



Developer's Guide

Table of Contents

I. INTRODUCTION.....	3
II. INTEGRATION STEPS.....	4
1) Install & Register the Plug-in	4
2) Install Components and Establish Initial QuickBooks Connection.....	5
3) Talking to QuickBooks	6
The Request and the Response Explained:	7
How to Create a Request:	8
How to Post the Request:	10
How to Parse the Response:.....	11
Custom Fields:	13
4) How To Use the OSR.....	15
Step 1 - Set Up	15
Step 2 - Using the OSR.....	16
Step 3 - Proper Field Order	18
5) Tips.....	19
Tax Tips:.....	19
Canadian Versions:.....	19
UK and Australian Versions:	19
6) Tutorial Videos and Sample Code.....	20
III. CONTACT US.....	20

I. Introduction

Description:

The FM Books Connector plug-in is a powerful tool used to move data between FileMaker® Pro and Intuit QuickBooks® applications. The Developer's Guide will explain the necessary integration steps, how the plug-in 'talks' to QuickBooks and the concept of how to create your FileMaker scripts which can be applied to any area in QuickBooks. Understanding the concept of the script construction and how the data exchange functions will save a tremendous amount of time and confusion during the plug-in integration process.

Intended Audience:

FileMaker developers or persons, who have knowledge of FileMaker scripting, calculations and relationships as proper use of the plug-in requires that FileMaker integration scripts be created in your FileMaker solution.

Successful Integration Practices:

- 1) Read the Developer's Guide
- 2) Familiarize yourself with basic accounting practices and QuickBooks
- 3) Read the Functions Guide
- 4) Use the OSR (Intuit's Onscreen Reference Manual)

OSR can be found here: <http://developer.intuit.com/qbsdk-current/Common/newOSR/index.html>

- 5) Look at our FileMaker Demo and video tutorials

Demo and video tutorials can be found here: <http://www.fmbooksconnector.com>

- 6) Re-read the Developer's Guide

Error Handling:

Any of the plug-in functions may encounter an error during processing. The function will return the string "**!! ERROR!!**". When an error occurs during processing, immediately call the PCQB_SGetStatus function in order to obtain a full description of the error. This function returns the message associated with the last error. The status is used to identify errors in the request or the processing of requests. The text returned by this function will help troubleshoot script or logic failures.

II. Integration Steps

1) Install & Register the Plug-in

Installing the Plug-in:

The first step is to install the plug-in into FileMaker.

- 1) Quit FileMaker Pro completely.
- 2) Locate the plug-in in your download which will be located in a folder called "Plug-in". On Windows the plug-in will have a ".fmx" extension.
- 3) Copy the actual plug-in and paste it to the Extensions folder which is inside the FileMaker program folder. On Windows this is normally located here: C:\Program Files\FileMaker\FileMaker X\Extensions.
- 4) Start FileMaker. Confirm that the plug-in has been successfully installed by navigating to "Preferences" in FileMaker Pro, then click the "Plug-ins" Tab. There you should see the plug-in listed with a corresponding check box. This indicates that you have successfully installed the plug-in

Registration:

The next step is to register the plug-in which enables all plug-in functions.

- 5) Confirm that you have access to the internet and open our FileMaker demo file, which can be found the in "FileMaker Demo File" folder in your original download.
- 6) If you are registering the plug-in in Demo mode, then simply click the "Register the Plug-in" button and do not change any of the fields. Your plug-in should now be running in "DEMO" mode. The mode is noted in the upper right hand corner of our FileMaker Demo file on the Setup tab.
- 7) If you are registering a licensed copy, then simply enter your license number in the "LicenseID" field and click the "Register the Plug-in" button. Make sure you remove the Demo License ID and enter your registration information exactly as it appears in your confirmation email. Your plug-in should now be running in "LIVE" mode. The mode is noted in the upper right hand corner of our FileMaker Demo file on the Setup tab.

Congratulations! You have now successfully installed and registered the plug-in!

Why do I need to Register?

In an effort to reduce software piracy, Productive Computing, Inc. has implemented a registration process for all plug-ins. The registration process sends information over the internet to a server managed by Productive Computing, Inc. The server uses this information to confirm that there is a valid license available and identifies the machine. If there is a license available, then the plug-receives an acknowledgment from the server and installs a certificate on the machine. This certificate never expires. If the certificate is ever moved, modified or deleted, then the client will be required to register again. On Windows this certificate is in the form of a "pci" file.

The registration process also offers developers the ability to automatically register each client machine behind the scenes by hard coding the license ID in the PCQB_Register function. This proves beneficial by eliminating the need to manually enter the registration number on each client machine. There are other various functions available such as PCQB_GetOperatingMode and PCQB_Version which can assist you when developing an installation and registration process in your FileMaker solution.

2) Install Components and Establish Initial QuickBooks Connection

After you have installed and registered the plug-in we then need to install two components and establish a connection with a QuickBooks File.

Installing the QBXMLRP2 file:

This installer is found in the QBXMLRP2 folder. Inside this folder you will find a file called: "QBXMLRP2Installer.exe". Please double click this file and run the installer. This file is required for users of all versions of QuickBooks in order to ensure that all necessary components have been installed.

Installing the Microsoft XML Processor:

Included in the package is a download link for all users of Windows.

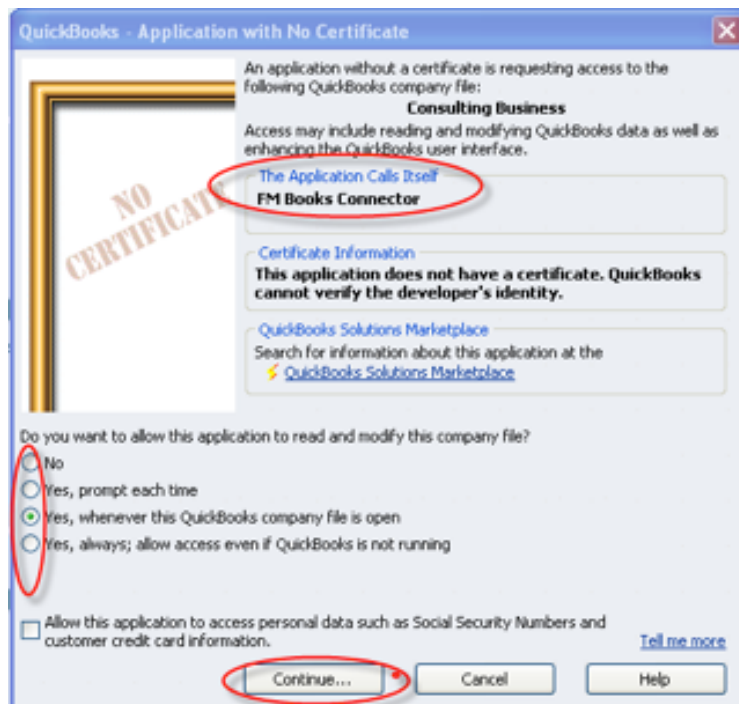
Name of link is: "Install MSXML processor update from Microsoft (Required Install)"

This link will direct you to install version 6 of the MSXML Processor from Microsoft. We use version 6 of the MSXML processor in the plug-in and require that you install this processor. Please save the file to your desktop and then run the file locally from your machine to ensure proper installation.

Establish Initial Connection with QuickBooks Company File:

First log into that QuickBooks file with "Admin" access. When making a call to QuickBooks via the FM Books Connector plug-in, you will see a screen similar to the one below. Once this screen appears, select the appropriate radio button to continue and allow communication with QuickBooks. We recommend selecting the 3rd or 4th radio button as shown below. These settings can be changed later in QuickBooks under the Edit, Preferences, Integrated Applications Area in Quick Books.

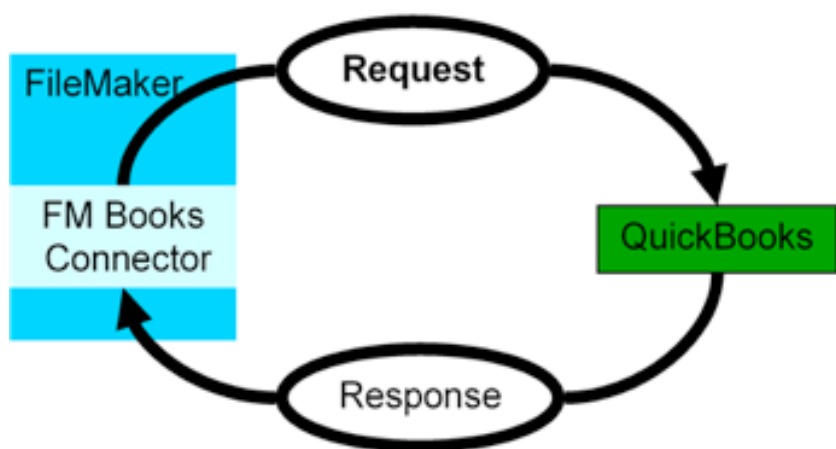
Figure 1.0 - Sample Screen Shot of QuickBooks Certificate



3) Talking to QuickBooks

It is easiest to picture the information exchanged between the plug-in and QuickBooks as a very short conversation. Actually, this conversation is made simply of a request and a response to that request. That's it - a very short and simple conversation. See Figure 2.0 below.

Figure 2.0 – File Maker and QuickBooks Information Exchange



No matter what the request is, whether it is to add contacts to a QuickBooks company file or to find all unpaid invoices, the conversation always takes the form of a request to do something and a response suitable for the type of request made.

The following rules also apply:

a) The plug-in always initiates any conversation with a request.

b) QuickBooks always responds to a properly posed request.

c) The conversation is always finished after QuickBooks responds to the request.

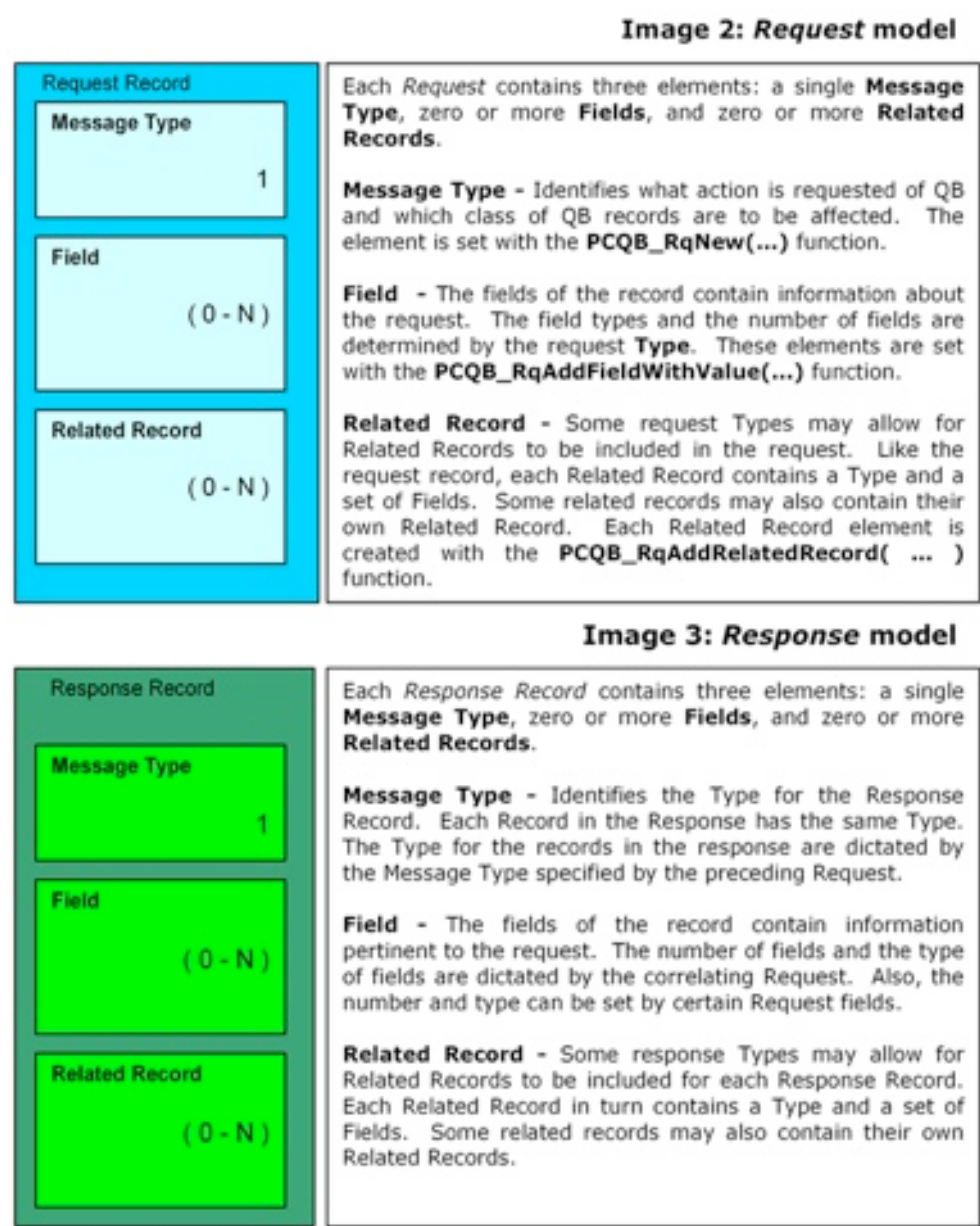
The request and response conversation is the foundation for exchanging information between FileMaker and QuickBooks. It is imperative to remember that this is the form of the conversation no matter what the plug-in is requesting of QuickBooks. It will be especially useful to remember this when it comes time to create scripts in your FileMaker solution for exchanging data with QuickBooks

d) We now know how the plug-in talks to QuickBooks. We know that the conversation is short and to the point. What we have yet to learn is what is 'said' during this conversation and how it is 'spoken.' We could go in to the gory details of the language that is used in this conversation, but we created this plug-in specifically so that the user does not need to know these details. With the plug-in all you need know is how to create a request, how to post the request to QuickBooks, and how to read QuickBooks's response.

The Request and the Response Explained:

Requests are created and responses are read using external functions. Different requests and responses have different fields defined for them. Furthermore, requests and responses may have related items. Please refer to Figure 2.1 for an explanation of the three different elements for both a request and response.

Figure 2.1



*Please come back and reference Figure 2.1 after you have read this document. *

How to Create a Request:

Each request has a certain predefined Message Type. A Message Type for a request has the distinct honor of telling QuickBooks what the request is aiming to do, and which records are to be affected. In terms that are more technical, the Message Type defines which class of records we want to work with and which action we want applied to those records. For instance, the 'CustomerAdd' Message Type tells QuickBooks that the request wants to create a new Customer record in the QuickBooks file. The available Message Types for a request are listed in the OSR and are left out of this document for the sake of brevity.

Some sample request Message Types and their definitions follow:

CustomerMod the request wants to modify an existing Customer record in QuickBooks.

InvoiceQuery the request wants to find Invoice records in QuickBooks

SalesOrderAdd the request wants to add a Sales Order record to QuickBooks

A Request also contains fields that further define what the request is to do and which records are to be affected. The field names available for the different request Message Types are also listed in the OSR.

Two functions are used to create a basic request. They are:

PCQB_RqNew(MessageType)

PCQB_AddFieldWithValue(Fieldname ; Value)

The first function is used to create a request of the desired Message Type, and the second is used to populate the fields of the request. A simple request to add a Customer follows:

PCQB_RqNew("CustomerAdd")

PCQB_RqAddFieldWithValue("Name" ; "Bob Jones")

PCQB_RqAddFieldWithValue("BillAddress::Addr1" ; "123 Any Street")

PCQB_RqAddFieldWithValue("BillAddress::City" ; "Any Town")

PCQB_RqAddFieldWithValue("BillAddress::State" ; "Any State")

PCQB_RqAddFieldWithValue("BillAddress::PostalCode " ; "11111")

PCQB_RqAddFieldWithValue("Email" ; "bill@someisp.com")

For some request Message Types, the developer will want to include related records in the request. For example, when adding an invoice the developer would like to add the line items also. To accomplish this, two other functions are also used. They are:

```
PCQB_RqAddRelatedRecord( ElementName )  
PCQB_RqCloseRelatedRecord
```

A sample InvoiceAdd request demonstrates the usage of these two functions.

```
PCQB_RqNew( "InvoiceAdd" )  
PCQB_RqAddFieldWithValue( "CustomerRef::FullName" ; "Bob Jones" )  
PCQB_RqAddFieldWithValue( "TxnDate" ; "2006/01/01" )  
PCQB_RqAddFieldWithValue( "RefNumber" ; "123456789" )  
PCQB_RqAddFieldWithValue( "BillAddress::Addr1" ; "123 Any Street" )  
PCQB_RqAddFieldWithValue( "BillAddress::City" ; "Any Town" )  
PCQB_RqAddFieldWithValue( "BillAddress::State" ; "Any State" )  
PCQB_RqAddFieldWithValue( "BillAddress::PostalCode " ; "11111" )  
PCQB_RqAddRelatedRecord( "InvoiceLineAdd" )  
PCQB_RqAddFieldWithValue( "ItemRef::FullName" ; "Widgit" )  
PCQB_RqAddFieldWithValue( "Quantity" ; "20" )  
PCQB_RqAddFieldWithValue( "Amount" ; "79.95" )  
PCQB_RqCloseRelatedRecord  
PCQB_RqAddRelatedRecord( "InvoiceLineAdd" )  
PCQB_RqAddFieldWithValue( "ItemRef::FullName" ; "Gadget" )  
PCQB_RqAddFieldWithValue( "Quantity" ; "3" )  
PCQB_RqAddFieldWithValue( "Amount" ; "2.95" )  
PCQB_RqCloseRelatedRecord
```

You will notice that the PCQB_RqAddFieldWithValue function operates in the context of the current record. When the PCQB_RqAddRelatedRecord function is called the context shifts from the parent record (the main request) to the related record, in this case the InvoiceLineAdd record. Subsequent calls to PCQB_RqAddFieldWithValue will add field values to the related record until the PCQB_RqCloseRelatedRecord function is called and the context is shifted back to the parent record. Another important note to make is that the fields of a request must be set in a specific order. The order to set the fields is listed in the OSR. It is also important to note that the PCQB_RqAddRelatedRecord must also be called in a specific order. Again, this order is listed in the OSR. Later in this document we will discuss how to use the OSR.

How to Post the Request:

Now that we know how to create a request, we need to learn how to post the request to QuickBooks. This is a rather simple operation that requires only a single execute function. However this is the perfect time to explain two other functions that the execute function relies upon. The three functions are:

PCQB_BeginSession(CompanyFile ; ShareMode)

PCQB_Execute

PCQB_EndSession

PCQB_BeginSession is used to establish a session with QuickBooks. This function must be returned successfully before we can post a request to QuickBooks. It may be called anytime during the script, but since it may block other applications from accessing the QuickBooks file it is proper etiquette to call it immediately before executing a request and then calling PCQB_EndSession immediately after executing. This minimizes the possibility of blocking other applications from accessing the file while you script builds requests or processes responses.

The following table lists the effects of the mode in different operating environments:

Who started QuickBooks	Selected Mode	Who else may obtain access
FM Books Connector	"Single"	No one else
FM Books Connector	"Multi"	QuickBooks user on same machine = no access All other integrated applications = access QuickBooks users on other machines = access
QuickBooks User	"Single"	QuickBooks user already logged in Only one integrated application = access
QuickBooks User	"Multi"	QuickBooks users = access Integrated applications = access

How to Parse the Response:

Once a request is built and successfully executed, the plug-in will retain the response in memory. Depending on the type of request that is made the response may contain one or several records. For instance, after executing a request to add a customer (using a request of Message Type 'CustomerAdd') QuickBooks will respond with a 'CustomerRet' record that the plug-in will hold in memory. Query requests such as an 'InvoiceQuery' request may return several records in the response. Each of the records is accessed and parsed for the information they contain.

Accessing and parsing the records contained in the response is a rather simple process.

Six functions are used to read the contents of a response:

```
PCQB_RsOpenFirstRecord  
PCQB_RsOpenNextRecord  
PCQB_RsOpenFirstRelatedRecord( ElementName )  
PCQB_RsOpenNextRelatedRecord  
PCQB_RsCloseRelatedRecord  
PCQB_RsGetFieldValue( Name )
```

The first two functions are used to iterate through all the records in the response. PCQB_RsOpenFirstRecord opens the first record in the response and PCQB_RsOpenNextRecord is used to open successive records in the response. PCQB_RsOpenNextRecord will return "End" when there are no more records to be read in the response.

Once a record is opened with either of the open record functions the user is able to read the record contents using the PCQB_RsGetFieldValue(Name) function. The name of the desired field is passed with the function and the contents of the field are returned. See the "OSR" for acceptable field names.

As with a request there may be related records to each record in the response. These related records are accessed with the PCQB_RsOpenFirstRelatedRecord(ElementName) and PCQB_RsOpenNextRelatedRecord functions. Once the related record is opened, its contents can be read with the PCQB_RsGetFieldValue function.

An example of reading a response to an 'InvoiceQuery' request follows:

```
#Open the First record in the response
SetField[ someField ; PCQB_RsOpenFirstRecord ]
Loop
  #Exit the loop on error or after last record is read
  Exit Loop If [ (someField < 0) or (someField = "End") ]
  # Get the name of the Customer and the Accounts Receivable
  SetField [someField ; PCQB_RsGetFieldValue( "CustomerRef::FullName" ) ]
  SetField [someField ; PCQB_RsGetFieldValue( "ARAccountRef::FullName" ) ]
  # Get any and all related transactions to the current invoice
  SetField [someField ; PCQB_RsOpenFirstRelatedRecord( "LinkedTxn" ) ]
  Loop
    Exit Loop If [ (someField < 0) or (someField = "End") ]
    #Gets the information from the related transaction
    SetField [someField ; PCQB_RsGetFieldValue( "RefNumber" ) ]
    SetField [someField ; PCQB_RsGetFieldValue( "Amount" ) ]
    #Opens the next related transaction
    SetField [someField ; PCQB_RsOpenNextRelatedRecord ]
  End Loop
  #Close the related record to return to the main record
  SetField [someField ; PCQB_RsCloseRelatedRecord ]
  #Opens the invoice line item related records
  SetField [someField ; PCQB_RsOpenFirstRelatedRecord( "InvoiceLineRet" ) ]
  Loop
    Exit Loop If [ (someField < 0) or (someField = "End") ]
    #Gets the information from the line item
    SetField [someField ; PCQB_RsGetFieldValue( "ItemRef::FullName" ) ]
    SetField [someField ; PCQB_RsGetFieldValue( "Amount" ) ]
    #Opens the next line item
    SetField [someField ; PCQB_RsOpenNextRelatedRecord ]
  End Loop
  #Close the related record to return to the main record
  SetField [someField ; PCQB_RsCloseRelatedRecord ]
  #Open the next record in the response
  SetField [someField ; PCQB_RsOpenNextRecord ]
End Loop
```

Custom Fields:

Custom field are accessed using DataExt objects. Each custom field is a DataExt object as the QBSDK uses a DataExt object for custom fields. If a QB object (invoice, customer, etc) supports custom fields then the list of available fields in the Response for the object will contain a DataExtRet object. Accessing the DataExtRet object in a response is the same as accessing any other related record of the response. Setting values for DataExt objects is accomplished with DataExtMod requests. One must be familiar with creating requests using the OSR to successfully modify DataExt objects.

This DataExt object is available in the following QBs lists:

- Accounts
- Customers
- Vendors
- Items
- OtherNames
- Employees

The DataExt object is also available in the following transaction types:

- ARRefundCreditCard
- Bill
- BillPaymentCheck
- BillPaymentCreditCard
- BuildAssembly
- Charge
- Check
- CreditCardCharge
- CreditCardCredit
- CreditMemo
- Deposit
- Estimate
- InventoryAdjustment
- Invoice, ItemReceipt
- JournalEntry
- PurchaseOrder
- ReceivePayment
- SalesOrder
- SalesReceipt
- SalesTaxPaymentCheck
- VendorCredit

Adding values to a custom field is not very well defined in the QBSDK, therefore it is difficult to explain how to use them with our plug-in. In some cases the custom field is populated with a separate request and in other cases it is populated with the parent item request. For instance when adding/modifying Customer in QuickBooks with the plug-in the DataExt aggregate is not available in the CustomerAdd nor CustomerMod request. But in the InvoiceAdd request a DataExt aggregate is available for each line item in the invoice. This inconsistency makes accessing custom fields difficult at best.

The most consistent way to add/mod/del the contents of a custom field is to use the DataExtAdd, DataExtMod, and DataExtDel requests. These requests can be found in the "OSR".

Retrieving the contents of a custom field requires querying for the parent object and including the OwnerID field in the query, which is normally one of the last fields to be added to the request. Obtaining the DataExt values in the Response to a query objects (contacts, invoices,etc...) requires that the request contain the following PCQB_RqAddFieldWithValue("OwnerID" ; "0").

The above function adds the OwnerID field to the request, and populates it with a '0'. This causes QuickBooks to return the public data extensions (custom fields) with the response. (Advanced users can cause QuickBooks to return private data extensions by passing the GUID instead of '0', but this is only for advanced users). When QuickBooks returns the DataExt (custom fields) in the response, the plug-in user can access the information in the data extension. The following script demonstrates accessing the custom fields in a response:

```
If [ 0 = PCQB_RsOpenFirstRelatedRecord( "DataExtRet" ) ]  
  Loop  
    #the name of the custom field  
    Set Field[ N_Field ; PCQB_RsGetFirstFieldValue( "DataExtName" ) ]  
    #the value of the custom field  
    Set Field[ D_Field ; PCQB_RsGetFirstFieldValue( "DataExtValue" ) ]  
    #get next custom field/exit if there are no more  
    Exit Loop If[ 0 <> PCQB_RsOpenNextRelatedRecord ]  
  End Loop  
Set Field[ SomeField ; PCQB_RsCloseRelatedRecord ] End If ...
```

Since custom fields are more advanced and can be quite complex, we are available for hire to assist with this development.

4) How To Use the OSR

Now that you thoroughly understand how FileMaker and QuickBooks “talk” to each other by making requests and responses, we are ready to introduce Intuit’s OSR (Onscreen Reference Manual). The OSR contains a complete list of all available fields/filters, detailed descriptions of the request types, errors and specifies field order. This will be crucial during your development.

The OSR can be found at the following link:

<http://developer.intuit.com/qbsdk-current/Common/newOSR/index.html>

Step 1 - Set Up

First use the control panel on the left to select the proper settings.

- The **SDK Version** should be set to 7.0 when using QBs 2008 or QBs v8. If using QBs 2007 or v7 then the SDK should be set to 6.0. If using QBs 2006 or v6 then the SDK should be set to 5.0. The formula is typically the QBs version minus 1.
- The **Format** will need to be changed to qbXML. The format should ALWAYS be set to qbXML.
- The **QB Editions** specify what international version of QBs you are using. If you are using the US edition then select “US”. If you are using the Canadian or UK edition, then please select “Allow CA and UK” and select the appropriate edition. For the Australian version select UK.

For example, in figure 5.0 below the SDK Version is 7.0, the format is qbXML and the US edition of QuickBooks has been selected.

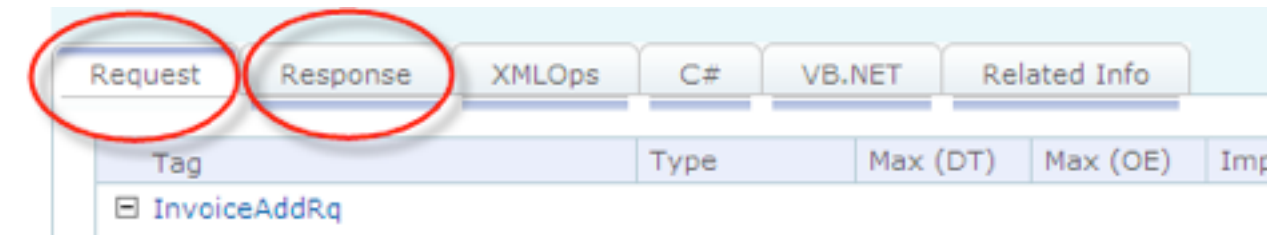
Figure 5.0



Step 2 - Using the OSR

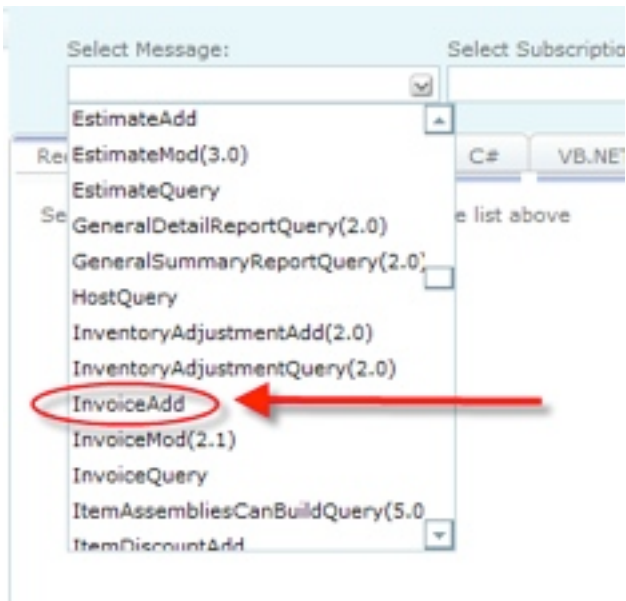
You will only be working with the Request and Response Tabs as shown below in Figure 5.1. All other tabs can be ignored.

Figure 5.1



Since you already understand what Requests and Responses are, select the appropriate tab to begin. After specifying the Response or Request tab, then you will select a Message. The Message defines which class of records we want to work with and which action we want applied to those records. In our example in Figure 5.2 we will select the Message called "InvoiceAdd" in order to add an Invoice to QuickBooks. The "InvoiceAdd" Message tells QuickBooks that the request wants to create a new Invoice record in the QuickBooks file.

Figure 5.2



You will now be looking at a screen with columns titled Tag, Type, Max (DT), Max (OE), Implementation and Occurrences. Let's have a closer look at what these columns mean in Figure 5.3.

Figure 5.3

Tag	Type	Max (DT)	Max (OE)	Implementation	Occurrences
InvoiceAddRq					
InvoiceAdd (defMacro)					1
CustomerRef					1
ListID	IDTYPE				0 - 1
FullName	STRTYPE	209 chars	1000 chars		0 - 1
ClassRef					0 - 1
ListID	IDTYPE				0 - 1
FullName	STRTYPE	159 chars	1000 chars		0 - 1
ARAccountRef					0 - 1
ListID	IDTYPE				0 - 1
FullName	STRTYPE	159 chars	1000 chars		0 - 1
TemplateRef					0 - 1
ListID	IDTYPE			3.0	0 - 1

1. Tag = name of the element or field and the value that you put into the function
For example, `PCQB_RqAddFieldWithValue("CustomerRef::FullName" ; "Bob Jones")`
Please left click on the element name to obtain a detailed description as shown in Figure 5.4 below.
Some element names will also allow you to right click and obtain a corresponding QBs field mapping image.

Figure 5.4

Tag	Type	Max (DT)	Max (OE)	Implementation	Occurrences
InvoiceAddRq					
InvoiceAdd (defMacro)					1
CustomerRef					1
ListID	IDTYPE				0 - 1

ListID
Along with FullName, ListID is a way to identify a list object. When a list object is added to QuickBooks through the SDK or through the QuickBooks user interface, the server assigns it a ListID. A ListID is not unique across lists, but it is unique across each particular type of list. For example, two customers could not have the same ListID, and a customer could not have the same ListID as an employee (because Customer and Employee are both name lists). But a customer could have the same ListID as a non-inventory item.

Tag	Type	Max (DT)	Max (OE)	Implementation	Occurrences
FullName	STRTYPE	159 chars	1000 chars		0 - 1
TemplateRef				3.0	0 - 1

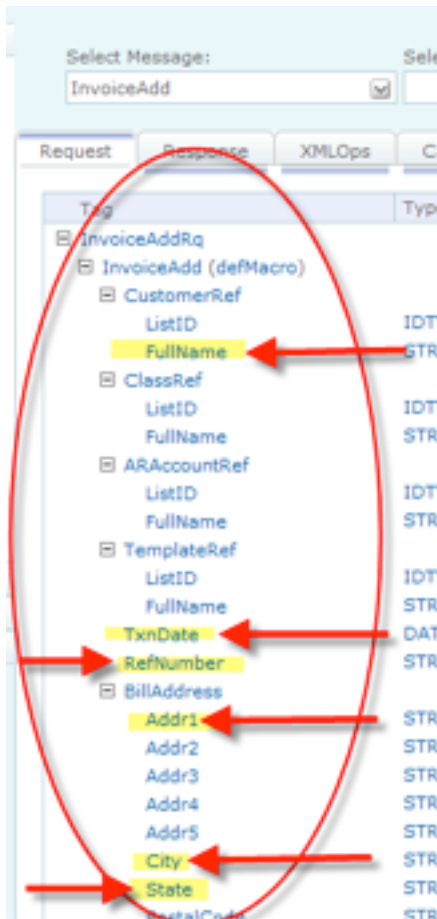
- 2. Type = type of data that goes into this field
- 3. Max (DT) = number of allowable characters in the field for desktop versions
- 4. Max (OE) = number of allowable characters in the field for online versions and currently not applicable
- 5. Implementation = Required SDK version indicated by each country's flag. If a number accompanies the flag such as 3.0 as shown above, then that is the minimum version of the SDK that is required for that field.
- 6. Occurrences = 1 implies a required field, 0-1 implies an optional field, and
0-n implies that a record is not required or that multiple records can be used (or 0-many)

Step 3 - Proper Field Order

The proper field order is crucial. If one required field is missing or one field is out of order, then you will encounter errors. The field order is specified from top to bottom in the OSR and must be referenced in the script in that exact order. For Example, in Figure 5.5 when adding an invoice to QuickBooks we must specify the proper field order which is determined by the OSR as shown below.

Figure 5.5 - Sample script showing field order taken directly from the OSR

```
PCQB_RqAddFieldWithValue( "CustomerRef::FullName" ; "Bob Jones" )
PCQB_RqAddFieldWithValue( "TxnDate" ; "2006/01/01" )
PCQB_RqAddFieldWithValue( "RefNumber" ; "123456789" )
PCQB_RqAddFieldWithValue( "BillAddress::Addr1" ; "123 Any Street" )
PCQB_RqAddFieldWithValue( "BillAddress::City" ; "Any Town" )
PCQB_RqAddFieldWithValue( "BillAddress::State" ; "Any State" )
```



5) Tips

Tax Tips:

Tax varies depends on the version of QuickBooks you are using and requires knowledge of how QuickBooks handles tax. Since we do not provide QuickBooks training, we assume that you already have knowledge of how your QuickBooks file handles tax. For example, when using the Canadian version of QuickBooks you will probably want to turn off Sales Tax. When using the US version sales tax will be applicable and should be turned on.

Here are some helpful hints when using the US versions:

- The tax rate is pre-entered in QuickBooks. However you can modify the tax rate from FM to QBs.
- The tax rate is set up at invoice level and not the line item level.
- An item must be specified as taxable or non-taxable. This is set up at the line item level for an invoice or at the item level. It is synonymous.
- If you do not specify the tax rate in QBs, then the QBs file default will be applied.

Canadian Versions:

When using the Canadian version please make these two very important changes.

- Change "State" to "Province"
- Remove or adjust any reference to Sales Tax

UK and Australian Versions:

The Australian versions adhere to the UK version requirements when being referenced throughout our documentation and examples.

6) Tutorial Videos and Sample Code

1) Overview - Installing the plug-in / Connecting to QuickBooks for the 1st Time

Video Tutorial: <http://www.fmbooksconnector.com/videos/Overview/Overview.php?ID=83>

2) Push a Customer from FileMaker Pro to QuickBooks

Video Tutorial: <http://www.fmbooksconnector.com/videos/AddCustomer/AddCustomer.php?ID=83>

Sample Code: <http://www.fmbooksconnector.com/downloads/PushCustomer.pdf>

3) Push an Invoice from FileMaker to QuickBooks:

Video Tutorial: <http://www.fmbooksconnector.com/videos/AddInvoice/AddInvoice.php?ID=83>

Sample Code: <http://www.fmbooksconnector.com/downloads/PushInvoice.pdf>

4) Pull a Customer Balance from QuickBooks into FileMaker

Video Tutorial: <http://www.fmbooksconnector.com/videos/PullBalance/PullBalance.php?ID=83>

Sample Code: <http://www.fmbooksconnector.com/downloads/PullCustomerBalance.pdf>

5) Pull an Invoice Balance from QuickBooks into FileMaker

Sample Code: <http://www.fmbooksconnector.com/downloads/PullInvoiceBalance.pdf>

III. Contact Us

Successful integration of our products within your own system requires a working knowledge of FileMaker, especially in the areas of scripting and calculations. If you need additional support for scripting, customization or setup, then you can contact us via the avenues listed below.

Phone: 760-510-1200

Email: support@productivecomputing.com

Forum: www.productivecomputing.com/forum

However please note that assisting you with implementing this plug-in (excluding registration) is billable at our standard rate. We bill on a time and materials basis so you are only billed for the time it takes to assist you. If you are not a FileMaker Developer or are just too busy to create the integration scripts, then please contact us. We will be happy to provide you with a free estimate to create your integration scripts. We are ready to assist and look forward to hearing from you!