplicationLogin>WebTxnTester.intuit.com</ApplicationLogin>
         <ConnectionTicket>TGT-77-10298376541290876293 5Q</ConnectionTicket>
```


(c) 2008 Intuit Inc. All rights reserved.

```
      </SignonAppCertRq>
   </SignonMsgsRq>
</QBMSXML>
```

Of course, your own values for date/time, application login, and the connection ticket would be different, and the connection ticket would of course be different for each of your merchants.

This is where you'd POSTing the SignonAppCertRq to get the session ticket:

For production QBMS:

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

For the PTC test environment:

```
https://webmerchantaccount.ptc.quickbooks.com/j/AppGateway
```

Posting QBMS Transactions to the Data Exchange URL

Listing 9-1 shows what your XML string should look like when you POST QBMS transactions. That listing shows only one QBMS transaction request: you can have a maximum of 20 requests in each POST.

Notice that a hosted web application cannot POST to the same QBMS data exchange URL that desktop applications do. Hosted web applications and desktop applications must use different URLs.

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

For production QBMS:

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

For PTC test:

```
https://webmerchantaccount.ptc.quickbooks.com/j/AppGateway
```

URLs Used to Access QBMS from Hosted Applications

The URLs used to access QBMS are listed in the following table.

(c) 2008 Intuit Inc. All rights reserved.

Table 9-1 PTC Test Environment and Production URLs

Function of the URL	**PTC URLs**	**Production URLs**
Send user to get connection ticket	https://login.ptc.quickbooks.com/j/qbn/sdkapp/confirm?appid=<myAppid>&serviceid=1002&appdata=<myAppData> where MyAppID is your AppId value, and MyAppData is your own ID identifying the merchant whose connection ticket this is.	https://login.quickbooks.com/j/qbn/sdkapp/confirm?appid=<myAppid>&serviceid=1002&appdata=<myAppData> where MyAppID is your AppId value, and MyAppData is your own ID identifying the merchant whose connection ticket this is.
POSTing the SignonAppCert request to get the session ticket	https://webmerchantaccount.ptc.quickbooks.com/j/AppGateway	https://webmerchantaccount.quickbooks.com/j/AppGateway
Where to POST transaction requests	https://webmerchantaccount.ptc.quickbooks.com/j/AppGateway	https://webmerchantaccount.quickbooks.com/j/AppGateway

(c) 2008 Intuit Inc. All rights reserved.

(c) 2008 Intuit Inc. All rights reserved.

CHAPTER 10
SUPPORTING YOUR CUSTOMER/Merchant

This chapter describes some of the issues affecting your customers that you need to be aware of.

Customers With Existing QBMS Accounts

Some of your customer/merchants may already have a QBMS account, but one that is not set up for ecommerce applications. This will be detected when those merchants first try to access their QBMS account via your application, and they will get a dialog screen that will prompt them to set up their account for ecommerce.

You can add this ability to your existing QBMS account *at this location in the QBMS website*. The web site calls this “adding a Web Store process to your existing account.”

(c) 2008 Intuit Inc. All rights reserved.

(c) 2008 Intuit Inc. All rights reserved.

APPENDIX A
STATUS CODES RETURNED IN RESPONSES

Each qbmsXML response message returned from QBMS includes a statusCode, a statusSeverity, and statusMessage element.

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

Status Code	Status Message	Description
0	Status OK.	Request succeeded.
10100	Warning. The transaction processed with the ID had a validation error, but could not be voided automatically. If you want to void or cancel it, you must do so manually.	For Example: The transaction processed with the ID 25235 had a validation error:(Incorrect Street Address and Zip Code), but could not be voided automatically. If you want to void or cancel it, you must do so manually
10101	Warning. The transaction processed with the ID <id> had a validation error.	This warning can be returned if the fraud settings for the merchant account cause some auth transactions to be rejected. In this case, even in the case of a rejected auth, the auth is kept on file against the customer card up to 15-30 after the rejection.
10200	An error occurred while communicating with the credit card processing gateway.	Indicates a communication problem rather than an error resulting from transaction content.
10201	An error occurred during login to the processing gateway.	Could not log in to the QBMS credit card processing gateway.
10202	An error occurred during account validation.	Could not validate the QBMS account.
10300	An error occurred while converting the amount in the field.	A numeric value was of the wrong type. For example, some values require integers only, others require decimals. See the OSR for required types in the request fields.
10301	This credit card account number is invalid.	Customer's credit card number was invalid.
10302	An error occurred while validating the date value in the field.	The date string format is invalid. See the OSR for required format.
10303	A required element is empty.	A mandatory element is missing from the request. See the OSR for required elements for the request you are using. A required element CreditCardNumber or Track2Data is empty.

(c) 2008 Intuit Inc. All rights reserved.

Status Code	Status Message	Description
10304	The string in the field is too long. The maximum length is <length>	The request field accepts fewer characters than the supplied string. See the OSR for per-field character limits.
10305	An error occurred during data validation.	Unexpected data was found during the validation.
10306	An error occurred while validating the boolean value in the field.	You may only supply the Boolean values true or false
10307	The string in the field is too short. The minimum length is <length>	The expected minimum data is not present. For example, Track3Data requires a minimum of 23 characters.
10308	Maximum requests allowed in one POST cannot exceed 20	You have too many requests in one POST. A maximum of 20 is allowed.
10309	The field has an invalid format.	The data has an invalid format, for example, Track2Data is not formatted as required.
10312	The field is invalid.	You'll get this error for various reasons. If a value supplied in a field of type AMTTYPE is not in the format required by AMTTYPE, you'll get this error. Another instance where you'll get this error is if a supplied field lacks the minimum characters required or exceeds the maximum. For example, ServerID in the Restaurant aggregate must be two and only two characters.
10313	The aggregate is invalid	Currently, you'll get this error from the Lodging aggregate if CheckInDate is later than CheckOutDate.
10400	This account does not have sufficient funds to process this transaction.	The attempted transaction exceeded the customer's credit limit on the credit card. The transaction failed.
10401	The request to process this transaction has been declined.	The card issuer/processor refused to authorize the transaction. For example, the account may have been closed, or activities may have been temporarily stopped for that card due to security reasons.
10402	The merchant bank account does not support this type of credit card.	See the list of supported credit card types in Chapter 1 of this manual.
10403	The merchant account information submitted is not recognized.	The QBMS account contained in the request is not valid.
10404	This transaction has been declined, but can be approved by obtaining a Voice Authorization code from the card issuer.	You need to obtain a voice authorization code from the card issuer, then invoke the CustomerCreditCardCharge request with the VoiceAuthCode field filled in with that code.

(c) 2008 Intuit Inc. All rights reserved.

Status Code	Status Message	Description
10405	An error occurred while attempting to void this transaction.	The transaction could not be voided. Possibly the transaction has already been settled.
10406	An error occurred while processing the capture transaction.	This capture transaction could not be processed for one of the following reasons. The transaction has been captured already, has been voided, has expired, or the capture request has used the incorrect authorization transaction ID. Also, notice that only one capture is allowed for each auth.
10407	The merchant sale total exceeds the sales cap.	The transaction amount exceeded the per transaction limit imposed by the card issuer.
10408	An error occurred due to invalid data format.	An incorrect format was used for a field value, such as the transaction ID.
10409	A validation error occurred while processing this transaction. : . Please correct values and try processing again.	Where {0} will be substituted with any of the following error messages: Incorrect Card Verification Code Incorrect Zip Code Incorrect Street Address Incorrect Street Address and Zip Code Card Verification Code not available Street Address and/or Zip Code not available For Example: Card Declined: {Incorrect Street Address}. Please correct values and try to process the transaction again.
10413	More than one batch is open. Please provide BatchID.	The MerchantBatchClose request is missing the BatchID.
10500	A general error occurred at the credit card processing gateway.	An unknown server error occurred that prevents processing of the qbmsxml request.
10501	A general system error occurred while processing the request.	An unknown server error occurred that prevents processing of the qbmsxml request.

(c) 2008 Intuit Inc. All rights reserved.

(c) 2008 Intuit Inc. All rights reserved.

APPENDIX B
SIGNON REQUESTS AND RESPONSES XML

Each qbmsXML document that is posted to the QBMS server must include a fully constructed signon message. The responses returned from the server therefore also include a signon response.

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

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

```
<?xml version="1.0" ?>
<!--
-->
<!--
======================================================================== -->
<!-- INTUIT CONFIDENTIAL.
-->
<!-- Copyright 2001-2002 Intuit Inc. All rights reserved.
-->
<!-- Use is subject to the terms specified at:
-->
<!--          http://developer.intuit.com/legal/devsite_tos.html
-->
<!--
-->
<!--
======================================================================== -->
<!--
-->
<!-- Sample data for dtd: qbmsxmlso20.dtd
-->
<!--
-->
<!-- This dtd contains requests/responses for the Signon message set.
-->
<!--
-->
<!-- Comments use the following abbreviations:
-->
<!--   QBD stands for the QuickBooks Desktop SDK
-->
<!--   QBMS stands for the QBMS SDK                      -->
<!--
-->
<!-- Message set Signon contains the following requests and responses:
-->
<!--
```


(c) 2008 Intuit Inc. All rights reserved.

```
-->
<!--    Signon (AppCert, Desktop and Ticket)
-->
<!--
-->
<!-- This means that Signon has, for example, 3 separate requests.
-->
<!-- They are: SignonAppCert, SignonDesktop and SignonTicket
-->
<!--SignonAppCert is for web apps only -->
<!--
-->
<QBMSXML>
  <SignonMsgsRq>
    <!-- SignonAppCertRq contains 1 optional attribute: 'requestID' -->
    <SignonAppCertRq requestID = "UUIDTYPE">
      <ClientDateTime>DATETIMETYPE</ClientDateTime>
      <ApplicationLogin>STRTYPE</ApplicationLogin>
      <ConnectionTicket>STRTYPE</ConnectionTicket>
      <InstallationID>IDTYPE</InstallationID> <!-- opt -->
      <Language>STRTYPE</Language> <!-- opt -->
      <AppID>STRTYPE</AppID> <!-- opt -->
      <AppVer>STRTYPE</AppVer> <!-- opt -->
    </SignonAppCertRq>
    <!--SignonDesktopRq contains 1 optional attribute:    'requestID' -->
    <SignonDesktopRq requestID = "UUIDTYPE">
      <ClientDateTime>DATETIMETYPE</ClientDateTime>
      <ApplicationLogin>STRTYPE</ApplicationLogin>
      <ConnectionTicket>STRTYPE</ConnectionTicket>
      <InstallationID>IDTYPE</InstallationID> <!-- opt -->
      <Language>STRTYPE</Language> <!-- opt -->
      <AppID>STRTYPE</AppID> <!-- opt -->
      <AppVer>STRTYPE</AppVer> <!-- opt -->
    </SignonDesktopRq>
    <!-SignonTicketRq contains 1 optional attribute: 'requestID'>
    <SignonTicketRq requestID = "UUIDTYPE">
      <ClientDateTime>DATETIMETYPE</ClientDateTime>
      <SessionTicket>STRTYPE</SessionTicket>
      <AuthID>IDTYPE</AuthID> <!-- opt -->
      <InstallationID>IDTYPE</InstallationID> <!-- opt -->
      <Language>STRTYPE</Language> <!-- opt -->
      <AppID>STRTYPE</AppID> <!-- opt -->
      <AppVer>STRTYPE</AppVer> <!-- opt -->
    </SignonTicketRq>
  </SignonMsgsRq>
  <SignonMsgsRs>
      <!-- SignonAppCertRs contains 4 attributes -->
      <!--    'requestID' is optional -->
      <!--    'statusCode' is required -->
      <!--    'statusSeverity' is required -->
      <!--    'statusMessage' is optional -->
      <SignonAppCertRs requestID = "UUIDTYPE"
            statusCode = "INTTYPE"
            statusSeverity = "STRTYPE" statusMessage = "STRTYPE">
        <ServerDateTime>DATETIMETYPE</ServerDateTime>
        <SessionTicket>STRTYPE</SessionTicket> <!-- opt -->
      </SignonAppCertRs>
```


(c) 2008 Intuit Inc. All rights reserved.

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

SignonDesktopRq

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

Table B-1 SignonDesktopRq Values

Element Name	Description
ClientDateTime	The ClientDateTime is the current system time, which provides a timestamp for the signon request.
	Required.
ApplicationLogin	The AppLogin value returned to you when you registered your application with IDN.
	Required only for SignonDesktopRq.
ConnectionTicket	The connection ticket created by the user at the QBMSlogin site and pasted into your application.
	Required.
InstallationID	Normally not used.
Language	Normally not used.
AppID	Normally not used.
AppVer	Normally not used.

(c) 2008 Intuit Inc. All rights reserved.

SignonDesktopRs

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

Table B-2 SignonDesktopRs Values

Element Name	Description
ServerDateTime	The current system time at the QuickBooks Online Site server taken at the time the request was processed.
StatusCode	The value indicating success or failure, with the value 0 indicating success. There is a failure, the code indicates the nature of the failure, with various values possibly returned. A status code of 2020 means session authentication is required See Appendix A for a list of possible values.
SessionTicket	If the SignonDesktopRq was successful, a session ticket is returned in the response. You don't have to do anything with this ticket.
StatusSeverity	Indicates level of failure.
StatusMessage	Provides a user-readable indication of the failure.

SignonTicketRq

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

Table B-3 SignonTicketRq Values

Element Name	Description
ClientDateTime	The ClientDateTime is the current system time, which provides a timestamp for the signon request. Required.
ApplicationLogin	Normally not used.
SessionTicket	The session ticket.
InstallationID	Normally not used.
Language	Normally not used.
AppID	Normally not used..
AppVer	Normally not used.

(c) 2008 Intuit Inc. All rights reserved.

APPENDIX C
THE QBMSLIB CONVENIENCE LIBRARY

This appendix describes QBMSLib, which is a fully functional .NET class library wrapped around qbmsXML. The purpose of this library is to provide an alternative to XML in sending requests and processing responses.

We provide QBMSLib in source form as a sample to allow you to easily customize it to your specific needs. You can think of it as a form of "open source" foundation classes for QuickBooks Merchant Services.

Structure

QBMSLib implements the Intuit.QBMSLib namespace in which there is one primary class (QBMSRequestor), six response encapsulation classes (one for each request supported by QBMS, for example AuthResponse), one interface (RequestSender) for POSTing requests to QBMS, and one implementation of that interface (SimpleRequestSender) which provides sufficient functionality for desktop-based applications to work with QBMS.

To use this library for a hosted web application, you will need to supply another impl of the RequestSender interface that is configured to supply a client certificate. (See the sample program IDNRequestor included in the QB SDK samples subdirectory \samples\qbms\c-sharp\IDNRequestor.)

The QBMSRequestor class is the workhorse of the library, in addition to the constructor (which takes parameters to capture your registered AppID, etc.) this class implements six methods, one for each request supported by QBMS (for example the SendAuthRequest method sends a CustomerCreditCardAuthRq message to QBMS) and the return type of each method is class which encapsulates all the fields returned by that request as well as the result code and result message from QBMS.

For example, to send a CustomerCreditCardAuthRq to QBMS we would use the following code (note that the parameters correspond to the fields that the OSR indicates could be supplied to the CustomerCreditCardAuthRq message:

```
QBMSRequestor requestor;
…
AuthResponse resp
Resp = Requestor.SendAuthRequest(CCNum,ExpireMonth,
      ExpireYear,Amount,
      CardHolderName,BillAddr,
      BillZip,
      CommercialCardCode,
      SalesTaxAmount,
      CardSecurityCode,
      IsECommerce, IsRecurring);
```


(c) 2008 Intuit Inc. All rights reserved.

And we could then view the fields of the response (as shown by the OSR) from the Resp object:

```
int result = Resp.ResultCode;
string transID = Resp.CreditCardTransID;
```

Reference

Interfaces

Type	Summary
RequestSender	The RequestSender interface is used by the QBMSRequestor to send qbmsXML requests to QuickBooks merchant services. Any object which implements this interface will work with QBMSRequestor, including the QBMSLib.SimpleRequestSender which just does a standard HTTPS POST, with no client certificate presentation. The QBMSDonorSample sample shows the use of an EnterpriseServices COM+ component to send requests with a client certificate for web solutions.

Enumerations

Type	Summary
QBMSAppType	Used in the QBMSRequestor constructor to indicate if the application using QBMSRequestor is registered as a web or desktop application.

Classes

Type	Summary
AuthResponse	Encapsulates the parsing of the XML response from QBMS and making that data available to clients. Contains the following properites for authorization transaction ID, authorization code, result of address validation for street and zip data result of security code checking.
CaptureResponse	Encapsulates the parsing of the XML response from QBMS and making that data available to clients. Contains the following properites for authorization transaction ID, authorization code, result of address validation for street and zip data result of security code checking.

(c) 2008 Intuit Inc. All rights reserved.

ChargeResponse	Encapsulates the parsing of the XML response from QBMS and making that data available to clients. Contains the following properites for authorization transaction ID, authorization code, result of address validation for street and zip data result of security code checking.
QBMSRequestor	The main class for this library, provide information regarding your app's registration with QBMS (from http://appreg.intuit.com) when you construct this class. There are methods for sending each supported QBMS request with the respose returned as a unique object type containing properties corresponding to the data returned by QBMS.
RefundResponse	Encapsulates the parsing of the XML response from QBMS and making that data available to clients. Contains the following properites for refund transaction ID, authorization code, result of address validation for street and zip data result of security code checking.
SimpleRequestSender	The SimpleRequestSender class implements the RequestSender interface and simply does a POST of the request to the given URL, no client certificate set up, etc. This is fine for desktop-based applications.
VoiceAuthResponse	Encapsulates the parsing of the XML response from QBMS and making that data available to clients. Contains the following properites for authorization transaction ID, authorization code, result of address validation for street and zip data result of security code checking.
VoidResponse	Encapsulates the parsing of the XML response from QBMS and making that data available to clients. Contains the following properites for authorization transaction ID, authorization code, result of address validation for street and zip data result of security code checking.

RequestSender Interface

Summary

```
public interface RequestSender
```

The RequestSender interface is used by the QBMSRequestor to send qbmsXML requests to QuickBooks merchant services. Any object which implements this interface will work with QBMSRequestor, including the QBMSLib.SimpleRequestSender which just does a standard HTTPS POST, with no client certificate presentation. The QBMSDonorSample sample shows the use of an EnterpriseServices COM+ component to send requests with a client certificate for web solutions.

Method Members

public String SendRequest(String URL, String request)

send an XML request to qbMS and return the XML response.

Parameters:

(c) 2008 Intuit Inc. All rights reserved.

- String URL : The URL to which the request should be POSTed
- String request : The XML string to POST

QBMSAppType Enumeration

Summary

```
public enumeration QBMSAppType
```

Used in the QBMSRequestor constructor to indicate if the application using QBMSRequestor is registered as a web or desktop application.

Enumeration Members

Field	Summary
Desktop	
web	

SimpleRequestSender Class

Summary

```
public class SimpleRequestSender : Intuit.QBMSLib.RequestSender
```

The SimpleRequestSender class implements the RequestSender interface and simply does a POST of the request to the given URL, no client certificate set up, etc. This is fine for desktop-based applications. See the QBMSDonorSample for an example of a class that implements the RequestSender interface to manage client certificates for a web application.

Constructor Members

public SimpleRequestSender()

Initializes a new instance of the class.

Method Members

public String SendRequest(String URL, String request)

send an XML request to qbMS and return the XML response.

Parameters:

- String URL : The URL to which the request should be POSTed
- String request : The XML string to POST

QBMSRequestor Class

Summary

```
public class QBMSRequestor
```


(c) 2008 Intuit Inc. All rights reserved.

The main class for this library, provide information regarding your app's registration with QBMS (from http://appreg.intuit.com) when you construct this class. There are methods for sending each supported QBMS request with the respose returned as a unique object type containing properties corresponding to the data returned by QBMS.

Constructor Members

public QBMSRequestor(QBMSAppType type, String appLogin, String connTkt, String installID, String lang, String appID, String appVer, RequestSender sender, Boolean useIDNBeta)

Create a QBMSRequestor with all the information about your application registration from http://appreg.intuit.com, and other infor needed by the QBMSXML Signon blocks.

Parameters:

- QBMSAppType type : hosted or desktop
- String appLogin : registered applogin name
- String connTkt : the connection ticket to use for the current user
- String installID : installation ID to use, can be empty string
- String lang : English is only valid value at this time
- String appID : appID you got when you registered at appreg.intuit.com
- String appVer : application version string
- RequestSender sender : An object implementing the QBMSLib.RequestSender interface
- Boolean useIDNBeta : boolean indicating whether to use IDNBeta or production environment

Method Members

public AuthResponse SendAuthRequest(String CCNum, String ExpireMonth, String ExpireYear, String Amount, String CCName, String CCAddr, String CCZip, String CCCCode, String STAmount, String CSCode, Boolean IsECommerce, Boolean IsRecurring)

Send a CustomerCreditCardAuthRq Request.

Parameters:

- String CCNum : Credit card number
- String ExpireMonth : card expiration month
- String ExpireYear : card expiration year
- String Amount : amount to authorize
- String CCName : name on the card
- String CCAddr : card billing street address (may be empty string)
- String CCZip : card billing postal code (may be empty string)
- String CCCCode : commercial card code (may be empty string)
- String STAmount : sales tax amount (may be empty string)
- String CSCode : card security code (may be empty string)

(c) 2008 Intuit Inc. All rights reserved.

- Boolean IsECommerce : true if this request is part of an online transaction, false if from a telephone order, etc.
- Boolean IsRecurring : true if the card will be rebilled at regular intervals

public CaptureResponse SendCaptureRequest(String transID, String amount)

Send a request to capture (charge) a prior authorization

Parameters:

- String transID : the transaction ID from a previous Auth request
- String amount : amount to capture

public ChargeResponse SendChargeRequest(String CCNum, String ExpireMonth, String ExpireYear, String Amount, String CCName, String CCAddr, String CCZip, String CCCCode, String STAmount, String CSCode, Boolean IsECommerce, Boolean IsRecurring)

Authorize and charge a card in one swoop.

Parameters:

- String CCNum : Credit card number
- String ExpireMonth : card expiration month
- String ExpireYear : card expiration year
- String Amount : amount to authorize
- String CCName : name on the card
- String CCAddr : card billing street address (may be empty string)
- String CCZip : card billing postal code (may be empty string)
- String CCCCode : commercial card code (may be empty string)
- String STAmount : sales tax amount (may be empty string)
- String CSCode : card security code (may be empty string)
- Boolean IsECommerce : true if this request is part of an online transaction, false if from a telephone order, etc.
- Boolean IsRecurring : true if the card will be rebilled at regular intervals

public RefundResponse SendRefundRequest(String CCNum, String ExpireMonth, String ExpireYear, String Amount, String CCName, String CCCCode, String STAmount, Boolean IsECommerce)

Refund a customer credit card.

Parameters:

- String CCNum : Credit card number
- String ExpireMonth : card expiration month

(c) 2008 Intuit Inc. All rights reserved.

- String ExpireYear : card expiration year
- String Amount : amount to authorize
- String CCName : name on the card
- String CCCCode : commercial card code (may be empty string)
- String STAmount : sales tax amount (may be empty string)
- Boolean IsECommerce : true if this request is part of an online transaction, false if from a telephone order, etc.

public VoiceAuthResponse SendVoiceAuthRequest(String CCNum, String ExpireMonth, String ExpireYear, String Amount, String AuthCode, String CCCCode, String STAmount, Boolean IsECommerce)

Authorize a card with a voice authorization code

Parameters:

- String CCNum : Customer credit card number
- String ExpireMonth : Card expiration month
- String ExpireYear : Card expiration year
- String Amount : Amount of charge
- String AuthCode : voice authorization code
- String CCCCode : Commercial card code
- String STAmount : Sales tax amount
- Boolean IsECommerce :

public VoidResponse SendVoidRequest(String transID)

Void a previous card transaction

Parameters:

- String transID : the transaction ID to void

AuthResponse Class

Summary

```
public class AuthResponse
```

Encapsulates the parsing of the XML response from QBMS and making that data available to clients.

Property Members

Name	Access	Summary
AuthorizationCode : String	public	
AVSStreet : String	public	

(c) 2008 Intuit Inc. All rights reserved.

AVSZip : String	public	
CardSecurityCodeMatch : String	public	
CreditCardTransID : String	public	
ResultCode : Int32	public	
ResultMessage : String	public	

CaptureResponse Class

Summary

```
public class CaptureResponse
```

Encapsulates the parsing of the XML response from QBMS and making that data available to clients.

Property Members

Name	Access	Summary
AuthorizationCode : String	public	
CreditCardTransID : String	public	
MerchantAccountNumber : String	public	
PaymentGroupingCode : String	public	
PaymentStatus : String	public	
ReconBatchID : String	public	
ResultCode : Int32	public	
ResultMessage : String	public	
TxnAuthorizationStamp : String	public	
TxnAuthorizationTime : String	public	

ChargeResponse Class

Summary

```
public class ChargeResponse
```

Encapsulates the parsing of the XML response from QBMS and making that data available to clients.

Property Members

Name	Access	Summary
AuthorizationCode : String	public	
AVSStreet : String	public	
AVSZip : String	public	
CardSecurityCodeMatch : String	public	
CreditCardTransID : String	public	
MerchantAccountNumber : String	public	
PaymentGroupingCode : String	public	

(c) 2008 Intuit Inc. All rights reserved.

PaymentStatus : String	public	
ReconBatchID : String	public	
ResultCode : Int32	public	
ResultMessage : String	public	
TxnAuthorizationStamp : String	public	
TxnAuthorizationTime : String	public	

RefundResponse Class

Summary

```
public class RefundResponse
```

Encapsulates the parsing of the XML response from QBMS and making that data available to clients.

Property Members

Name	Access	Summary
CreditCardTransID : String	public	
MerchantAccountNumber : String	public	
PaymentGroupingCode : String	public	
PaymentStatus : String	public	
ReconBatchID : String	public	
ResultCode : Int32	public	
ResultMessage : String	public	
TxnAuthorizationStamp : String	public	
TxnAuthorizationTime : String	public	

VoidResponse Class

Summary

```
public class VoidResponse
```

Encapsulates the parsing of the XML response from QBMS and making that data available to clients.

Property Members

Name	Access	Summary
CreditCardTransID : String	public	
ResultCode : Int32	public	
ResultMessage : String	public	

VoiceAuthResponse Class

Summary

```
public class VoiceAuthResponse
```


(c) 2008 Intuit Inc. All rights reserved.

Encapsulates the parsing of the XML response from QBMS and making that data available to clients.

Property Members

Name	Access	Summary
AuthorizationCode : String	public	
CreditCardTransID : String	public	
MerchantAccountNumber : String	public	
PaymentGroupingCode : String	public	
PaymentStatus : String	public	
ReconBatchID : String	public	
ResultCode : Int32	public	
ResultMessage : String	public	
TxnAuthorizationStamp : String	public	
TxnAuthorizationTime : String	public	

(c) 2008 Intuit Inc. All rights reserved.

APPENDIX D
SUPPORTED ROOT CERTIFICATE AUTHORITIES

You can reduce risk to your project by using root CAs that are already known to work with QBMS. If you use another authority and encounter problems it may take awhile to get the issues resolved.

The following root CA providers are known to work with QBMS. (Each group of owner/issuers are one item.)

Baltimore CyberTrust

```
Owner: CN=Baltimore CyberTrust Code Signing Root, OU=CyberTrust, O=Baltimore, C=IE
Issuer: CN=Baltimore CyberTrust Code Signing Root, OU=CyberTrust, O=Baltimore, C=IE
```

```
Owner: CN=Baltimore CyberTrust Root, OU=CyberTrust, O=Baltimore, C=IE
Issuer: CN=Baltimore CyberTrust Root, OU=CyberTrust, O=Baltimore, C=IE
```

Equifax

```
Owner: OU=Equifax Secure Certificate Authority, O=Equifax, C=US
Issuer: OU=Equifax Secure Certificate Authority, O=Equifax, C=US
```

```
Owner: CN=Equifax Secure eBusiness CA-1, O=Equifax Secure Inc., C=US
Issuer: CN=Equifax Secure eBusiness CA-1, O=Equifax Secure Inc., C=US
```

```
Owner: CN=Equifax Secure Global eBusiness CA-1, O=Equifax Secure Inc., C=US
Issuer: CN=Equifax Secure Global eBusiness CA-1, O=Equifax Secure Inc., C=US
```

GTE CyberTrust

```
Owner: CN=GTE CyberTrust Global Root, OU="GTE CyberTrust Solutions, Inc.", O=GTE Corporation,
C=US
Issuer: CN=GTE CyberTrust Global Root, OU="GTE CyberTrust Solutions, Inc.", O=GTE
Corporation, C=US
```


(c) 2008 Intuit Inc. All rights reserved.

Owner: CN=GTE CyberTrust Root, O=GTE Corporation, C=US
Issuer: CN=GTE CyberTrust Root, O=GTE Corporation, C=US

Owner: CN=GTE CyberTrust Root 5, OU="GTE CyberTrust Solutions, Inc.", O=GTE Corporation, C=US
Issuer: CN=GTE CyberTrust Root 5, OU="GTE CyberTrust Solutions, Inc.", O=GTE Corporation, C=US

RSA Data Security, Inc.

Owner: OU=Secure Server Certification Authority, O="RSA Data Security, Inc.", C=US
Issuer: OU=Secure Server Certification Authority, O="RSA Data Security, Inc.", C=US

Thawte

Owner: EMAILADDRESS=personal-basic@thawte.com, CN=Thawte Personal Basic CA, OU=Certification Services Division, O=Thawte Consulting, L=Cape Town, ST=Western Cape, C=ZA
Issuer: EMAILADDRESS=personal-basic@thawte.com, CN=Thawte Personal Basic CA, OU=Certification Services Division, O=Thawte Consulting, L=Cape Town, ST=Western Cape, C=ZA

Owner: EMAILADDRESS=personal-freemail@thawte.com, CN=Thawte Personal Freemail CA, OU=Certification Services Division, O=Thawte Consulting, L=Cape Town, ST=Western Cape, C=ZA
Issuer: EMAILADDRESS=personal-freemail@thawte.com, CN=Thawte Personal Freemail CA, OU=Certification Services Division, O=Thawte Consulting, L=Cape Town, ST=Western Cape, C=ZA

Owner: EMAILADDRESS=personal-premium@thawte.com, CN=Thawte Personal Premium CA, OU=Certification Services Division, O=Thawte Consulting, L=Cape Town, ST=Western Cape, C=ZA
Issuer: EMAILADDRESS=personal-premium@thawte.com, CN=Thawte Personal Premium CA, OU=Certification Services Division, O=Thawte Consulting, L=Cape Town, ST=Western Cape, C=ZA

Owner: EMAILADDRESS=premium-server@thawte.com, CN=Thawte Premium Server CA, OU=Certification Services Division, O=Thawte Consulting cc, L=Cape Town, ST=Western Cape, C=ZA
Issuer: EMAILADDRESS=premium-server@thawte.com, CN=Thawte Premium Server CA, OU=Certification Services Division, O=Thawte Consulting cc, L=Cape Town, ST=Western Cape, C=ZA

(c) 2008 Intuit Inc. All rights reserved.

Owner: EMAILADDRESS=server-certs@thawte.com, CN=Thawte Server CA, OU=Certification Services Division, O=Thawte Consulting cc, L=Cape Town, ST=Western Cape, C=ZA
Issuer: EMAILADDRESS=server-certs@thawte.com, CN=Thawte Server CA, OU=Certification Services Division, O=Thawte Consulting cc, L=Cape Town, ST=Western Cape, C=ZA

UTN-USERFirst

Owner: CN=AddTrust External CA Root, OU=AddTrust External TTP Network, O=AddTrust AB, C=SE
Issuer: CN=UTN-USERFirst-Hardware, OU=http://www.usertrust.com, O=The USERTRUST Network, L=Salt Lake City, ST=UT, C=US

Owner: CN=UTN-USERFirst-Hardware, OU=http://www.usertrust.com, O=The USERTRUST Network, L=Salt Lake City, ST=UT, C=US
Issuer: CN=UTN-USERFirst-Hardware, OU=http://www.usertrust.com, O=The USERTRUST Network, L=Salt Lake City, ST=UT, C=US

Owner: CN=UTN-USERFirst-Network Applications, OU=http://www.usertrust.com, O=The USERTRUST Network, L=Salt Lake City, ST=UT, C=US
Issuer: CN=UTN-USERFirst-Network Applications, OU=http://www.usertrust.com, O=The USERTRUST Network, L=Salt Lake City, ST=UT, C=US

Valicert

Owner: EMAILADDRESS=info@valicert.com, CN=http://www.valicert.com/, OU=ValiCert Class 2 Policy Validation Authority, O="ValiCert, Inc.", L=ValiCert Validation Network
Issuer: EMAILADDRESS=info@valicert.com, CN=http://www.valicert.com/, OU=ValiCert Class 2 Policy Validation Authority, O="ValiCert, Inc.", L=ValiCert Validation Network

VeriSign

Owner: OU=Class 1 Public Primary Certification Authority, O="VeriSign, Inc.", C=US
Issuer: OU=Class 1 Public Primary Certification Authority, O="VeriSign, Inc.", C=US

(c) 2008 Intuit Inc. All rights reserved.

Owner: OU=Class 2 Public Primary Certification Authority, O="VeriSign, Inc.", C=US
Issuer: OU=Class 2 Public Primary Certification Authority, O="VeriSign, Inc.", C=US

Owner: OU=Class 3 Public Primary Certification Authority, O="VeriSign, Inc.", C=US
Issuer: OU=Class 3 Public Primary Certification Authority, O="VeriSign, Inc.", C=US

Owner: OU=Class 4 Public Primary Certification Authority, O="VeriSign, Inc.", C=US
Issuer: OU=Class 4 Public Primary Certification Authority, O="VeriSign, Inc.", C=US

Owner: OU=www.verisign.com/CPS Incorp.by Ref. LIABILITY LTD.(c)97 VeriSign, OU=VeriSign International Server CA - Class 3, OU="VeriSign, Inc.", O=VeriSign Trust Network
Issuer: OU=Class 3 Public Primary Certification Authority, O="VeriSign, Inc.", C=US

(c) 2008 Intuit Inc. All rights reserved.