This paper was originally presented at the Southwest Fox conference in Mesa, Arizona in October, 2008. http://www.swfox.net

Automating QuickBooks with QODBC

Rick Borup
Information Technology Associates
701 Devonshire Drive, Suite 127
Champaign, IL 61820
Email: rborup@ita-software.com
Web: www.ita-software.com
Blog: http://rickborup.com/blog
Twitter: rickborup

Overview

Developers who create software solutions for small and medium-sized businesses are often asked to integrate their products with the popular QuickBooks accounting system from Intuit. While Intuit provides the QuickBooks SDK, a free software development kit for this purpose, the SDK can be somewhat challenging to work with. QODBC is a commercial product that offers an alternative to the QuickBooks SDK by enabling access to QuickBooks data files through ODBC using standard SQL commands. This session covers the installation, configuration, and use of QODBC to automate QuickBooks from Visual FoxPro applications, while also exploring some of the intricacies of the QuickBooks database schema itself.

What is QODBC?

QODBC is an ODBC driver for accessing QuickBooks data files. Its full name is the QODBC Driver for QuickBooks. QODBC is a commercial product from an Australian company named FLEXQuarters.com, LLC. The products installs a DLL and supporting files on your computer. Once installed and configured, you can use the QODBC driver to programmatically read and write data to QuickBooks databases.

Why would you want it?

The main reason you would want to use the QODBC Driver for QuickBooks is that it enables you to read and write QuickBooks data using standard SQL commands. Most Visual FoxPro developers are already quite familiar and comfortable with SQL syntax, so this puts QuickBooks data within easy reach without having to learn a completely different approach such as that used by the QuickBooks SDK.

Using QODBC, you can interact with QuickBooks data using familiar SQL statements such as:

```
SELECT * FROM customer
SELECT Balance FROM account WHERE AccountNumber = '1010'
SELECT SUM( CreditAmount) AS CreditAmout FROM VendorCredit
INSERT INTO OtherName ( Name) VALUES ( 'SomeName')
```

How to get it

The QODBC Driver for QuickBooks is available for download and purchase from the company's website at www.qodbc.com. A 30-day evaluation is available so you can try it out before buying.

The current version of QODBC is version 8.0. QODBC versions are tied to versions of QuickBooks: QODBC version 8.0 works with QuickBooks 2008, while the earlier QODBC version 7.0 works with QuickBooks 2007. If you buy a license for QODBC 7.0 for use with QuickBooks 2007 and then upgrade to QuickBooks 2008, you will also need to upgrade your QODBC license to version 8.0.

QODBC Editions

There are three editions of the QODBC Driver for QuickBooks

- The Desktop Edition Read Only Driver is licensed for use on a single computer and provides read-only access to QuickBooks.
- The Desktop Edition Read Write Driver is licensed for use on a single computer and provides both read and write access to QuickBooks
- The Server Edition Read Write Driver is designed for multi-user applications running on the Windows Server family of operating systems. It can also be used on a Web server to provide interactive access to QuickBooks data for Web applications.

The QODBC Driver for QuickBooks supports most versions of QuickBooks from 2002 through 2008. QODBC supports the USA versions of QuickBooks Pro, Premier, Enterprise, Simple Start, and Online editions. It also supports the UK, Canadian, and Australian versions of QuickBooks Pro, as well as others.

Installing QODBC

The Enterprise edition of QuickBooks comes with the QODBC Read Only Driver, so if you happen to have that product you already have all you need to get started. Otherwise, I'd recommend you download a 30-day evaluation version of the Read Write Driver to install and use as you work through the examples in the session.

Installation of QODBC is quite straightforward: simply download QODBC.EXE from the company's website and run it on your computer. The default installation location is C:\Program Files\QODBC Driver for QuickBooks. The product also installs a DLL and related files to C:\Windows\System32.[1] These files include:

- fqqdb32.dll, the main DLL at approximately 8MB in size
- xerces-com.dll, an XML parser for COM
- fqqbudsn.exe, fqq6.exe, and fqqbc32.exe

Once installed, you have a QODBC Driver for QuickBooks entry on your Start menu. This menu pad includes an item for configuring QODBC data sources, which you will use frequently while working with QODBC.

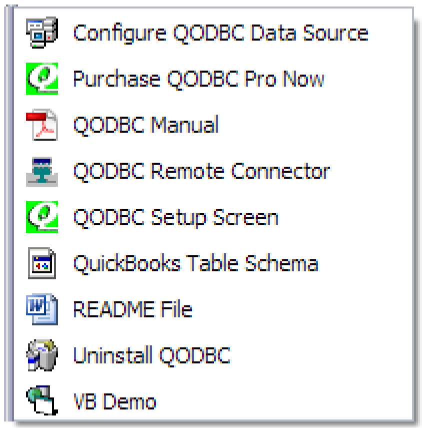


Figure 1: The QODBC Driver for QuickBooks menu includes an item for configuring QODBC data sources.

How QODBC accesses QuickBooks

When you use QuickBooks in the conventional manner as a desktop application, you as the end user interact with the QuickBooks program and the QuickBooks program in turn interacts with the QuickBooks database. The QuickBooks database file itself is locked down and cannot be

[1] Based on QODBC version 7.0 on Windows XP.

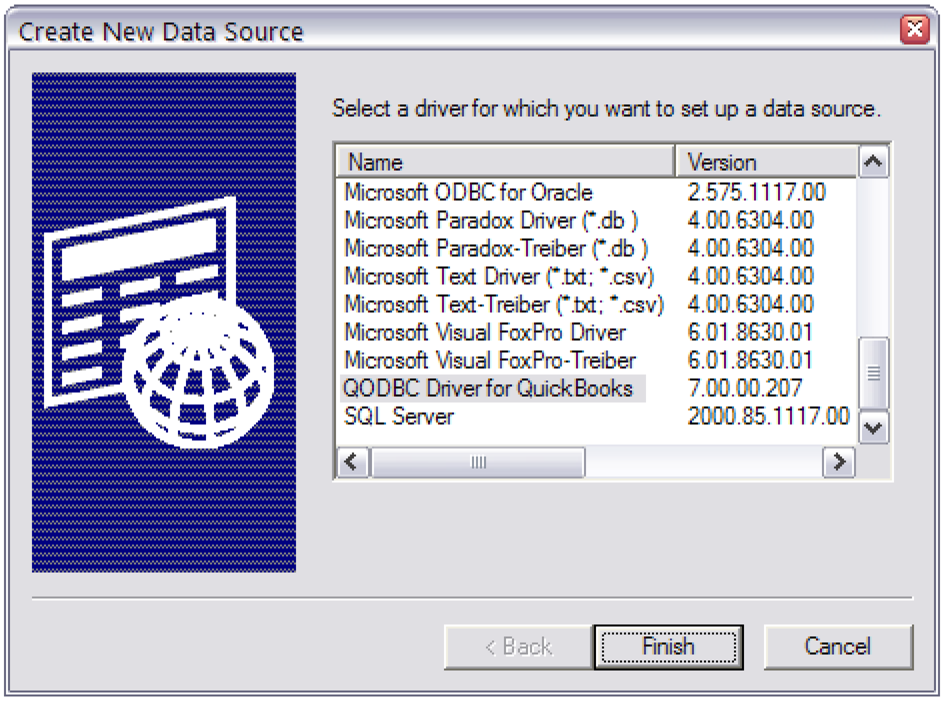
accessed directly from outside of the QuickBooks application. The reasons for this should be obvious: how much confidence could there be in a company's books if anyone or anything could easily open and access the database?

For the same reasons, QODBC cannot directly access a QuickBooks database either. Just as you must use the QuickBooks program to access a QuickBooks database, QODBC must also go through QuickBooks to access a QuickBooks database. What this means is that to use QODBC, QuickBooks itself must also be installed on the user's machine.

The QODBC driver works by creating XML queries in a format called qbXML, which QuickBooks understands. QODBC takes your queries, formats them as qbXML, passes them to QuickBooks, and returns the results. You can actually see this happening if you dig into the log files QODBC can create for debugging purposes.

Configuring QODBC

The first step in configuring a computer to use QODBC is to create a Data Source Name (DSN). Select the QODBC Driver for QuickBooks from the list of available drivers, and click Finish.

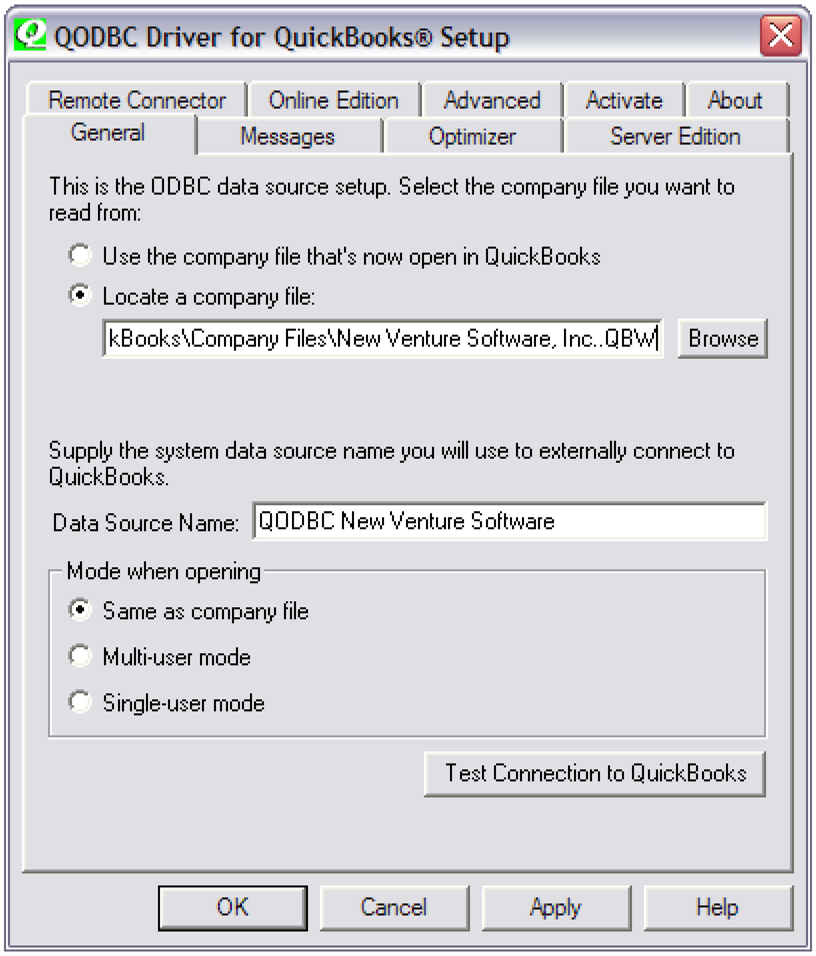


Select a driver for which you want to set up a data source.

Name	Version
Microsoft ODBC for Oracle	2.575.1117.00
Microsoft Paradox Driver (*.db)	4.00.6304.00
Microsoft Paradox-Treiber (*.db)	4.00.6304.00
Microsoft Text Driver (*.txt; *.csv)	4.00.6304.00
Microsoft Text-Treiber (*.txt; *.csv)	4.00.6304.00
Microsoft Visual FoxPro Driver	6.01.8630.01
Microsoft Visual FoxPro-Treiber	6.01.8630.01
QODBC Driver for QuickBooks	7.00.00.207
SQL Server	2000.85.1117.00

Figure 2: Create a new DSN using the QODBC Driver for QuickBooks.

When you configure the DSN, you can choose whether QODBC should always open a specific company file or whether it should use whatever company file is open in QuickBooks at runtime. The examples for this session are designed to work with the QuickBooks company file for a hypothetical company named New Venture Software, Inc. Therefore, I want the DSN to use that company file and no other. In Figure 3, you can see where this choice has been made by marking the “Locate a company file” option button and using the Browse button to specify the location of the New Venture Software company file.


QODBC Driver for QuickBooks® Setup

This is the ODBC data source setup. Select the company file you want to read from:

( ) Use the company file that's now open in QuickBooks
[✓ Locate a company file:]
kBooks\Company Files\New Venture Software, Inc..QBW

Supply the system data source name you will use to externally connect to QuickBooks.

Data Source Name: QODBC New Venture Software

Mode when opening
[✓ Same as company file]
( ) Multi-user mode
( ) Single-user mode

Figure 3: When you configure a DSN for QODBC, you choose whether to use the company file that's open in QuickBooks at runtime or whether to always open a specific company file.

The next step is to name the DSN. If the DSN is configured to access a specific company file, I recommend including the name of the company in the name of the DSN. This makes it easier for you to be sure you're connecting to the desired company when you set up the ODBC connection in your program. In Figure 3, you can see the DSN is named QODBC New Venture Software.

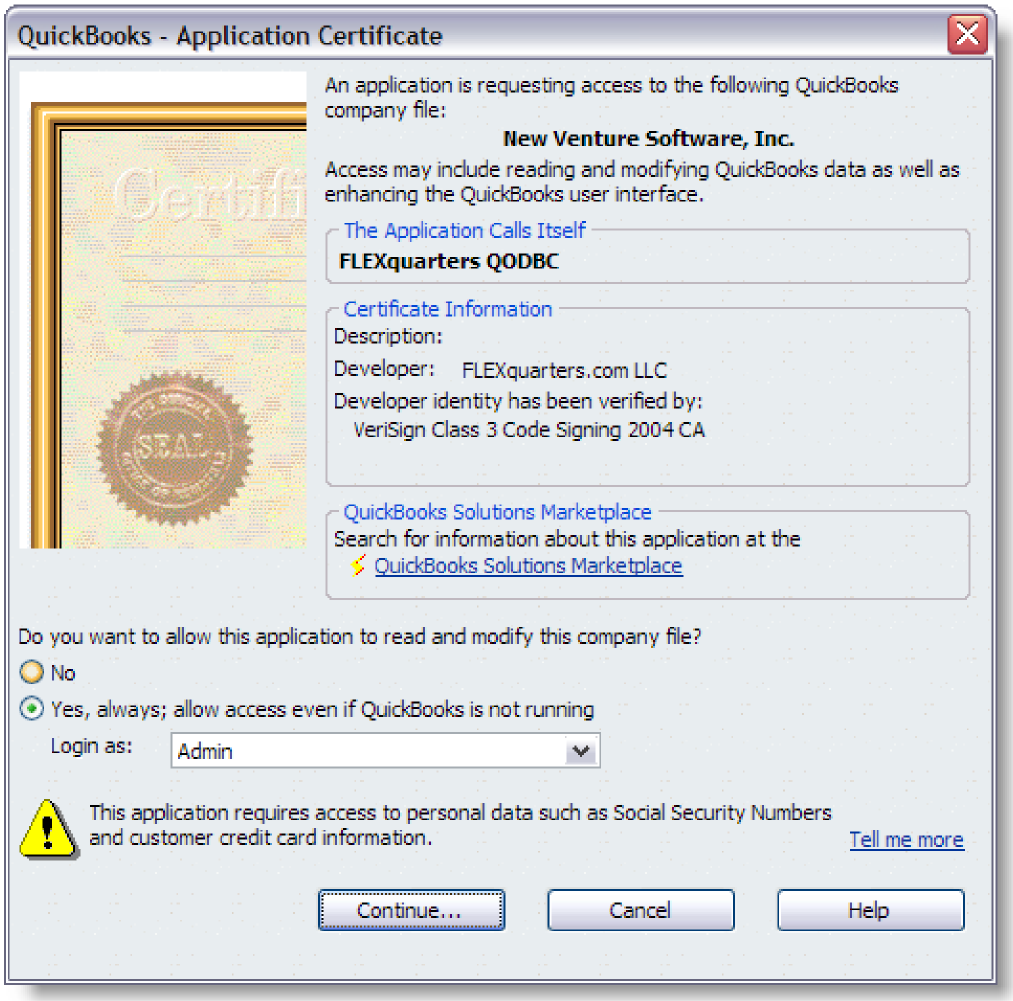
The bottom portion of the QODBC setup dialog enables you to choose the mode to use when opening the QuickBooks company file. QuickBooks can open a company file either in multi-user mode for shared access or in single-user mode for exclusive access. In QuickBooks, some tasks can be done only in single-user mode, but it is unlikely your application will need to do any of those things programmatically so you'll normally want to choose multi-user mode. If you want QODBC to use whatever mode the QuickBooks company file happens to be in at runtime, choose Same as company file.

There are several other pages in the QODBC setup dialog, but you don't have to be concerned about them at this point. In most cases, establishing the settings on the General page is all it take to enable the connection to QuickBooks.

Use the Test Connection to QuickBooks button at the bottom of the General page to test the new connection. This launches QuickBooks, if it is not already running. By default, QuickBooks does not allow external applications to access a company file, so at this point you need to configure QuickBooks to allow access by QODBC.

Configuring QuickBooks for QODBC

When QODBC connects to a company file for the first time, QuickBooks prompts you to allow access. When the Application Certificate dialog appears as shown in Figure 4, select the second option button to allow QODBC to access the company file even if QuickBooks is not already running. If the company file is set up to require logging in with a username, select the username QODBC should use to login.


QuickBooks - Application Certificate

An application is requesting access to the following QuickBooks company file:

New Venture Software, Inc.

Access may include reading and modifying QuickBooks data as well as enhancing the QuickBooks user interface.

The Application Calls Itself

FLEXquarters QODBC

Certificate Information

Description:

Developer: FLEXquarters.com LLC

Developer identity has been verified by:

VeriSign Class 3 Code Signing 2004 CA

QuickBooks Solutions Marketplace

Search for information about this application at the QuickBooks Solutions Marketplace

Do you want to allow this application to read and modify this company file?

No [✓ Yes, always; allow access even if QuickBooks is not running]

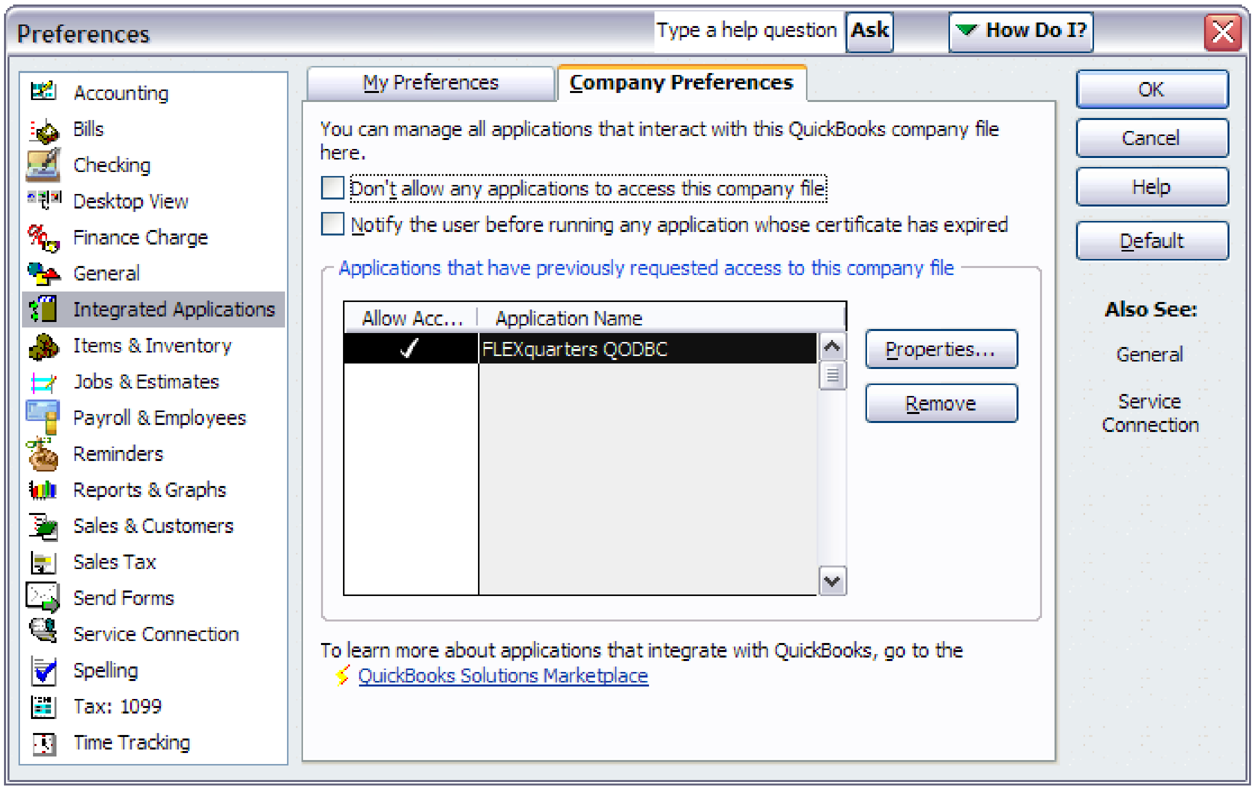
Login as: Admin

This application requires access to personal data such as Social Security Numbers and customer credit card information.

Figure 4: The first time you connect to a company file using QODBC, you need to tell QuickBooks to allow access.

Note that the name of the application requesting access is FLEXquarters QODBC. Confirm that the correct company file name is shown at the top, then click the Continue button to complete the configuration.

With the company file now open in QuickBooks, you can review the connection information you just configured by opening the QuickBooks Preferences dialog from the Edit | Preferences menu, selecting Integrated Applications and choosing the Company Preferences page, as shown in Figure 5.



You can manage all applications that interact with this QuickBooks company file here.

☐ Don't allow any applications to access this company file

☐ Notify the user before running any application whose certificate has expired

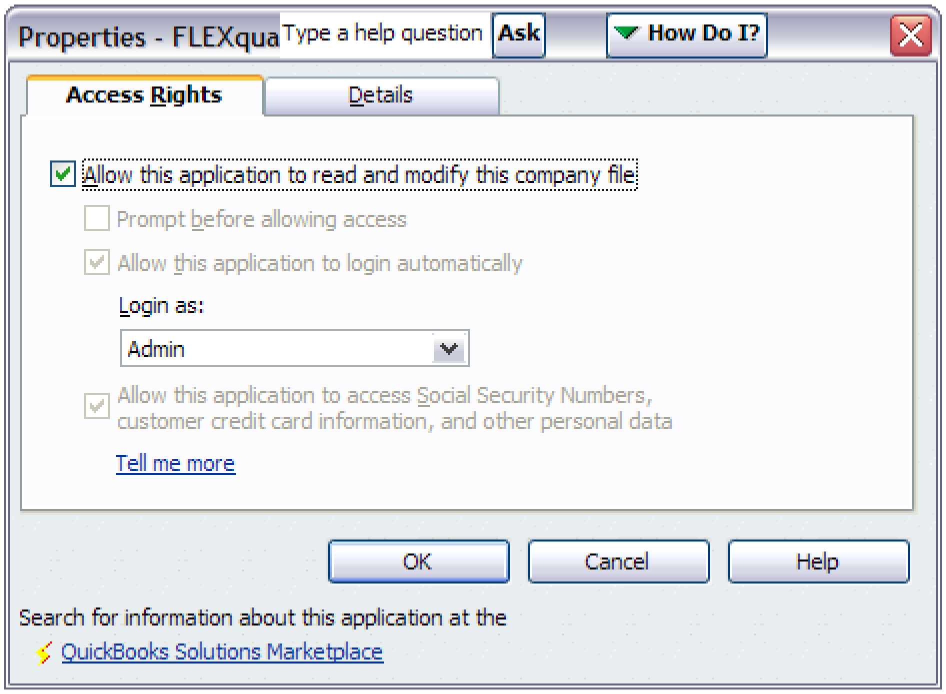
Applications that have previously requested access to this company file

Allow Acc...	Application Name
✓	FLEXquarters QODBC

To learn more about applications that integrate with QuickBooks, go to the QuickBooks Solutions Marketplace

Figure 5: The authorization for QODBC to open the QuickBooks company file can be found under Integrated Applications in the QuickBooks Preferences dialog.

Select the FLEXquarters QODBC application name in the list and click the Properties button to view the access rights for this application. If necessary, the settings can be edited from the Properties dialog, as illustrated in Figure 6.

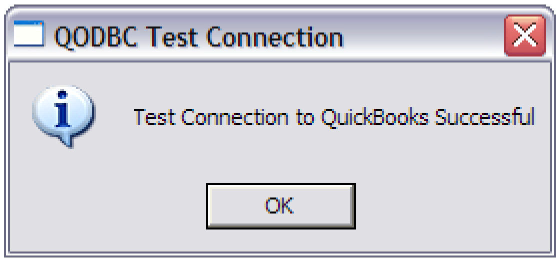


☑ Allow this application to read and modify this company file
☐ Prompt before allowing access
☑ Allow this application to login automatically
Login as:
Admin
☑ Allow this application to access Social Security Numbers, customer credit card information, and other personal data

Search for information about this application at the QuickBooks Solutions Marketplace

Figure 6: The access rights can be edited from the Properties dialog.

With the DSN configured and the appropriate access rights granted, the Test Connection process should now be showing a dialog indicating the test connection was successful.



Test Connection to QuickBooks Successful

Figure 7: This message tells you QODBC successfully connected to the QuickBooks company file.

If the connection was unsuccessful, you can usually find out what's wrong by checking the QODBC log file. The Messages page of the QODBC Setup dialog provides access to this file. The Messages page is discussed in the Troubleshooting and Debugging section of this paper.

Using QODBC with Visual FoxPro

Once you've created a DSN, configured the QODBC Driver, and configured QuickBooks to allow the connection, you're ready to start using QODBC with Visual FoxPro. As noted earlier, QODBC enables to you interact with a QuickBooks company file using standard SQL statements, just as you would with any other back-end database.

There are basically five steps involved in using QODBC programmatically. Following is an overview of these steps, along with some sample syntax for implementing them in Visual FoxPro.

- Establish a connection

```
lnConnHandle = SQLSTRINGCONNECT( "dsn='Your QODBC DSN Name'; uid='Admin'; pwd='admin'")
```

- Create a SQL statement

```
lcSQL = "SELECT SUM( AmountDue) as AmountDue FROM Bill WHERE isPaid = 0"
```

- Run the SQL statement

```
SQLEXEC( lnConnHandle, lcSQL, "csrBills")
```

- Do something with the result

```
WAIT WINDOW "Amount due = " + STR( csrBills.AmountDue)
```

- Close the connection

```
SQLDISCONNECT( lnConnHandle)
```

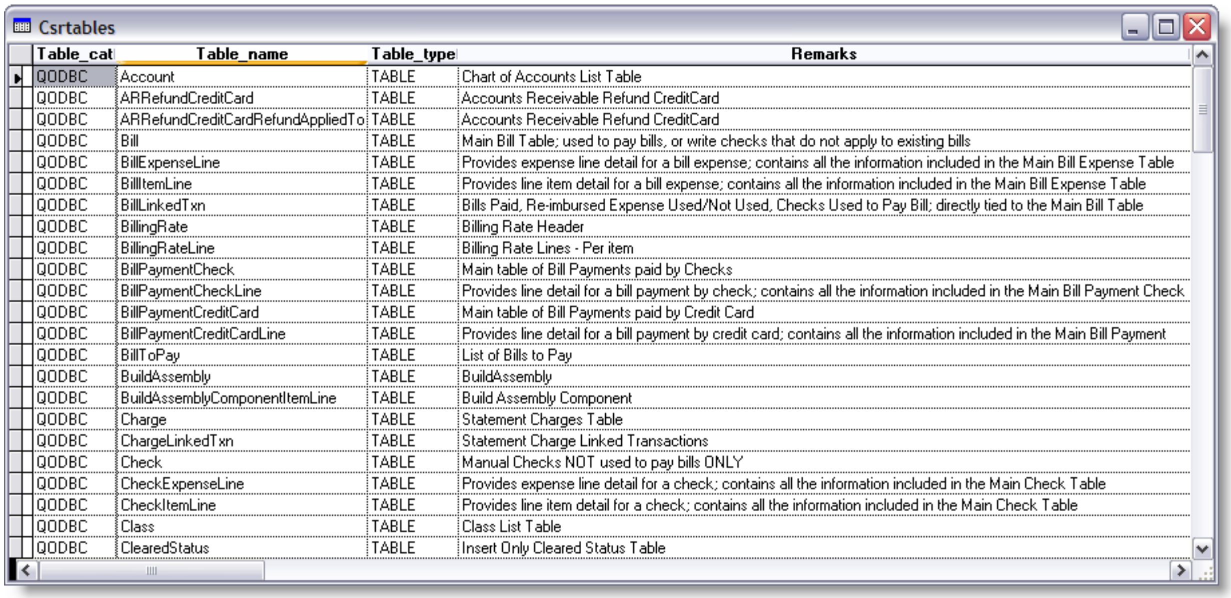
QuickBooks Database Schema

Before you can any useful work with a QuickBooks company file, you need to know something about the QuickBooks database schema. At a minimum you need to know the names of the tables and the names of the columns they contain, and which tables contain the data you are interested in. In some cases you also need to be aware of the relationship between certain tables.

The QuickBooks database schema is fairly complex and can be somewhat intimidating until you become familiar with it. One way to explore it is to get a connection to the database and then run the SQLTABLES( ) and SQLCOLUMNS( ) functions on it.

```
lnConnHandle = SQLSTRINGCONNECT("dsn=QODBC New Venture Software;uid='Admin';pwd='admin'")
SQLTABLES( lnConnHandle, 'table', 'csrTables')
BROWSE
```

This gives you a list of the tables in the database, as shown in Figure 8.


Csrtables

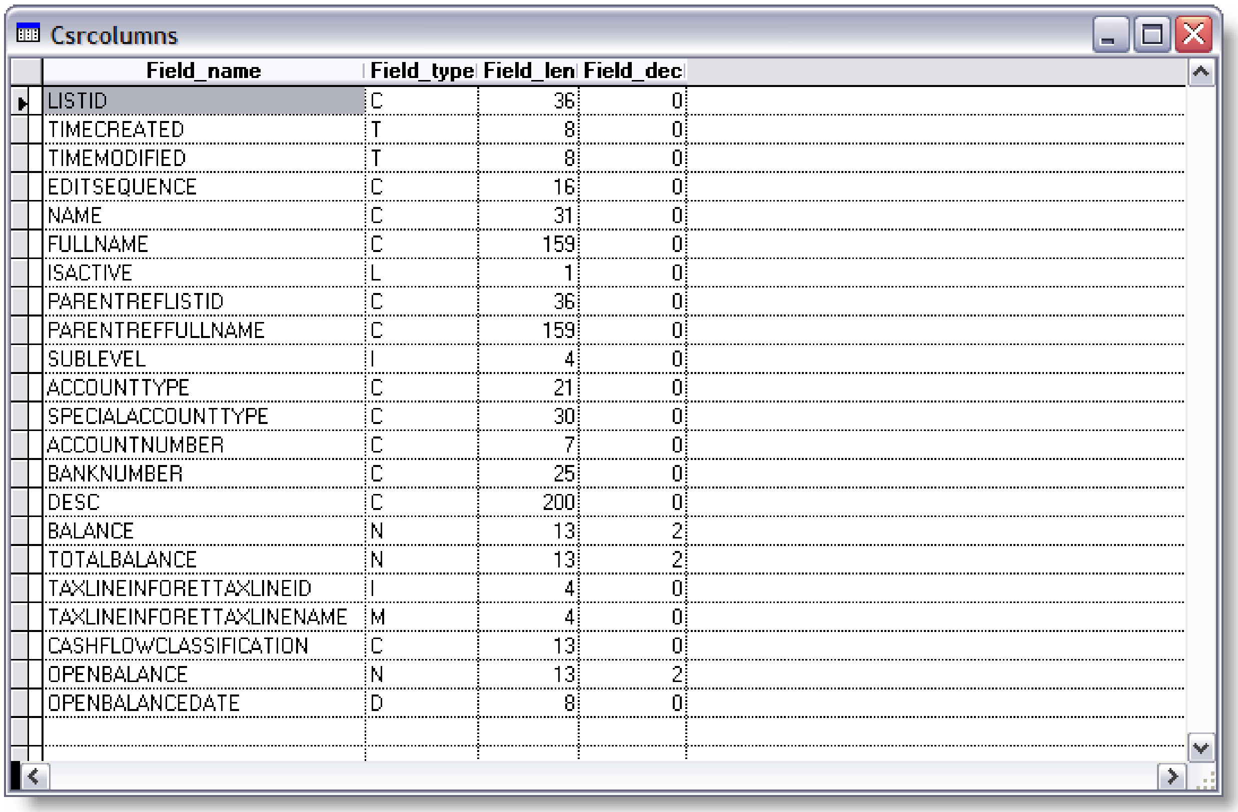
Table_cat	Table_name	Table_type	Remarks
QODBC	Account	TABLE	Chart of Accounts List Table
QODBC	ARRefundCreditCard	TABLE	Accounts Receivable Refund CreditCard
QODBC	ARRefundCreditCardRefundAppliedTo	TABLE	Accounts Receivable Refund CreditCard
QODBC	Bill	TABLE	Main Bill Table; used to pay bills, or write checks that do not apply to existing bills
QODBC	BillExpenseLine	TABLE	Provides expense line detail for a bill expense; contains all the information included in the Main Bill Expense Table
QODBC	BillItemLine	TABLE	Provides line item detail for a bill expense; contains all the information included in the Main Bill Expense Table
QODBC	BillLinkedTxn	TABLE	Bills Paid, Re-imbursed Expense Used/Not Used, Checks Used to Pay Bill; directly tied to the Main Bill Table
QODBC	BillingRate	TABLE	Billing Rate Header
QODBC	BillingRateLine	TABLE	Billing Rate Lines - Per item
QODBC	BillPaymentCheck	TABLE	Main table of Bill Payments paid by Checks
QODBC	BillPaymentCheckLine	TABLE	Provides line detail for a bill payment by check; contains all the information included in the Main Bill Payment Check
QODBC	BillPaymentCreditCard	TABLE	Main table of Bill Payments paid by Credit Card
QODBC	BillPaymentCreditCardLine	TABLE	Provides line detail for a bill payment by credit card; contains all the information included in the Main Bill Payment
QODBC	BillToPay	TABLE	List of Bills to Pay
QODBC	BuildAssembly	TABLE	BuildAssembly
QODBC	BuildAssemblyComponentItemLine	TABLE	Build Assembly Component
QODBC	Charge	TABLE	Statement Charges Table
QODBC	ChargeLinkedTxn	TABLE	Statement Charge Linked Transactions
QODBC	Check	TABLE	Manual Checks NOT used to pay bills ONLY
QODBC	CheckExpenseLine	TABLE	Provides expense line detail for a check; contains all the information included in the Main Check Table
QODBC	CheckItemLine	TABLE	Provides line item detail for a check; contains all the information included in the Main Check Table
QODBC	Class	TABLE	Class List Table
QODBC	ClearedStatus	TABLE	Insert Only Cleared Status Table

Figure 8: You can list the tables in a QuickBooks company file database by using the SQLTABLES( ) function.

Choose a table to explore, and run the SQLCOLUMNS( ) function on it. For example, you can see from Figure 8 that the Account table contains the chart of accounts, which seems like a good place to start. Run the following code to create a cursor of the columns in the Account table.

```
SQLCOLUMNS( lnConnHandle, 'Account', 'FoxPro', 'csrColumns')
```

Browsing the result enables you to see the column names and data types for the Account table.


Csrcolumns

Field_name	Field_type	Field_len	Field_dec
LISTID	C	36	0
TIMECREATED	T	8	0
TIMEMODIFIED	T	8	0
EDITSEQUENCE	C	16	0
NAME	C	31	0
FULLNAME	C	159	0
ISACTIVE	L	1	0
PARENTREFLISTID	C	36	0
PARENTREFFULLNAME	C	159	0
SUBLEVEL	I	4	0
ACCOUNTTYPE	C	21	0
SPECIALACCOUNTTYPE	C	30	0
ACCOUNTNUMBER	C	7	0
BANKNUMBER	C	25	0
DESC	C	200	0
BALANCE	N	13	2
TOTALBALANCE	N	13	2
TAXLINEINFORETTAXLINEID	I	4	0
TAXLINEINFORETTAXLINENAME	M	4	0
CASHFLOWCLASSIFICATION	C	13	0
OPENBALANCE	N	13	2
OPENBALANCEDATE	D	8	0

Figure 9: Running SQLCOLUMNS( ) on the Account table creates a cursor of the column names and data types.

Don't forget to disconnect from the QuickBooks database when you're done exploring.

```
SQLDISCONNECT( lnConnHandle)
```

Using SQLTABLES( ) and SQLCOLUMNS( ) is instructive, but somewhat tedious. Fortunately, there is another way to learn about the QuickBooks database schema. The QODBC website includes an extensive, interactive reference that documents the schema is great detail.

The tables in a QuickBooks database can be categorized according the type of data the contain, as summarized in the following table. List table, transaction tables and information tables comprise the physical tables in the database. The fourth category, View Tables, are views that pull data from one or more of the tables in the database.

Category	Examples
List tables	Account, Customer, Item, …
Transaction tables	Bill, Check, Credit Card Charge, Deposit, …
Information tables	Company, Preferences, …
View tables	Sales (custom QODBC "combination table")

You'll see this categorization when you dig into the table documentation on the QODBC website. Start from the home page at www.qodbc.com. Choose QODBC Technical Reference from the Quick Links, then click the Data Layouts link on that page. The data layouts may differ by country, so click the link for the country you're working with. In this session I'm using the documentation for the U.S. version of QuickBooks.

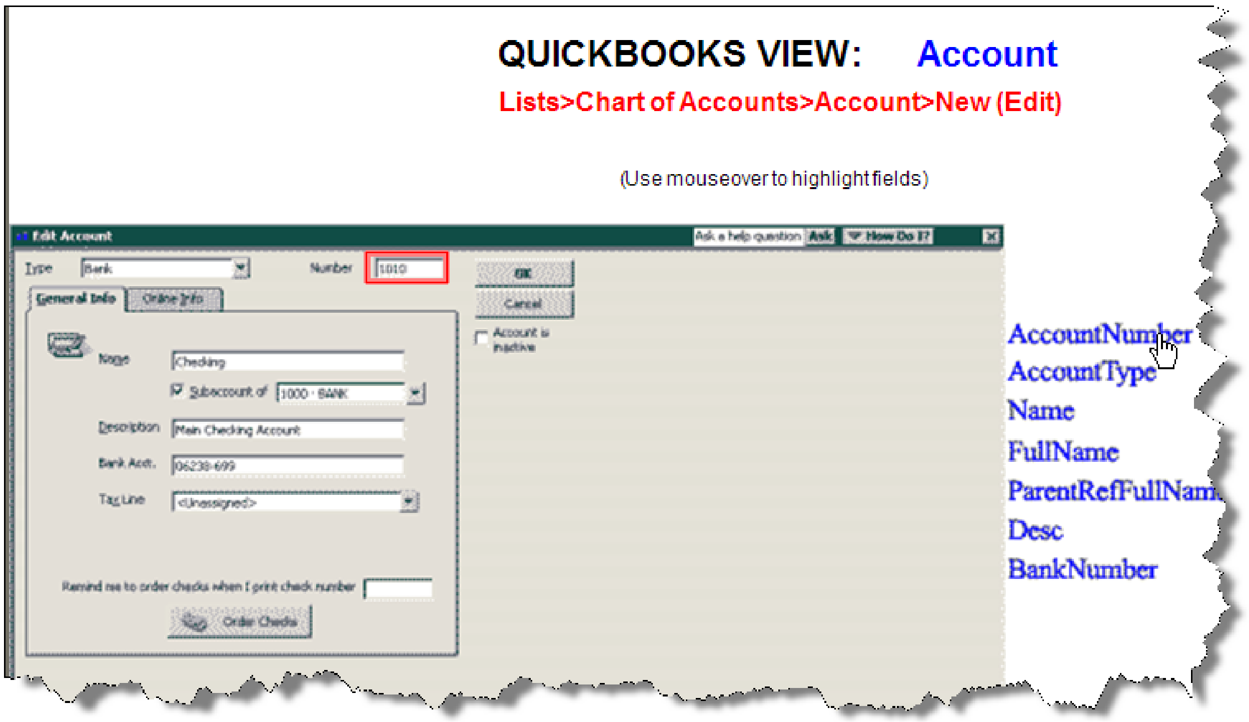
Clicking the country link takes you to a Tables and Reports Overview page, where you'll find a short paragraph and a link to each of the categories of tables mentioned above.[2] Clicking the link for a particular category takes you to a list of the tables in that category. For example, clicking the List Tables link takes you to a page that shows all of the tables in the List category, which includes the Account table.

The coolest thing they've done with this website is to provide an interactive reference for each table based on a screenshot of an actual form in QuickBooks itself. As a user of QuickBooks, you are probably already familiar with these forms. The interactive tool shows the screenshot of the form on the left and a list of column names on the right. Moving the mouse over a column name on the right automatically highlights the corresponding field on the form to the left. In this way, you can learn which fields on a form are bound to which columns in the QuickBooks database.

This is a truly excellent learning tool. Figure 10 shows the interactive webpage for the Account table from the QODBC website.[3] Hovering the mouse over the column name AccountNumber on the right highlights the corresponding field on the screen with a red outline. Similarly, hovering the mouse over the AccountType column name highlights that field on the form, and so on down the list.

[2] The direct URL to this page for US data layouts is http://doc.qodbc.com/qodbc/Qodbc_20_us.html.

[3] From http://doc.qodbc.com/qodbc/20/tables/qbview_d_account1207.html?qbviewd_id=1. Copyright © 2008 FLEXquarters.com LLC


QUICKBOOKS VIEW: Account

Lists>Chart of Accounts>Account>New (Edit)

(Use mouseover to highlight fields)

Edit Account

Type: Bank

Number: 1010

Name: Checking

☑ Subaccount of: 1000 · BANK

Description: Main Checking Account

Bank Acct.: 06238-699

Tax Line: <Unassigned>

☐ Account is inactive

Remind me to order checks when I print check number

AccountNumber

AccountType

Name

FullName

ParentRefFullName

Desc

BankNumber

Figure 10: The QODBC website provides an interactive tool showing which fields in a QuickBooks table correspond to which fields on the QuickBooks form.

Once you've gained enough familiarity with the QuickBooks database schema, you're ready to start writing code to read and write data. The session downloads include several pieces of code illustrating how to do this. Some of that code is reproduced in this paper. In other places I refer to code in the downloads even though it's not reproduced here in the paper.

Reading data from QuickBooks

This section gives several examples of how to read various types of data from a QuickBooks database using QODBC. The code presented here is taken from the program QODBC_Fetch, which is included in the session downloads. Please note that the code in this program is designed for illustration purposes only and is not intended to be run as a complete program.

As always, the first step is to establish a connection to the QuickBooks database and store the connection handle for further use. I'm using a public variable here because I want the connection handle to remain in scope throughout the session, even though I'm not running all the code as a complete program.

```
PUBLIC gnConnHandle
gnConnHandle = SQLSTRINGCONNECT("dsn=QODBC New Venture Software;uid='Admin';pwd='admin'")
```

With the connection established, you can retrieve and browse the contents of the Account table like this:

```
*-------------
* All accounts
*-------------
```

```
lcSQL = "SELECT * FROM account"
SQLEXEC( gnConnHandle, lcSQL, "csrAccounts")
BROWSE
USE IN csrAccounts
```

To retrieve only selected columns, change the SQL Select statement and run the same code.

```
lcSQL = "SELECT AccountNumber, Name, Balance FROM account"
```

Suppose you want to retrieve all unpaid bills. Knowing the appropriate table and column names, you can construct a conditional Select statement with a 'where' clause.

```
*-----------------
* All unpaid bills
*-----------------
lcSQL = "SELECT VendorRefFullName, DueDate, AmountDue, ispaid FROM Bill where isPaid = 0"
SQLEXEC( gnConnHandle, lcSQL, "csrUnpaidBills")
BROWSE
USE IN csrUnpaidBills
```

Or you might want to pull all unpaid bills that are due before a certain date.

```
*------------------------------------------
* All unpaid bills due before a certain date
*------------------------------------------
lcSQL = "SELECT VendorRefFullName, DueDate, AmountDue FROM Bill " + ;
         "where isPaid = 0 and DueDate <= {d '2008-02-29'}"
SQLEXEC( gnConnHandle, lcSQL, "csrUnpaidBillsByDate")
BROWSE
USE IN csrUnpaidBillsByDate
```

Note the format of the date in the above code. This is the format recommended by QODBC.

The following code shows how to pull all deposits from an account named 'Business Checking Account'. In the Deposit table, the full name of the account is stored in the column named DepositToAccountRefFullName.

```
*--------------------------------------------
* All deposits for 'Business Checking Account'
*--------------------------------------------
lcSQL = "SELECT TxnDate, DepositTotal as amount FROM Deposit " + ;
        "where DepositToAccountRefFullName = 'Business Checking Account' order by 1"
SQLEXEC( gnConnHandle, lcSQL, "csrDeposits")
BROWSE
USE IN csrDeposits
```

Pulling information for checks written is not quite as straightforward. Internally, QuickBooks differentiates between manual checks and checks written to pay a bill, and the two are stored in different tables. To retrieve checks data, you need to specify which table you want to access.

This code retrieves all manual check drawn on the account named 'Business Checking Account'.

```
*------------------------------------------------------
* All manual checks drawn on 'Business Checking Account'
*------------------------------------------------------
lcSQL = "SELECT TxnDate, RefNumber, Amount FROM Check " + ;
        "where AccountRefFullName = 'Business Checking Account' order by 1, 2"
SQLEXEC( gnConnHandle, lcSQL, "csrChecks")
BROWSE
```

```
USE IN csrChecks
```

The code to pull checks written to pay bills is the same except for the name of the table.

```
*--------------------------------------------------------------
* All bill payment checks drawn on 'Business Checking Account'
*--------------------------------------------------------------
lcSQL = "SELECT TxnDate, RefNumber, Amount FROM BillPaymentCheck " + ;
        "where BankAccountRefFullName = 'Business Checking Account' order by 1, 2"
SQLEXEC( gnConnHandle, lcSQL, "csrBillPaymentChecks")
BROWSE
USE IN csrBillPaymentChecks
```

When finished, close the connection.

```
SQLDISCONNECT( gnConnHandle)
```

You can find more examples in program QODBC_Fetch.prg in the session downloads.

Writing data to QuickBooks

Writing data to QuickBooks is similar to reading, except of course you use SQL INSERT statements instead of SQL SELECT. However, there are a few intricacies involved in writing data that you'll find yourself repeating time after time—formatting data for SQL INSERT, for example—so it makes sense to encapsulate these things in a class whose methods can be called whenever needed.

The session downloads include a class named clsQuickBooks for this purpose. This class includes methods for connecting and disconnecting from a data source, getting an account name from its account number, formatting dates and strings for use in a SQL statement, creating SQL statements to insert checks and deposit items, and more.

The code that follows is taken from the program QODBC_Insert, which is also included in the session downloads. QODBC_Insert creates an instance of clsQuickBooks and calls its methods in various places. Like the code in QODBC_Fetch, the code in QODBC_Insert is designed for illustration purposes only and is not intended to be run as a complete program.

The first thing QODBC_Fetch does is to create an instance of clsQuickBooks and call its Connect method to connect to the QuickBooks database.

```
PUBLIC goQuickBooks as clsQuickBooks
goQuickBooks = NEWOBJECT( "clsQuickBooks", "clsFoxConQuickBooks.prg")
lcDSN = "QODBC New Venture Software"
lcUsername = "Admin"
lcPassword = "admin"
goQuickBooks.Connect( lcDSN, lcUsername, lcPassword)
```

The goQuickBooks object is public so it stays in scope in the calling code while I run through the various demonstrations during the session. Among other things, this object stores the connection handle in its nConnHandle property, which is then used internally by the object's other methods.

Working with checks

Inserting checks into a QuickBooks database is fairly straightforward, but there is more data involved than you might expect. A check requires not only the name of the payee, a date, and an amount, but also the QuickBooks account representing the bank account on which the check is drawn and the QuickBooks expense account against which the check is to be charged. In addition, a check may also have a check number, a check memo, and an expense memo. Each of these requires an individual variable in your code.

When you insert a check into QuickBooks, you insert it into the CheckExpenseLine table. This table references the bank account and the expense account by name, not by account number. Account numbers are optional in QuickBooks, but if they are used you'll need a way to look up the corresponding QuickBooks full reference name. The clsQuickBooks class provides a method called GetAccountNameFromNumber( ) for this purpose.

QuickBooks rejects an attempt to insert a check into the CheckExpenseLine table if the name of the payee does not already exist in QuickBooks' list of names. If you use QuickBooks, you know that the concept of a name is fundamental to its design. A name can be a vendor, a customer, an employee, or an other name.

In order to avoid errors at runtime, your code can check to be sure the payee names exists before attempting to insert a check payable to that payee. The clsQuickBooks class provides a method called CheckIfNameExists( ) for this purpose. If the name does not exist, you can inform the user of the exception and let them rectify it outside your application, or you can call the InsertName( ) method to add the name programmatically.

In the sample program QODBC_Insert, the code to insert a check is organized into logical pieces. The first part sets up variables for the account names and numbers.

```
LOCAL lcBankAcctName, lcBankAcctNbr, lcExpenseAcctName, lcExpenseAcctNbr
lcBankAcctNbr = "1000"
lcBankAcctName = ""
lcExpenseAcctNbr = "6200"
lcExpenseAcctName = ""
```

As you can see, the sample code uses literals to establish the account numbers. In a real application, the account numbers would probably be passed in as parameters. The assumption is that while the account numbers are known, the account names are not and therefore need to be looked up, as shown in the following code.

```
* Get name of bank account number 1000 (s/b 'Business Checking Account')
lcBankAcctName = goQuickBooks.GetAccountNameFromNumber( lcBankAcctNbr)
* Get name of expense account number 6200 (s/b 'Continuing Education')
lcExpenseAcctName = goQuickBooks.GetAccountNameFromNumber( lcExpenseAcctNbr)
```

The next part sets up the variables for the rest of the data required to insert a check.

```
LOCAL lcCheckNumber, lcPayeeName, lnAmount, lcCheckMemo, ;
      lcExpenseMemo, llIsToBePrinted, llFQSaveToCache, ldDate
lcCheckNumber = "1003"
lcPayeeName = "ABC TRAINING CO."
lnAmount = 476.61
ldDate = {^2008-02-23}
```

```
lcCheckMemo = "VFP FOR DUMMIES"
lcExpenseMemo = "VFP FOR DUMMIES"
llIsToBePrinted = .F.
llFQSaveToCache = .F.
```

Again, for illustration purposes, the values for these variables are set from literals. In a real application they would typically come from some other source.

The last two variables deserve special mention. The llIsToBePrinted variable determines whether QuickBooks marks the check as needing to be printed. If you want to assign a specific check number when you insert the check using QODBC, set this variable to False. If you set it to True, QuickBooks assigns the next sequential check number when the check is printed.

The llFQSaveToCache variable tells QODBC whether or not to cache this transaction. When inserting a single check, set this value to False. The need for this variable becomes apparent when you insert a multi-item deposit, which is discussed below.

With all the variables and their values in place, the code makes a call to the InsertCheckLine( ) method on the goQuickBooks object. This method takes care of some required data formatting tasks and then creates and executes the SQL INSERT statement.

```
goQuickBooks.InsertCheckExpenseLine( lcBankAcctName, ;
                                     lcCheckNumber, ;
                                     lcPayeeName, ;
                                     lnAmount, ;
                                     ldDate, ;
                                     lcCheckMemo, ;
                                     lcExpenseAcctName, ;
                                     lcExpenseMemo,  ;
                                     llIsToBePrinted, ;
                                     llFQSaveToCache)
```

The InsertCheckLine( ) method checks the return value from the SQLEXEC call to the ODBC driver, and returns True if it succeeded and False if it failed. This enables the calling method to take action as appropriate in the event of an error.

Working with deposits

Working with deposits is similar to working with checks. The data requirements are a little different, and there is the additional complication that a deposit might consist of only one deposited item or it might comprise several deposited items. For example, when you deposit your paycheck, that's a single deposit item. On the other hand, when a rental property owner deposits several rent checks from tenants, that's multiple deposit items in a single deposit.

The code for inserting a deposit is much like the code for inserting a check. The entire sample code can be found in the QODBC_Insert program under "Working with deposits", so I won't reproduce it here. One thing that does need discussion, however, is the setting of the cache flag.

When you insert a deposit, you insert a row into the DepositLine table. The DepositLine table stores information about each deposited item. Internally, QuickBooks also inserts a new row into the Deposit table, which is kind of a header record for deposits. Although I didn't mention it above, the same sort of thing happens when you insert a check into the CheckExpenseLine table: QuickBooks automatically inserts a row into the Check table as well.

When you're working with a single deposit item, one row in the DepositLine table corresponds to one row in the Deposit table. Both are created at the same time, so there's no need for QODBC to cache anything.

However, when working with multiple deposit items, several rows in the DepositLine table correspond to only one row in the Deposit table. The catch is that items are inserted one into the DepositLine table one at a time, but the row for the Deposit table cannot be built until the last DepositLine row has been inserted. To handle this, QODBC provides the cache flag. If this flag is set True, QODBC caches the item being inserted. When the flag is set to False, QODBC processes the item and any previously cached items as a single transaction.

In the sample code, the cache flag is passed to QODBC via the llFQSaveToCache variable. To insert a multiple item deposit, set llFQSaveToCache to True for all but the last item, and set it to False for the last item. When the last item is inserted, QODBC submits the entire transaction to the database, thus establishing the one to many relationship between the many explicitly inserted rows in the DepositLine table and the one internally generated row in the Deposit table.

QODBC Stored Procedures

QODBC also comes with several stored procedures you can use. Although Visual FoxPro provides the SQLTABLES( ) and SQLCOLUMNS( ) functions, you can get the same information by calling the SP_TABLES and SP_COLUMNS provided by QODBC.

```
* List the tables in the database
SQLEXEC( lnConnHandle, "sp_tables", "csrTables")


* List the columns in the Account table
SQLEXEC( lnConnHandle, "sp_columns Account", "csrColumns")
```

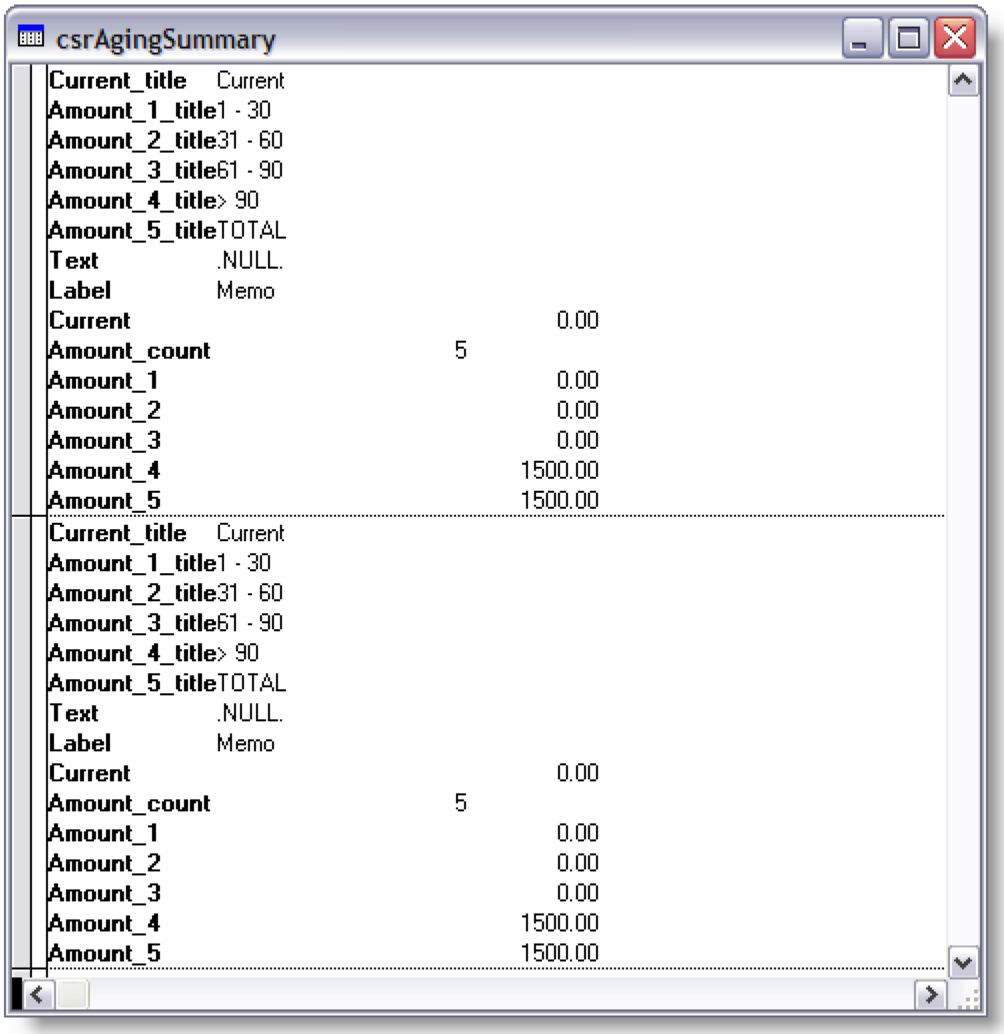
The SP_BATCHSTART, SP_BATCHUPDATE, and SP_BATCHCLEAR stored procedures can be used to do batch mode inserts and updates. These and the rest of the stored procedures are documented on the QODBC website at http://www.qodbc.com/QODBCStoredProcedures.htm.

QODBC Reports

The special stored procedure SP_REPORT enables you to pull data for familiar QuickBooks reports in one step. For example, you can pull the data for the Account Receivable Aging Summary report like this:

```
SQLEXEC( lnConnHandle, "sp_report ARAgingSummary show Current_Title, Amount_Title, Text, Label,
Current, Amount parameters DateMacro = 'Today', AgingAsOf = 'Today'", "csrAgingSummary")
```

Figure 11 shows the resulting cursor, displayed in Edit mode so you can see all the columns. The memo field named Label is the customer name. In a cursor with ‘n’ rows, the first n-1 rows are the detail and the last row contains the totals. This sample cursor has only one detail row, so the amounts in the totals row are the same as the detail.



```
Current_title   Current
Amount_1_title1 - 30
Amount_2_title31 - 60
Amount_3_title61 - 90
Amount_4_title> 90
Amount_5_titleTOTAL
Text            .NULL.
Label           Memo
Current                              0.00
Amount_count                    5
Amount_1                             0.00
Amount_2                             0.00
Amount_3                             0.00
Amount_4                          1500.00
Amount_5                          1500.00
Current_title   Current
Amount_1_title1 - 30
Amount_2_title31 - 60
Amount_3_title61 - 90
Amount_4_title> 90
Amount_5_titleTOTAL
Text            .NULL.
Label           Memo
Current                              0.00
Amount_count                    5
Amount_1                             0.00
Amount_2                             0.00
Amount_3                             0.00
Amount_4                          1500.00
Amount_5                          1500.00
```

Figure 11: The SP_REPORT stored procedure can be used to pull data for familiar QuickBooks reports, such as this one for the Accounts Receivable Aging Summary report for customers.

The syntax for this SP_REPORT store procedure call comes from an example on the QODBC website. The complete list of reports available via the SP_REPORT stored procedure is documented on the website at http://doc.qodbc.com/qodbc/20/reports/sp_report_detail.html.

Troubleshooting and Debugging

In my experience there are two primary sources of frustration when working with QuickBooks programmatically. Fortunately, QODBC makes is relatively easy to diagnose the problem by providing logs and even optional detail tracing of what it's doing when accessing QuickBooks.

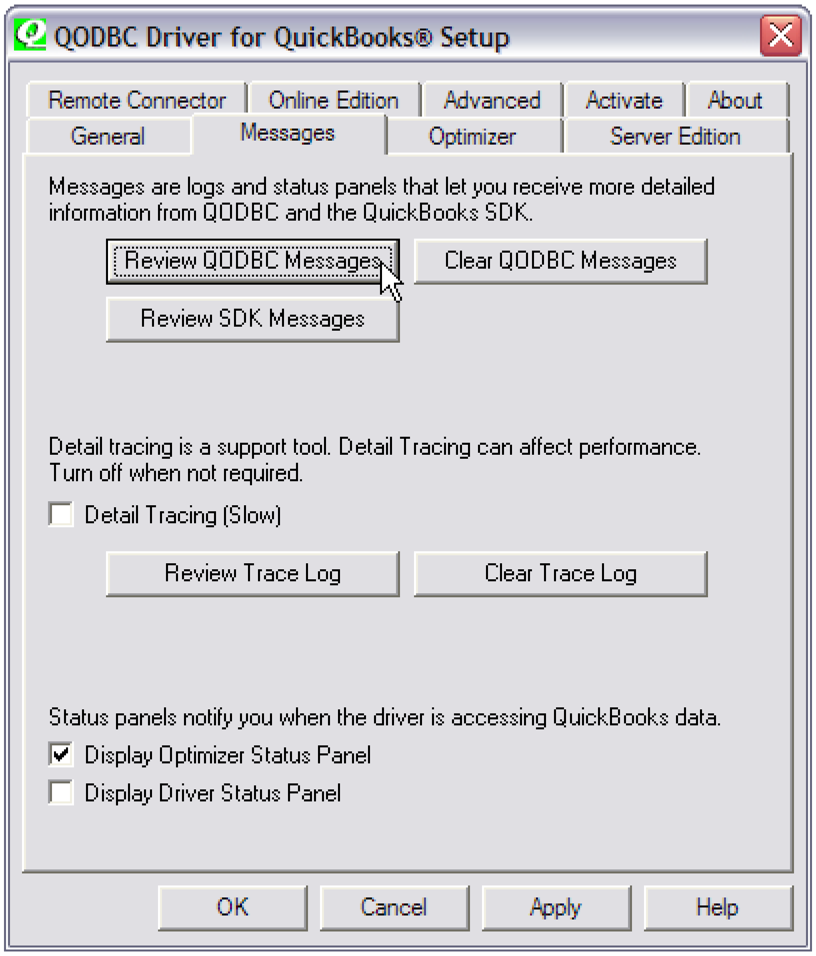
Connection Problems

Connection problems probably top the list of things that can go wrong and prevent your application from successfully accessing a QuickBooks database. Some of the common reasons for connection problems include:

- QuickBooks not installed on the user's machine
- QuickBooks not configured to allow access to a company file
- DSN not defined or not properly configured
- A different company file is already open in QuickBooks on the user's machine

The last one can be particularly frustrating because it's one you generally have no control over at run time. QuickBooks only allows one company file to be open at a time. This is simply the way QuickBooks is designed; it is not a restriction imposed by QODBC. As you've seen, you can configure QuickBooks access rights to allow QODBC to access a company file even if it is not already open in QuickBooks, but QODBC then has to tell QuickBooks to open the company file and that fails if another company file is already open. The result is that the connection attempt fails as well.

QODBC writes a log file of everything it does. You can use this log file to help determine why an error occurred. To access the log file, open the QODBC Setup dialog (the same one you used to configure the DSN) and select the Messages page.



Messages are logs and status panels that let you receive more detailed information from QODBC and the QuickBooks SDK.

Detail tracing is a support tool. Detail Tracing can affect performance. Turn off when not required.

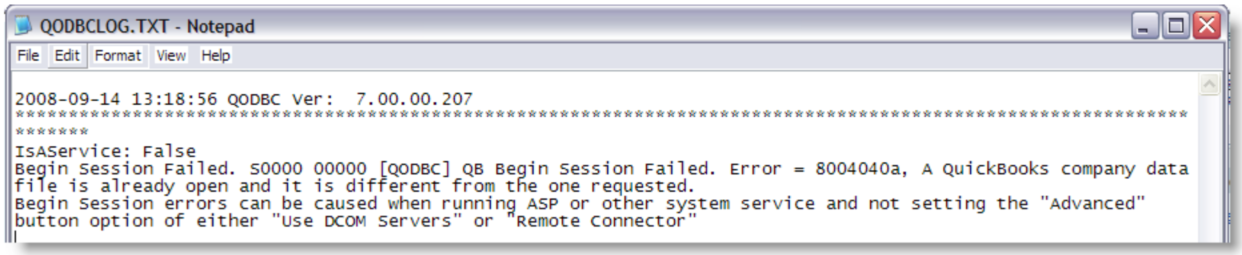
- [ ] Detail Tracing (Slow)

Status panels notify you when the driver is accessing QuickBooks data.

- [✓] Display Optimizer Status Panel
- [ ] Display Driver Status Panel

Figure 12: Use the Messages page of the QODBC Setup dialog to access the log files and for other help in troubleshooting and debugging.

Click on the Review ODBC Messages button to open the log in Notepad or your default text editor.



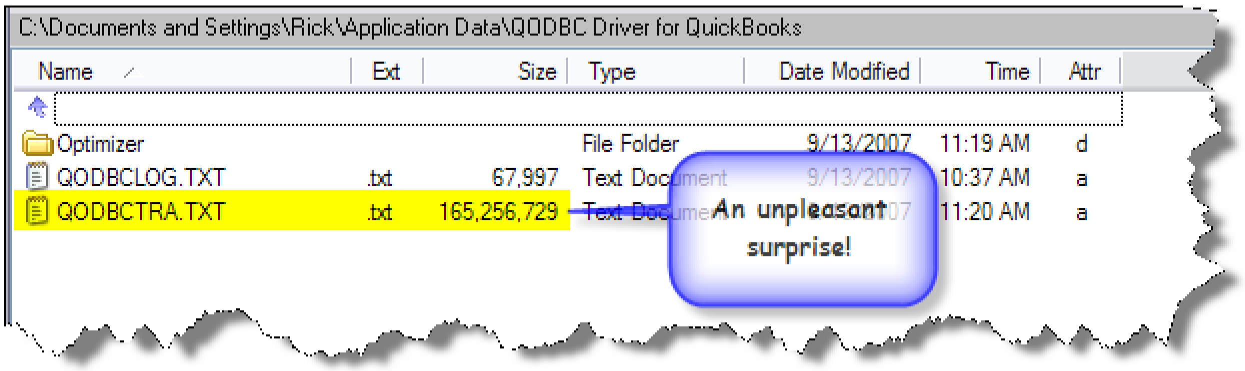
```
2008-09-14 13:18:56 QODBC Ver:  7.00.00.207
*****************************************************************************************************
*******
IsAService: False
Begin Session Failed. S0000 00000 [QODBC] QB Begin Session Failed. Error = 8004040a, A QuickBooks company data
file is already open and it is different from the one requested.
Begin Session errors can be caused when running ASP or other system service and not setting the "Advanced"
button option of either "Use DCOM Servers" or "Remote Connector"
```

Figure 13: When an error occurs, the QODBC log file can help you figure out what's wrong.

In Figure 13, the log file tells you in plain English what the problem is: "A QuickBooks company file is already open and it is different from the one requested." If this happens during testing and debugging, you can usually take care of the problem yourself. It may be a challenge to figure out how best to handle this if it happens at runtime on the client's site, but at least you know what the problem is.

The Messages page of the Setup dialog also provides access to other tools you can use to help in troubleshooting and debugging. The Review SDK Message button gives you access to a different log file where the details of the qbXML interaction between QODBC and QuickBooks are recorded. And if you really get stuck on a difficult problem, you can turn on Detail Tracing for an even deeper look at what's going on.

Detail Tracing in QODBC is similar to Coverage Logging in Visual FoxPro, and in both cases the trace log can get really big really quickly. If you do turn on Detail Tracing, don't forget to turn it off again. For one thing, detail tracing can have a noticeable impact on performance. For another thing, leaving detail tracing on allows the trace log to grow continuously, and you may be in for an unpleasant surprise when you investigate to find out where all your free disk space went.



C:\Documents and Settings\Rick\Application Data\QODBC Driver for QuickBooks

Name	Ext	Size	Type	Date Modified	Time	Attr
Optimizer			File Folder	9/13/2007	11:19 AM	d
QODBCLOG.TXT	.txt	67,997	Text Document	9/13/2007	10:37 AM	a
QODBCTRA.TXT	.txt	165,256,729	Text Document	[illegible]	11:20 AM	a

Figure 14: The QODBC trace log can get very large very quickly. Don't forget to turn Detail Tracing off when you're done with it.

SQL Update and Insert problems

Issues with SQL Update and Insert statements comprise the other category of common problems when using QODBC. Aside from simple syntax errors, common causes of SQL statement problems include such things as the order in which columns are named, and the format of date fields.

If you encounter errors with SQL Insert or Update commands that cannot be traced to other causes, QODBC recommends that you ensure the columns are placed in the SQL statement in the same order in which they are defined in the database schema. A while back, QODBC indicated they were working to resolve this issue in a newer release and they may in fact have done so in the latest release. However, this advice still stands as kind of a last resort, something you can try if everything else seems to be in order but the SQL statement still fails.

The recommended format for dates in SQL statements sent to QuickBooks via QODBC is this:

```
{d 'yyyy-mm-dd'}
```

The first 'd' is a literal and yyyy-mm-dd is the date, enclosed in single quote marks, with the entire expression enclosed in curly braces. I don't know what other date formats might also work, but I do know that by consistently using this format I have never had a problem with a date field.

QODBC Support

As noted earlier, QODBC has an extensive website with a rich variety of documents to help you learn and use the product. A site-wide search is provided at http://www.qodbc.com/search.htm. The support and FAQs page is found at http://www.qodbc.com/QODBCAutoFAQ.htm. In addition, QODBC provides a free technical support forum, which is typically quite active and which I have found helpful on more than one occasion. The URL for the support forum home page is www.qdeveloper.com.au/forum.php. Be patient: the site sometimes takes a several seconds to respond. There are several categories of discussion on the forum, organized according to the different versions of QODBC.

Conclusion

If you need to access QuickBooks data programmatically, QODBC provides a great solution. It installs as an ODBC driver and enables you to use standard SQL statements to read and write QuickBooks data. QODBC is a commercial product you have to pay for, but in my opinion the value it delivers fully justifies the cost. The QODBC website provides extensive documentation both about their own product and about the QuickBooks database schema. The intent of this session has been to provide you with enough information to get you started using QODBC with Visual FoxPro.

About the author

Rick Borup is owner and president of Information Technology Associates, a professional software development, computer services, and information systems consulting firm he founded in 1993. Rick spent several years developing software applications for mainframe computers before turning to microcomputer database development tools in the late 1980s. He began working with FoxPro in 1991, and has worked full time in FoxPro and Visual FoxPro since 1993. He is co-author of the books Deploying Visual FoxPro Solutions and Visual FoxPro Best Practices For The Next Ten

Years, and a frequent speaker at Visual FoxPro conferences and user groups. Rick is a Microsoft Certified Solution Developer (MCSD) and a Microsoft Certified Professional (MCP) in Visual FoxPro.